

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.

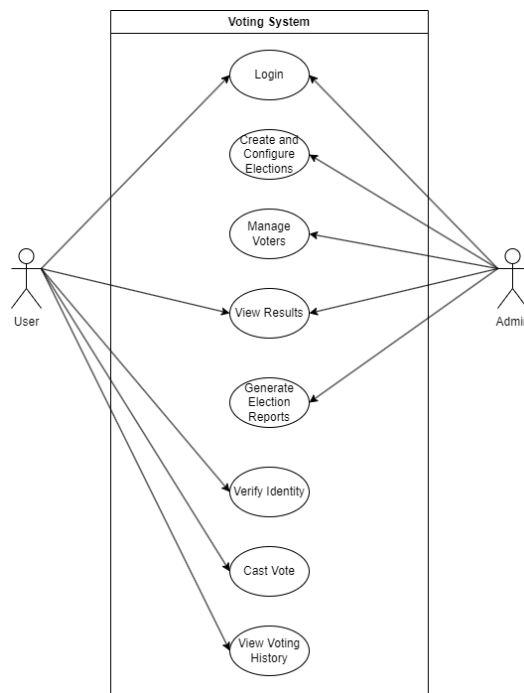


Figure 1.1: Use Case Diagram

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

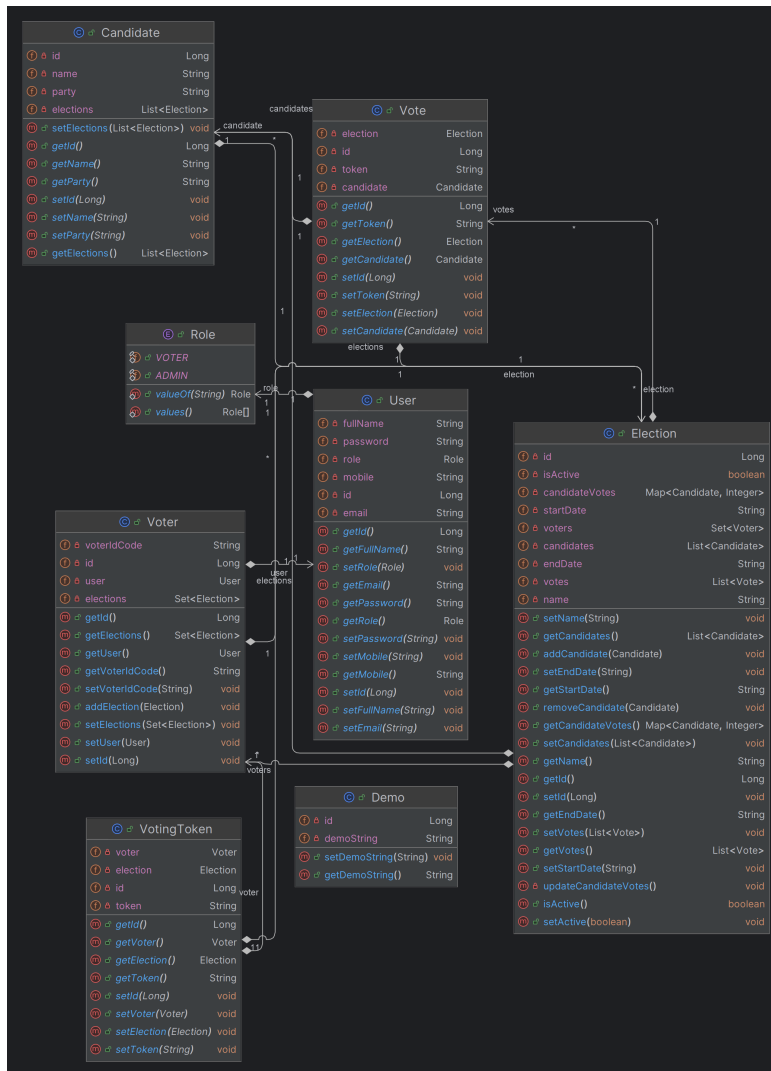


Figure 1.2: Model Layer Class Diagram

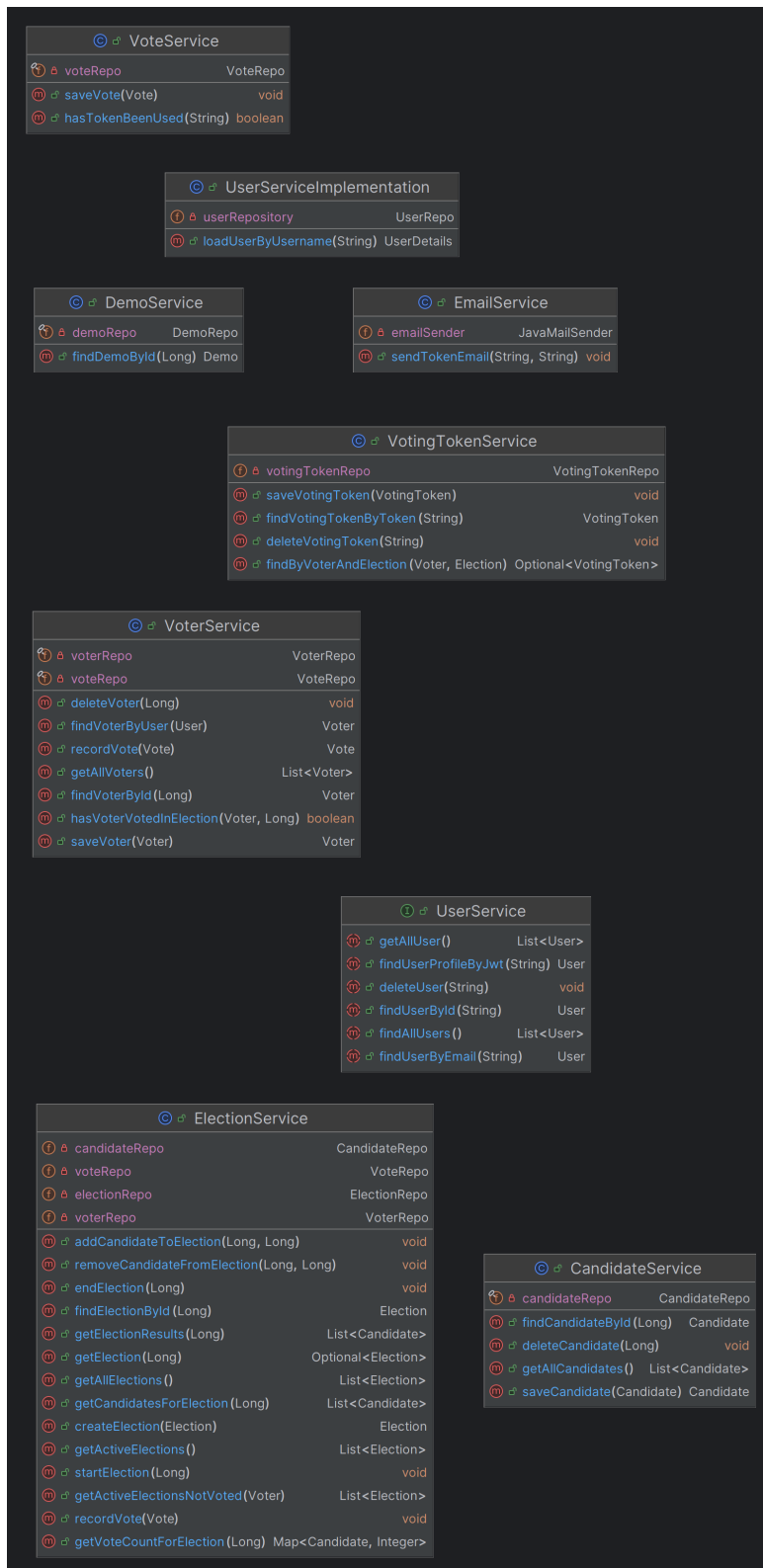


Figure 1.3: Service Layer Class Diagram

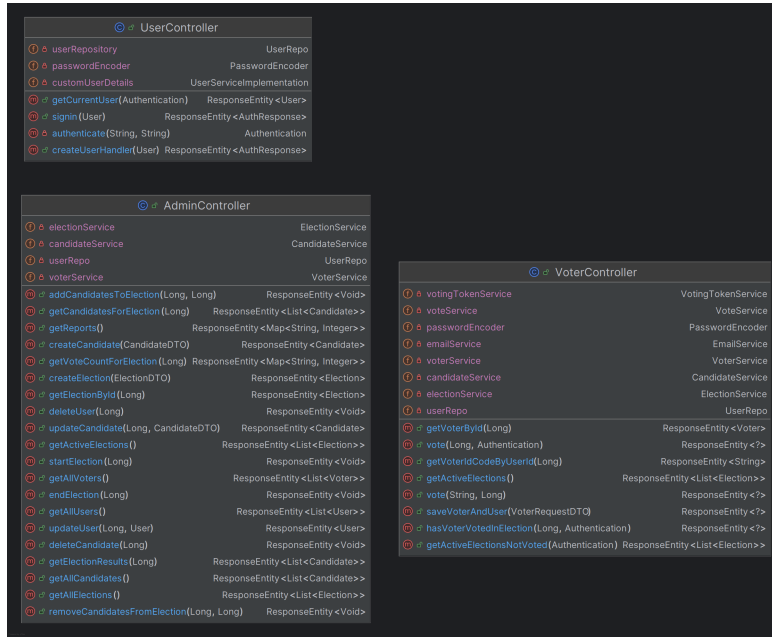


Figure 1.4: Controller Layer Class Diagram

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.

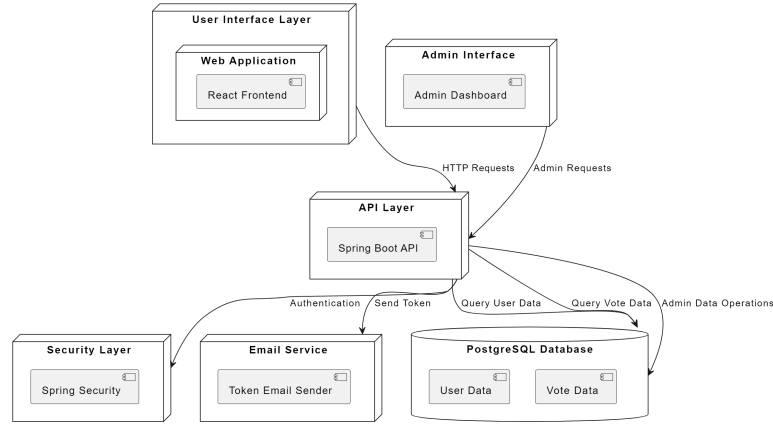


Figure 1.5: Deployment Diagram

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.

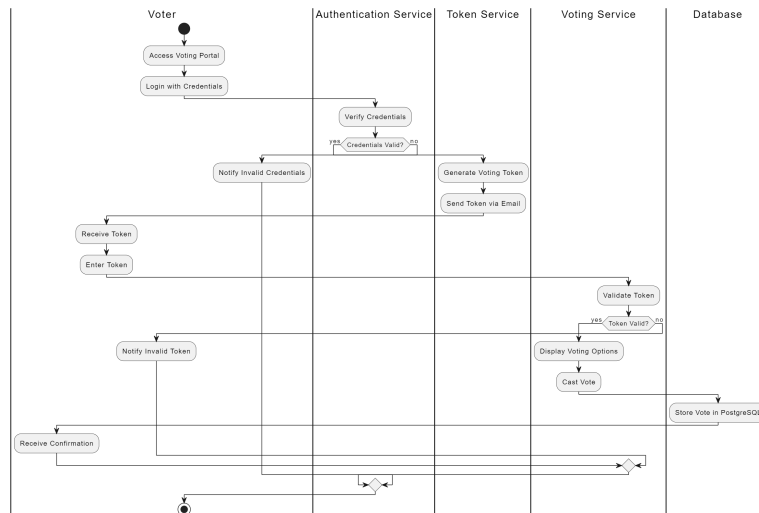


Figure 1.6: Activity Diagram

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

```
1 package com.example.votingSystem.model;
2
3 public enum Role {
4     ADMIN, VOTER
5 }
6
7 package com.example.votingSystem.model;
8
9 import com.fasterxml.jackson.annotation.JsonProperty;
10 import jakarta.persistence.*;
11 import lombok.AllArgsConstructor;
12 import lombok.Getter;
13 import lombok.NoArgsConstructor;
14 import lombok.Setter;
15
16 @Entity
17 @Getter
18 @Setter
19 @NoArgsConstructor
20 @AllArgsConstructor
21 @Table(name = "app_user")
22 public class User {
23     @Id
24     @GeneratedValue(strategy = GenerationType.AUTO,
25         generator = "user_seq")
26     private Long id;
```

```

27     private String fullName;
28     private String email;
29
30     @JsonProperty(access = JsonProperty.Access.
        WRITE_ONLY)
31     private String password;
32
33     @Enumerated(EnumType.STRING)
34     private Role role;
35     private String mobile;
36
37 }
38
39 package com.example.votingSystem.model;
40
41 import com.fasterxml.jackson.annotation.JsonIgnore;
42 import jakarta.persistence.*;
43 import lombok.Getter;
44 import lombok.Setter;
45
46 @Entity
47 @Getter
48 @Setter
49 public class Vote {
50     @Id
51     @GeneratedValue(strategy = GenerationType.AUTO)
52     private Long id;
53
54     private String token;
55
56     @ManyToOne(fetch = FetchType.LAZY)
57     @JoinColumn(name = "election_id")
58     @JsonIgnore // Prevent cyclic reference from the
        Vote to Election direction
59     private Election election;
60
61     @ManyToOne(fetch = FetchType.LAZY)
62     @JoinColumn(name = "candidate_id")
63     private Candidate candidate;
64
65     public Vote() {
66
67

```

```

68
69     public Vote(String token,Candidate candidate,
70                 Election election) {
71         this.token = token;
72         this.candidate = candidate;
73         this.election = election;
74     }
75 }
76
77 package com.example.votingSystem.model;
78
79 import com.fasterxml.jackson.annotation.
80     JsonManagedReference;
81 import jakarta.persistence.*;
82
83 import java.util.HashSet;
84 import java.util.Set;
85
86 @Entity
87 public class Voter {
88     @Id
89     @GeneratedValue(strategy = GenerationType.AUTO)
90     private Long id;
91
92     @OneToOne
93     private User user;
94
95     private String voterIdCode;
96
97     @ManyToMany
98     @JoinTable(
99         name = "voter_election",
100         joinColumns = @JoinColumn(name = "voter_id")
101         ,
102         inverseJoinColumns = @JoinColumn(name = "
103             election_id")
104     )
105     @JsonManagedReference // Prevent circular reference
106         in the Voter to Election direction
107     private Set<Election> elections = new HashSet<>();
108
109     public Voter() {

```

```
106
107     }
108
109     public Voter(User user, String voterIdCode) {
110         this.user = user;
111         this.voterIdCode = voterIdCode;
112     }
113
114     public User getUser() {
115         return user;
116     }
117
118     public void setUser(User user) {
119         this.user = user;
120     }
121
122     public Long getId() {
123         return id;
124     }
125
126     public void setId(Long id) {
127         this.id = id;
128     }
129
130     public String getVoterIdCode() {
131         return voterIdCode;
132     }
133
134     public void setVoterIdCode(String voterIdCode) {
135         this.voterIdCode = voterIdCode;
136     }
137
138     public Set<Election> getElections() {
139         return elections;
140     }
141
142     public void setElections(Set<Election> elections) {
143         this.elections = elections;
144     }
145
146     public void addElection(Election election) {
147         this.elections.add(election);
148     }
```



```

149 }
150
151 package com.example.votingSystem.model;
152
153 import com.example.votingSystem.service.VoteService;
154 import jakarta.persistence.*;
155 import lombok.AllArgsConstructor;
156 import lombok.Getter;
157 import lombok.NoArgsConstructor;
158 import lombok.Setter;
159
160 @Entity
161 @Getter
162 @Setter
163 @AllArgsConstructor
164 @NoArgsConstructor
165 public class VotingToken {
166     @Id
167     @GeneratedValue(strategy = GenerationType.AUTO)
168     private Long id;
169
170     @ManyToOne
171     @JoinColumn(name = "voter_id")
172     private Voter voter;
173
174     @ManyToOne
175     @JoinColumn(name = "election_id")
176     private Election election;
177
178     private String token;
179 }
180
181 package com.example.votingSystem.model;
182
183 import com.fasterxml.jackson.annotation.
    JsonBackReference;
184 import jakarta.persistence.*;
185 import lombok.Getter;
186 import lombok.Setter;
187
188 import java.util.ArrayList;
189 import java.util.List;
190

```

```

191
192 @Entity
193 @Getter
194 @Setter
195 public class Candidate {
196
197     @Id
198     @GeneratedValue(strategy = GenerationType.AUTO)
199     private Long id;
200
201     private String name;
202
203     private String party;
204
205     @ManyToMany(mappedBy = "candidates")
206     @JsonBackReference
207     private List<Election> elections;
208
209     public Candidate() {
210     }
211
212     public Candidate(String name, String party) {
213         this.name = name;
214         this.party = party;
215         this.elections = new ArrayList<>();
216     }
217
218     // Getters and setters for elections
219     public List<Election> getElections() {
220         return elections;
221     }
222
223     public void setElections(List<Election> elections) {
224         this.elections = elections;
225     }
226 }
227
228 package com.example.votingSystem.model;
229
230 import com.fasterxml.jackson.annotation.
    JsonBackReference;
231 import com.fasterxml.jackson.annotation.
    JsonManagedReference;

```

```

232 import jakarta.persistence.*;
233 import java.util.*;
234
235 @Entity
236 public class Election {
237
238     @Id
239     @GeneratedValue(strategy = GenerationType.AUTO)
240     private Long id;
241
242     private String name;
243
244     private String startDate;
245
246     private String endDate;
247
248     private boolean isActive;
249
250     @ManyToMany
251     @JoinTable(
252         name = "candidate_election",
253         joinColumns = @JoinColumn(name = "
254             election_id"),
255         inverseJoinColumns = @JoinColumn(name = "
256             candidate_id")
257     )
258     @JsonManagedReference // Prevent circular reference
259     // in the Election to Candidate direction
260     private List<Candidate> candidates;
261
262     @OneToMany(fetch = FetchType.LAZY)
263     @JoinColumn(name = "election_id")
264     @JsonBackReference // Prevent cyclic reference from
265     // the Election to Vote direction
266     private List<Vote> votes;
267
268     @ManyToMany(mappedBy = "elections")
269     @JsonBackReference // Prevent circular reference in
270     // the Election to Voter direction
271     private Set<Voter> voters = new HashSet<>();

```

```

270
271 // Map to store vote count for each candidate
272 @Transient
273 private Map<Candidate, Integer> candidateVotes;
274
275 public Election() {
276     this.candidates = new ArrayList<>();
277     this.votes = new ArrayList<>();
278     this.candidateVotes = new HashMap<>();
279 }
280
281 public Election(String name, String startDate,
282     String endDate, boolean isActive) {
283     this.name = name;
284     this.startDate = startDate;
285     this.endDate = endDate;
286     this.isActive = isActive;
287     this.candidates = new ArrayList<>();
288     this.votes = new ArrayList<>();
289     this.candidateVotes = new HashMap<>();
290 }
291
292 public Long getId() {
293     return id;
294 }
295
296 public void setId(Long id) {
297     this.id = id;
298 }
299
300 public String getName() {
301     return name;
302 }
303
304 public void setName(String name) {
305     this.name = name;
306 }
307
308 public String getStartDate() {
309     return startDate;
310 }
311
312 public void setStartDate(String startDate) {

```

```

312         this.startDate = startDate;
313     }
314
315     public String getEndDate() {
316         return endDate;
317     }
318
319     public void setEndDate(String endDate) {
320         this.endDate = endDate;
321     }
322
323     public boolean isActive() {
324         return isActive;
325     }
326
327     public void setActive(boolean active) {
328         isActive = active;
329     }
330
331     public List<Candidate> getCandidates() {
332         return candidates;
333     }
334
335     public void setCandidates(List<Candidate> candidates
336         ) {
337         this.candidates = candidates;
338         updateCandidateVotes();
339     }
340
341     public List<Vote> getVotes() {
342         return votes;
343     }
344
345     public void setVotes(List<Vote> votes) {
346         this.votes = votes;
347         updateCandidateVotes();
348     }
349
350     // Method to count votes and update candidateVotes
351     map
352     private void updateCandidateVotes() {
353         candidateVotes.clear();
354         for (Candidate candidate : candidates) {

```

```

353         candidateVotes.put(candidate, 0); //
           Initialize vote count for each candidate
354     }
355
356     for (Vote vote : votes) {
357         Candidate votedCandidate = vote.getCandidate
           ();
358         if (candidateVotes.containsKey(
           votedCandidate)) {
359             candidateVotes.put(votedCandidate,
           candidateVotes.get(votedCandidate) +
           1);
360         }
361     }
362 }
363
364 // Getter for candidate vote count
365 public Map<Candidate, Integer> getCandidateVotes() {
366     return candidateVotes;
367 }
368
369 public void addCandidate(Candidate candidate) {
370     this.candidates.add(candidate);
371     candidateVotes.put(candidate, 0); // Initialize
           vote count when a candidate is added
372 }
373
374 public void removeCandidate(Candidate candidate) {
375     this.candidates.remove(candidate);
376     candidateVotes.remove(candidate); // Remove
           vote count when a candidate is removed
377 }
378 }

```

Listing 2.1: Model Layer Code

```

1 package com.example.votingSystem.repo;
2
3 import com.example.votingSystem.model.Candidate;
4 import org.springframework.data.jpa.repository.
   JpaRepository;
5
6
7 public interface CandidateRepo extends JpaRepository<

```

```

Candidate, Long> {
8 }
9
10 package com.example.votingSystem.repo;
11
12 import com.example.votingSystem.model.Election;
13 import org.springframework.data.jpa.repository.
    JpaRepository;
14 import org.springframework.stereotype.Repository;
15
16 import java.util.List;
17
18 @Repository
19 public interface ElectionRepo extends JpaRepository<
    Election, Long> {
20     List<Election> findAllByIsActive(boolean isActive);
21 }
22
23 package com.example.votingSystem.repo;
24
25 import com.example.votingSystem.model.User;
26 import org.springframework.data.jpa.repository.
    JpaRepository;
27 import org.springframework.stereotype.Repository;
28
29 @Repository
30 public interface UserRepo extends JpaRepository<User,
    Long> {
31     User findByEmail(String email);
32 }
33
34 package com.example.votingSystem.repo;
35
36 import com.example.votingSystem.model.Election;
37 import com.example.votingSystem.model.Vote;
38 import org.springframework.data.jpa.repository.
    JpaRepository;
39 import org.springframework.stereotype.Repository;
40
41 import java.util.List;
42
43 @Repository
44 public interface VoteRepo extends JpaRepository<Vote,

```

```

45     Long> {
46         List<Vote> findVotesByElection(Election election);
47
48         boolean existsVoteByToken(String token);
49     }
50
51 package com.example.votingSystem.repo;
52
53 import com.example.votingSystem.model.User;
54 import com.example.votingSystem.model.Voter;
55 import org.springframework.data.jpa.repository.
56     JpaRepository;
57 import org.springframework.data.jpa.repository.Query;
58 import org.springframework.stereotype.Repository;
59
60 import java.util.Optional;
61
62 @Repository
63 public interface VoterRepo extends JpaRepository<Voter,
64     Long> {
65     Optional<Voter> findById(Long userId);
66     Voter findByUser(User user);
67     @Query("SELECT CASE WHEN COUNT(v) > 0 THEN TRUE ELSE
68         FALSE END FROM Voter v JOIN v.elections e WHERE
69         v = ?1 AND e.id = ?2")
70     boolean hasVoterVotedInElection(Voter voter, Long
71         electionId);
72 }
73
74 package com.example.votingSystem.repo;
75
76 import com.example.votingSystem.model.Election;
77 import com.example.votingSystem.model.Voter;
78 import com.example.votingSystem.model.VotingToken;
79 import org.springframework.data.jpa.repository.
80     JpaRepository;
81 import org.springframework.stereotype.Repository;
82
83 import java.util.Optional;
84
85 @Repository
86 public interface VotingTokenRepo extends JpaRepository<
87     VotingToken, Long> {

```



```

80     VotingToken findByToken(String token);
81
82     void deleteByToken(String token);
83
84     Optional<VotingToken> findByVoterAndElection(Voter
85         voter, Election election);
86 }

```

Listing 2.2: Repository Layer Code

```

1  package com.example.votingSystem.service;
2
3  import com.example.votingSystem.model.Candidate;
4  import com.example.votingSystem.repo.CandidateRepo;
5  import org.springframework.stereotype.Service;
6
7  import java.util.List;
8
9  @Service
10 public class CandidateService {
11     private final CandidateRepo candidateRepo;
12
13     public CandidateService(CandidateRepo candidateRepo)
14     {
15         this.candidateRepo = candidateRepo;
16     }
17
18     public List<Candidate> getAllCandidates() {
19         return candidateRepo.findAll();
20     }
21
22     public Candidate findCandidateById(Long id) {
23         return candidateRepo.findById(id)
24             .orElseThrow(() -> new RuntimeException(
25                 "Candidate not found"));
26     }
27
28     public Candidate saveCandidate(Candidate candidate)
29     {
30         return candidateRepo.save(candidate);
31     }
32
33     public void deleteCandidate(Long id) {

```

```

31         candidateRepo.deleteById(id);
32     }
33 }
34
35 package com.example.votingSystem.service;
36
37
38 import com.example.votingSystem.model.Candidate;
39 import com.example.votingSystem.model.Election;
40 import com.example.votingSystem.model.Vote;
41 import com.example.votingSystem.model.Voter;
42 import com.example.votingSystem.repo.CandidateRepo;
43 import com.example.votingSystem.repo.ElectionRepo;
44 import com.example.votingSystem.repo.VoteRepo;
45 import com.example.votingSystem.repo.VoterRepo;
46 import org.springframework.beans.factory.annotation.
    Autowired;
47 import org.springframework.stereotype.Service;
48
49 import java.util.*;
50
51 @Service
52 public class ElectionService {
53
54     @Autowired
55     private ElectionRepo electionRepo;
56
57     @Autowired
58     private CandidateRepo candidateRepo;
59
60     @Autowired
61     private VoteRepo voteRepo;
62     @Autowired
63     private VoterRepo voterRepo;
64
65     // Create a new election
66     public Election createElection(Election election) {
67         return electionRepo.save(election);
68     }
69
70     // Get an election by its ID
71     public Optional<Election> getElection(Long
        electionId) {

```

```

72         return electionRepo.findById(electionId);
73     }
74
75     // Get all elections
76     public List<Election> getAllElections() {
77         return electionRepo.findAll();
78     }
79
80     // Add candidates to an election
81     public void addCandidateToElection(Long electionId,
82         Long candidateId) {
83         Optional<Election> electionOptional =
84             electionRepo.findById(electionId);
85         if (electionOptional.isPresent()) {
86             Election election = electionOptional.get();
87             election.addCandidate(candidateRepo.findById(
88                 candidateId).orElseThrow());
89             electionRepo.save(election);
90         }
91     }
92
93     // Record a vote
94     public void recordVote(Vote vote) {
95         voteRepo.save(vote);
96     }
97
98     // Get vote count for each candidate in a given
99     // election
100     public Map<Candidate, Integer>
101     getVoteCountForElection(Long electionId) {
102         Optional<Election> electionOptional =
103             electionRepo.findById(electionId);
104         if (electionOptional.isPresent()) {
105             Election election = electionOptional.get();
106             Map<Candidate, Integer> voteCountMap = new
107                 HashMap<>();
108
109             // Iterate through the votes and count the
110             // votes for each candidate
111             List<Vote> votes = voteRepo.
112                 findVotesByElection(election);
113             for (Vote vote : votes) {

```

```

106         Candidate candidate = vote.getCandidate
107             ();
108         voteCountMap.put(candidate, voteCountMap
109             .getOrDefault(candidate, 0) + 1);
110     }
111     return voteCountMap;
112 }
113
114 // Get sorted results for candidates in a given
115 // election by vote count
116 public List<Candidate> getElectionResults(Long
117     electionId) {
118     Map<Candidate, Integer> voteCountMap =
119         getVoteCountForElection(electionId);
120     if (voteCountMap != null) {
121         return voteCountMap.entrySet().stream()
122             .sorted((entry1, entry2) -> entry2.
123                 getValue() - entry1.getValue())
124             // Sort by vote count in
125             // descending order
126             .map(Map.Entry::getKey)
127             .toList();
128     }
129     return null;
130 }
131
132 // Get active elections
133 public List<Election> getActiveElections() {
134     return electionRepo.findAllByIsActive(true);
135 }
136
137 public List<Candidate> getCandidatesForElection(Long
138     electionId) {
139     Optional<Election> electionOptional =
140         electionRepo.findById(electionId);
141     if (electionOptional.isPresent()) {
142         Election election = electionOptional.get();
143         return election.getCandidates();
144     }
145     return null;
146 }

```

```

139
140     public Election findElectionById(Long electionId) {
141         return electionRepo.findById(electionId)
142             .orElseThrow(() -> new RuntimeException(
143                 "Election not found"));
144     }
145
146     public List<Election> getActiveElectionsNotVoted(
147         Voter voter) {
148         List<Election> activeElections = electionRepo.
149             findAllByIsActive(true);
150         Iterator<Election> iterator = activeElections.
151             iterator();
152         while (iterator.hasNext()) {
153             Election election = iterator.next();
154             if (voterRepo.hasVoterVotedInElection(voter,
155                 election.getId())) {
156                 iterator.remove();
157             }
158         }
159         return activeElections;
160     }
161
162     public void removeCandidateFromElection(Long
163         electionId, Long candidateId) {
164         Optional<Election> electionOptional =
165             electionRepo.findById(electionId);
166         if (electionOptional.isPresent()) {
167             Election election = electionOptional.get();
168             election.removeCandidate(candidateRepo.
169                 findById(candidateId).orElseThrow());
170             electionRepo.save(election);
171         }
172     }
173
174     public void startElection(Long electionId) {
175         Optional<Election> electionOptional =
176             electionRepo.findById(electionId);
177         if (electionOptional.isPresent()) {
178             Election election = electionOptional.get();
179             election.setActive(true);
180             electionRepo.save(election);
181         }
182     }

```

```

173     }
174
175     public void endElection(Long electionId) {
176         Optional<Election> electionOptional =
177             electionRepo.findById(electionId);
178         if (electionOptional.isPresent()) {
179             Election election = electionOptional.get();
180             election.setActive(false);
181             electionRepo.save(election);
182         }
183     }
184
185     package com.example.votingSystem.service;
186
187     import org.springframework.beans.factory.annotation.
188         Autowired;
189     import org.springframework.mail.SimpleMailMessage;
190     import org.springframework.mail.javamail.JavaMailSender;
191     import org.springframework.stereotype.Service;
192
193     @Service
194     public class EmailService {
195
196         @Autowired
197         private JavaMailSender emailSender;
198
199         public void sendTokenEmail(String to, String token)
200         {
201             SimpleMailMessage message = new
202                 SimpleMailMessage();
203             message.setTo(to);
204             message.setSubject("Your Voting Token");
205             message.setText("Hello,\n\n" +
206                 "Here is your voting token:" + token +
207                 "\n\n" +
208                 "Please use this token to cast your vote\n\n" +
209                 "Thank you.");
210
211             emailSender.send(message);
212         }
213     }

```

```

210
211 package com.example.votingSystem.service;
212
213 import com.example.votingSystem.model.User;
214
215 import java.util.List;
216
217
218 public interface UserService {
219
220
221     List<User> getAllUser() ;
222
223     User findUserProfileByJwt(String jwt);
224
225     User findUserByEmail(String email);
226
227     User findUserById(String userId) ;
228
229     List<User> findAllUsers();
230
231     void deleteUser(String userId) ;
232
233
234 }
235
236 package com.example.votingSystem.service;
237
238 import com.example.votingSystem.model.User;
239 import com.example.votingSystem.repo.UserRepo;
240 import org.springframework.beans.factory.annotation.
    Autowired;
241 import org.springframework.security.core.
    GrantedAuthority;
242 import org.springframework.security.core.userdetails.
    UserDetails;
243 import org.springframework.security.core.userdetails.
    UserDetailsService;
244 import org.springframework.security.core.userdetails.
    UsernameNotFoundException;
245 import org.springframework.stereotype.Service;
246
247

```

```

248 import java.util.ArrayList;
249 import java.util.List;
250 import java.util.Optional;
251
252
253 @Service
254 public class UserServiceImplementation implements
    UserDetailsService {
255
256     @Autowired
257     private UserRepo userRepository;
258
259     public UserServiceImplementation(UserRepo
        userRepository) {
260         this.userRepository=userRepository;
261     }
262
263
264     @Override
265     public UserDetails loadUserByUsername(String
        username) throws UsernameNotFoundException {
266         User user = userRepository.findByEmail(username)
            ;
267         System.out.println(user);
268
269         if(user==null) {
270             throw new UsernameNotFoundException("User
                not found with this email"+username);
271         }
272
273
274         System.out.println("Loaded user: " + user.
            getEmail() + ", Role: " + user.getRole());
275         List<GrantedAuthority> authorities = new
            ArrayList<>();
276         return new org.springframework.security.core.
            userdetails.User(
277             user.getEmail(),
278             user.getPassword(),
279             authorities);
280     }
281 }
282

```



```

283 package com.example.votingSystem.service;
284
285 import com.example.votingSystem.model.User;
286 import com.example.votingSystem.model.Vote;
287 import com.example.votingSystem.model.Voter;
288 import com.example.votingSystem.repo.VoteRepo;
289 import com.example.votingSystem.repo.VoterRepo;
290 import org.springframework.stereotype.Service;
291
292 import java.util.List;
293
294 @Service
295 public class VoterService {
296     private final VoterRepo voterRepo;
297     private final VoteRepo voteRepo;
298
299     public VoterService(VoterRepo voterRepo, VoteRepo
        voteRepo) {
300         this.voterRepo = voterRepo;
301         this.voteRepo = voteRepo;
302     }
303
304     public Voter findVoterById(Long id) {
305         return voterRepo.findById(id)
306             .orElseThrow(() -> new RuntimeException(
307                 "Voter not found"));
308     }
309
310     public Voter saveVoter(Voter voter) {
311         return voterRepo.save(voter);
312     }
313
314     public Voter findVoterByUser(User user) {
315         return voterRepo.findByUser(user);
316     }
317
318     public boolean hasVoterVotedInElection(Voter voter,
        Long electionId) {
319         return voterRepo.hasVoterVotedInElection(voter,
        electionId);
320     }
321

```

```

322     public Vote recordVote(Vote vote) {
323         return voteRepo.save(vote);
324     }
325
326
327     public List<Voter> getAllVoters() {
328         return voterRepo.findAll();
329     }
330
331     public void deleteVoter(Long id) {
332         voterRepo.deleteById(id);
333     }
334 }
335
336 package com.example.votingSystem.service;
337
338 import com.example.votingSystem.repo.VoteRepo;
339 import org.springframework.stereotype.Service;
340
341
342 @Service
343 public class VoteService {
344     private final VoteRepo voteRepo;
345
346     public VoteService(VoteRepo voteRepo) {
347         this.voteRepo = voteRepo;
348     }
349
350     public boolean hasTokenBeenUsed(String token) {
351         return voteRepo.existsVoteByToken(token);
352     }
353 }
354
355 package com.example.votingSystem.service;
356
357 import com.example.votingSystem.model.Election;
358 import com.example.votingSystem.model.Voter;
359 import com.example.votingSystem.model.VotingToken;
360 import com.example.votingSystem.repo.VotingTokenRepo;
361 import org.springframework.beans.factory.annotation.
    Autowired;
362 import org.springframework.stereotype.Service;
363

```

```

364 import java.util.Optional;
365
366 @Service
367 public class VotingTokenService {
368     @Autowired
369     private VotingTokenRepo votingTokenRepo;
370
371     public void saveVotingToken(VotingToken votingToken)
372     {
373         votingTokenRepo.save(votingToken);
374     }
375
376     public VotingToken findVotingTokenByToken(String
377         token) {
378         return votingTokenRepo.findByToken(token);
379     }
380
381     public void deleteVotingToken(String token) {
382         votingTokenRepo.deleteByToken(token);
383     }
384
385     public Optional<VotingToken> findByVoterAndElection(
386         Voter voter, Election election) {
387         return votingTokenRepo.findByVoterAndElection(
388             voter, election);
389     }
390 }

```

Listing 2.3: Service Layer Code

```

1 package com.example.votingSystem.controller;
2
3 import com.example.votingSystem.dto.CandidateDTO;
4 import com.example.votingSystem.dto.ElectionDTO;
5 import com.example.votingSystem.model.*;
6 import com.example.votingSystem.repo.UserRepo;
7 import com.example.votingSystem.service.ElectionService;
8 import com.example.votingSystem.service.CandidateService
9 ;
10 import com.example.votingSystem.service.VoterService;
11 import org.springframework.beans.factory.annotation.
12     Autowired;
13 import org.springframework.http.ResponseEntity;
14 import org.springframework.web.bind.annotation.*;

```

```

13
14 import java.util.HashMap;
15 import java.util.List;
16 import java.util.Map;
17
18 @RestController
19 @RequestMapping("/api/admin")
20 public class AdminController {
21
22     @Autowired
23     private ElectionService electionService;
24
25     @Autowired
26     private CandidateService candidateService;
27
28     @Autowired
29     private VoterService voterService;
30
31     @Autowired
32     private UserRepo userRepo;
33
34
35     // Create a new candidate
36     @PostMapping("/candidates/create")
37     public ResponseEntity<Candidate> createCandidate(
38         @RequestBody CandidateDTO candidateDTO) {
39         Candidate candidate = new Candidate();
40         candidate.setName(candidateDTO.getName());
41         candidate.setParty(candidateDTO.getParty());
42         Candidate createdCandidate = candidateService.
43             saveCandidate(candidate);
44         return ResponseEntity.ok(createdCandidate);
45     }
46
47     // Get all candidates
48     @GetMapping("/candidates")
49     public ResponseEntity<List<Candidate>>
50         getAllCandidates() {
51         List<Candidate> candidates = candidateService.
52             getAllCandidates();
53         return ResponseEntity.ok(candidates);
54     }
55

```

```

52 //Update a candidate
53 @PutMapping("/candidates/{id}")
54 public ResponseEntity<Candidate> updateCandidate(
    @PathVariable Long id, @RequestBody CandidateDTO
    candidateDTO) {
55     Candidate candidate = new Candidate();
56     candidate.setId(id);
57     candidate.setName(candidateDTO.getName());
58     candidate.setParty(candidateDTO.getParty());
59     Candidate updatedCandidate = candidateService.
        saveCandidate(candidate);
60     return ResponseEntity.ok(updatedCandidate);
61 }
62
63 // Delete a candidate
64 @DeleteMapping("/candidates/{id}")
65 public ResponseEntity<Void> deleteCandidate(
    @PathVariable Long id) {
66     candidateService.deleteCandidate(id);
67     return ResponseEntity.ok().build();
68 }
69
70 // Create a new election
71 @PostMapping("/elections/create")
72 public ResponseEntity<Election> createElection(
    @RequestBody ElectionDTO electionDTO) {
73     Election election = new Election();
74     election.setName(electionDTO.getName());
75     election.setActive(electionDTO.isActive());
76     election.setStartDate(electionDTO.getStartDate()
77 );
78     election.setEndDate(electionDTO.getEndDate());
79     Election createdElection = electionService.
        createElection(election);
80     return ResponseEntity.ok(createdElection);
81 }
82
83 // Get all elections
84 @GetMapping("/elections")
85 public ResponseEntity<List<Election>>
    getAllElections() {
    List<Election> elections = electionService.
        getAllElections();

```

```

86         return new ResponseEntity<>(elections, org.
            springframework.http.HttpStatus.OK);
87     }
88
89     // Get election by ID
90     @GetMapping("/elections/{id}")
91     public ResponseEntity<Election> getElectionById(
92         @PathVariable Long id) {
93         return electionService.getElection(id)
94             .map(ResponseEntity::ok)
95             .orElseGet(() -> ResponseEntity.notFound()
96                 .build());
97     }
98
99     // Get active elections
100    @GetMapping("/elections/active")
101    public ResponseEntity<List<Election>>
102        getActiveElections() {
103        List<Election> activeElections = electionService
104            .getActiveElections();
105        return ResponseEntity.ok(activeElections);
106    }
107
108    // Add candidates to an election
109    @PostMapping("/elections/{electionId}/candidates/{
110        candidateId}")
111    public ResponseEntity<Void> addCandidatesToElection(
112        @PathVariable Long electionId, @PathVariable Long
113        candidateId) {
114        electionService.addCandidateToElection(
115            electionId, candidateId);
116        return ResponseEntity.ok().build();
117    }
118
119    // Remove candidates from an election
120    @DeleteMapping("/elections/{electionId}/candidates/{
121        candidateId}")
122    public ResponseEntity<Void>
123        removeCandidatesFromElection(@PathVariable Long
124            electionId, @PathVariable Long candidateId) {
125        electionService.removeCandidateFromElection(
126            electionId, candidateId);
127        return ResponseEntity.ok().build();
128    }

```

```

116     }
117
118     // Get candidates for an election
119     @GetMapping("/elections/{electionId}/candidates")
120     public ResponseEntity<List<Candidate>>
121         getCandidatesForElection(@PathVariable Long
122             electionId) {
123         List<Candidate> candidates = electionService.
124             getCandidatesForElection(electionId);
125         return ResponseEntity.ok(candidates);
126     }
127
128     // Get the vote count for each candidate in a given
129     // election
130     @GetMapping("/elections/{electionId}/vote-counts")
131     public ResponseEntity<Map<String, Integer>>
132         getVoteCountForElection(@PathVariable Long
133             electionId) {
134         Map<Candidate, Integer> voteCounts =
135             electionService.getVoteCountForElection(
136                 electionId);
137         Map<String, Integer> formattedVoteCounts = new
138             HashMap<>();
139         for (Map.Entry<Candidate, Integer> entry :
140             voteCounts.entrySet()) {
141             formattedVoteCounts.put(entry.getKey().
142                 getName() + " (" + entry.getKey().
143                 getParty() + ")", entry.getValue());
144         }
145         return ResponseEntity.ok(formattedVoteCounts);
146     }
147
148     // Get the election results (sorted by vote count)
149     @GetMapping("/elections/{electionId}/results")
150     public ResponseEntity<List<Candidate>>
151         getElectionResults(@PathVariable Long electionId)
152         {
153         List<Candidate> sortedCandidates =
154             electionService.getElectionResults(electionId
155             );
156         return ResponseEntity.ok(sortedCandidates);
157     }
158 }

```

```

143 // Start an election
144 @PutMapping("/elections/{electionId}/start")
145 public ResponseEntity<Void> startElection(
146     @PathVariable Long electionId) {
147     electionService.startElection(electionId);
148     return ResponseEntity.ok().build();
149 }
150
151 // End an election
152 @PutMapping("/elections/{electionId}/end")
153 public ResponseEntity<Void> endElection(
154     @PathVariable Long electionId) {
155     electionService.endElection(electionId);
156     return ResponseEntity.ok().build();
157 }
158
159 // Get all voters
160 @GetMapping("/voters")
161 public ResponseEntity<List<Voter>> getAllVoters() {
162     List<Voter> voters = voterService.getAllVoters();
163     ;
164     return ResponseEntity.ok(voters);
165 }
166
167 // Get all users
168 @GetMapping("/users")
169 public ResponseEntity<List<User>> getAllUsers() {
170     List<User> users = userRepo.findAll();
171     return ResponseEntity.ok(users);
172 }
173
174 // Update a user
175 @PutMapping("/users/{id}")
176 public ResponseEntity<User> updateUser(@PathVariable
177     Long id, @RequestBody User user) {
178     user.setId(id);
179     User updatedUser = userRepo.save(user);
180     return ResponseEntity.ok(updatedUser);
181 }
182
183 // Delete a user
184 @DeleteMapping("/users/{id}")
185 public ResponseEntity<Void> deleteUser(@PathVariable

```



```

182         Long id) {
183             User user = userRepo.findById(id).orElseThrow();
184             if (user.getRole() == Role.VOTER) {
185                 Voter voter = voterService.findVoterByUser(
186                     user);
187                 voterService.deleteVoter(voter.getId());
188             }
189             userRepo.deleteById(id);
190             return ResponseEntity.ok().build();
191         }
192     }
193
194     package com.example.votingSystem.controller;
195
196     import com.example.votingSystem.config.JwtProvider;
197     import com.example.votingSystem.model.Role;
198     import com.example.votingSystem.model.User;
199     import com.example.votingSystem.repo.UserRepo;
200     import com.example.votingSystem.response.AuthResponse;
201     import com.example.votingSystem.service.UserService;
202     import com.example.votingSystem.service.
203         UserServiceImplementation;
204     import org.springframework.beans.factory.annotation.
205         Autowired;
206     import org.springframework.http.HttpStatus;
207     import org.springframework.http.ResponseEntity;
208     import org.springframework.security.authentication.
209         BadCredentialsException;
210     import org.springframework.security.authentication.
211         UsernamePasswordAuthenticationToken;
212     import org.springframework.security.core.Authentication;
213     import org.springframework.security.core.context.
214         SecurityContextHolder;
215
216     import org.springframework.security.core.userdetails.
217         UserDetails;
218     import org.springframework.security.crypto.password.
219         PasswordEncoder;
220     import org.springframework.web.bind.annotation.*;
221
222     @RestController
223     @RequestMapping("/api/auth")

```

```

216 public class UserController {
217
218     @Autowired
219     private UserRepo userRepository;
220     @Autowired
221     private PasswordEncoder passwordEncoder;
222
223
224     @Autowired
225     private UserServiceImplementation customUserDetails;
226
227
228     @PostMapping("/signup")
229     public ResponseEntity<AuthResponse>
230         createUserHandler(@RequestBody User user) {
231         String email = user.getEmail();
232         String password = user.getPassword();
233         String fullName = user.getFullName();
234         String mobile = user.getMobile();
235         String role = String.valueOf(user.getRole());
236
237         User existingUser = userRepository.findByEmail(
238             email);
239         if(existingUser != null) {
240             throw new BadCredentialsException("User
241                 already exists");
242         }
243
244         User createdUser = new User();
245         createdUser.setEmail(email);
246         createdUser.setFullName(fullName);
247         createdUser.setMobile(mobile);
248         createdUser.setRole(Role.valueOf(role));
249         createdUser.setPassword(passwordEncoder.encode(
250             password));
251
252         User savedUser = userRepository.save(createdUser
253             );
254         userRepository.save(savedUser);
255         Authentication authentication = new
256             UsernamePasswordAuthenticationToken(email,
257                 password);
258         SecurityContextHolder.getContext().

```

```

252         setAuthentication(authentication);
String token = JwtProvider.generateToken(
        authentication);
253
254
255     AuthResponse authResponse = new AuthResponse();
256     authResponse.setJwt(token);
257     authResponse.setMessage("Register_␣Success");
258     authResponse.setStatus(true);
259     return new ResponseEntity<AuthResponse>(
        authResponse, HttpStatus.OK);
260 }
261
262
263
264
265
266 @PostMapping("/signin")
267 public ResponseEntity<AuthResponse> signin(
    @RequestBody User loginRequest) {
268     String username = loginRequest.getEmail();
269     String password = loginRequest.getPassword();
270
271     System.out.println(username+"-----"+password);
272
273     Authentication authentication = authenticate(
        username,password);
274     SecurityContextHolder.getContext().
        setAuthentication(authentication);
275
276     String token = JwtProvider.generateToken(
        authentication);
277     AuthResponse authResponse = new AuthResponse();
278
279     authResponse.setMessage("Login_␣success");
280     authResponse.setJwt(token);
281     authResponse.setStatus(true);
282
283     return new ResponseEntity<>(authResponse,
        HttpStatus.OK);
284 }
285
286

```

```

287
288
289     private Authentication authenticate(String username,
290                                       String password) {
291
292         System.out.println(username+"---++---"+password
293                             );
294
295         UserDetails userDetails = customUserDetails.
296             loadUserByUsername(username);
297
298         System.out.println("Sign_in_in_user_details"+
299                             userDetails);
300
301         if(userDetails == null) {
302             System.out.println("Sign_in_details_-null"
303                                 + userDetails);
304
305             throw new BadCredentialsException("Invalid_
306                                                 username_and_password");
307         }
308         if(!passwordEncoder.matches(password,userDetails
309                                     .getPassword())) {
310             System.out.println("Sign_in_userDetails_-
311                                 password_mismatch"+userDetails);
312
313             throw new BadCredentialsException("Invalid_
314                                                 password");
315         }
316
317         return new UsernamePasswordAuthenticationToken(
318             userDetails,null,userDetails.getAuthorities()
319         );
320     }
321
322     @GetMapping("/me")
323     public ResponseEntity<User> getCurrentUser(
324         Authentication authentication) {
325         if (authentication == null || !authentication.
326             isAuthenticated()) {
327             return new ResponseEntity<>(HttpStatus.
328                 UNAUTHORIZED);
329         }
330     }

```

```

316     }
317
318     String email = authentication.getName(); //
        Retrieves the username/email from the
        authenticated context
319     User user = userRepository.findByEmail(email);
320     if (user == null) {
321         return new ResponseEntity<>(HttpStatus.
            NOT_FOUND);
322     }
323
324     return new ResponseEntity<>(user, HttpStatus.OK)
        ;
325 }
326 }
327
328 package com.example.votingSystem.controller;
329
330 import com.example.votingSystem.dto.VoterRequestDTO;
331 import com.example.votingSystem.model.*;
332 import com.example.votingSystem.repo.UserRepo;
333 import com.example.votingSystem.service.*;
334 import jakarta.transaction.Transactional;
335 import org.springframework.beans.factory.annotation.
    Autowired;
336 import org.springframework.http.HttpStatus;
337 import org.springframework.http.ResponseEntity;
338 import org.springframework.security.core.Authentication;
339 import org.springframework.security.crypto.password.
    PasswordEncoder;
340 import org.springframework.web.bind.annotation.*;
341
342 import java.util.List;
343 import java.util.Optional;
344 import java.util.UUID;
345
346 @RestController
347 @RequestMapping("/api/voter")
348 public class VoterController {
349     @Autowired
350     private VoterService voterService;
351     @Autowired
352     private ElectionService electionService;

```

```

353     @Autowired
354     private UserRepo userRepo;
355     @Autowired
356     private PasswordEncoder passwordEncoder;
357     @Autowired
358     private CandidateService candidateService;
359     @Autowired
360     private VotingTokenService votingTokenService;
361     @Autowired
362     private EmailService emailService;
363     @Autowired
364     private VoteService voteService;
365
366
367     @GetMapping("/{id}")
368     public ResponseEntity<Voter> getVoterById(
369         @PathVariable Long id) {
370         Voter voter = voterService.findVoterById(id);
371         return new ResponseEntity<>(voter, HttpStatus.OK);
372     }
373
374     @PostMapping("/register")
375     public ResponseEntity<?> saveVoterAndUser(
376         @RequestBody VoterRequestDTO voterRequestDTO) {
377         // Check if user already exists
378         String email = voterRequestDTO.getEmail();
379         User existingUser = userRepo.findByEmail(email);
380         if (existingUser != null) {
381             return new ResponseEntity<>("User already exists", HttpStatus.BAD_REQUEST);
382         }
383
384         // Create new user
385         User user = new User();
386         user.setEmail(voterRequestDTO.getEmail());
387         user.setFullName(voterRequestDTO.getFullName());
388         user.setMobile(voterRequestDTO.getMobile());
389         user.setPassword(passwordEncoder.encode(
390             voterRequestDTO.getPassword())); // Encrypt password
391         user.setRole(Role.VOTER); // Set role to Voter

```

```

390         User savedUser = userRepo.save(user);
391
392         // Create Voter and associate with the newly
           created User
393         Voter voter = new Voter();
394         voter.setUser(savedUser);
395         voter.setVoterIdCode(voterRequestDTO.
           getVoterIdCode());
396
397         Voter savedVoter = voterService.saveVoter(voter)
           ;
398         return new ResponseEntity<>(savedVoter,
           HttpStatus.CREATED);
399     }
400
401     @GetMapping("/voterIdCode/{id}")
402     public ResponseEntity<String> getVoterIdCodeByUserId
       (@PathVariable Long id) {
403         User user = userRepo.findById(id).orElse(null);
404         if (user == null) {
405             return new ResponseEntity<>("User not found"
               , HttpStatus.NOT_FOUND);
406         }
407
408         Voter voter = voterService.findVoterByUser(user)
           ;
409         if (voter == null) {
410             return new ResponseEntity<>("Voter not found"
               , HttpStatus.NOT_FOUND);
411         }
412
413         return new ResponseEntity<>(voter.getVoterIdCode
           (), HttpStatus.OK);
414     }
415
416     @GetMapping("/active-elections")
417     public ResponseEntity<List<Election>>
       getActiveElections() {
418         List<Election> activeElections = electionService
           .getActiveElections();
419         return ResponseEntity.ok(activeElections);
420     }
421

```

```

422 @GetMapping("/active-elections-not-voted")
423 public ResponseEntity<List<Election>>
    getActiveElectionsNotVoted(Authentication
    authentication) {
424     // Check if the user is authenticated
425     if (authentication == null) {
426         return new ResponseEntity<>(HttpStatus.
            UNAUTHORIZED);
427     }
428
429     // Get the User object
430     String email = authentication.getName();
431     User user = userRepo.findByEmail(email);
432
433     if (user == null) {
434         return new ResponseEntity<>(HttpStatus.
            NOT_FOUND);
435     }
436
437     // Ensure the user is a Voter
438     if (user.getRole() != Role.VOTER) {
439         return new ResponseEntity<>(HttpStatus.
            FORBIDDEN);
440     }
441
442     // Get the Voter object associated with the User
443     Voter voter = voterService.findVoterByUser(user)
        ;
444     if (voter == null) {
445         return new ResponseEntity<>(HttpStatus.
            NOT_FOUND);
446     }
447
448     List<Election> activeElections = electionService
        .getActiveElectionsNotVoted(voter);
449     return ResponseEntity.ok(activeElections);
450 }
451
452 @PostMapping("/generate-token/{electionId}")
453 public ResponseEntity<?> vote(@PathVariable Long
    electionId, Authentication authentication) {
454     // Check if the user is authenticated
455     if (authentication == null) {

```



```

456         return new ResponseEntity<>("User not logged
           in", HttpStatus.UNAUTHORIZED);
457     }
458
459     // Get the User object
460     String email = authentication.getName();
461     User user = userRepo.findByEmail(email);
462
463     if (user == null) {
464         return new ResponseEntity<>("User not found"
           , HttpStatus.NOT_FOUND);
465     }
466
467     // Ensure the user is a Voter
468     if (user.getRole() != Role.VOTER) {
469         return new ResponseEntity<>("User is not a
           voter", HttpStatus.FORBIDDEN);
470     }
471
472     // Get the Voter object associated with the User
473     Voter voter = voterService.findVoterByUser(user)
474         ;
475     if (voter == null) {
476         return new ResponseEntity<>("Voter not found
           ", HttpStatus.NOT_FOUND);
477     }
478
479     // Check if the voter has already voted in this
480     // election
481     if (voterService.hasVoterVotedInElection(voter,
482         electionId)) {
483         return new ResponseEntity<>("Voter has
           already voted in this election",
484             HttpStatus.BAD_REQUEST);
485     }
486
487     Election election = electionService.
488         findElectionById(electionId);
489
490     Optional<VotingToken> existingToken =
491         votingTokenService.findByVoterAndElection(
492             voter, election);
493     if (existingToken.isPresent()) {

```

```

487         return new ResponseEntity<>("You have
         already voted or have a pending vote for
         this election", HttpStatus.BAD_REQUEST);
488     }
489
490     String token = UUID.randomUUID().toString();
491
492     VotingToken votingToken = new VotingToken();
493     votingToken.setVoter(voter);
494     votingToken.setElection(election);
495     votingToken.setToken(token);
496
497     votingTokenService.saveVotingToken(votingToken);
498
499     emailService.sendTokenEmail(voter.getUser().
        getEmail(), token);
500
501     return new ResponseEntity<>("Token generated,
        please confirm your vote.", HttpStatus.OK);
502 }
503
504 @Transactional
505 @PostMapping("/vote/{token}/{candidateId}")
506 public ResponseEntity<?> vote(@PathVariable String
    token, @PathVariable Long candidateId) {
507     VotingToken votingToken = votingTokenService.
        findVotingTokenByToken(token);
508     if (votingToken == null) {
509         return new ResponseEntity<>("Invalid token",
            HttpStatus.BAD_REQUEST);
510     }
511
512     if (voteService.hasTokenBeenUsed(token)) {
513         return new ResponseEntity<>("Token has
            already been used", HttpStatus.
                BAD_REQUEST);
514     }
515
516     Election election = votingToken.getElection();
517
518     Candidate candidate = candidateService.
        findCandidateById(candidateId);
519     if (candidate == null) {

```

```

520         return new ResponseEntity<>("Candidate not
521         found", HttpStatus.NOT_FOUND);
522     }
523     Voter voter = votingToken.getVoter();
524
525     votingTokenService.deleteVotingToken(token);
526
527     Vote vote = new Vote();
528     vote.setCandidate(candidate);
529     vote.setElection(election);
530     vote.setToken(token);
531
532     electionService.recordVote(vote);
533
534     voter.addElection(election);
535     voterService.saveVoter(voter);
536
537     return new ResponseEntity<>("Vote recorded",
538     HttpStatus.OK);
539 }
540
541 @GetMapping("/active-elections/{electionId}")
542 public ResponseEntity<?> hasVoterVotedInElection(
543     @PathVariable Long electionId, Authentication
544     authentication) {
545     // Check if the user is authenticated
546     if (authentication == null) {
547         return new ResponseEntity<>("User not logged
548         in", HttpStatus.UNAUTHORIZED);
549     }
550
551     // Get the User object
552     String email = authentication.getName();
553     User user = userRepo.findByEmail(email);
554
555     if (user == null) {
556         return new ResponseEntity<>("User not found",
557         HttpStatus.NOT_FOUND);
558     }
559
560     // Ensure the user is a Voter
561     if (user.getRole() != Role.VOTER) {

```

```

557         return new ResponseEntity<>("User is not a
           voter", HttpStatus.FORBIDDEN);
558     }
559
560     // Get the Voter object associated with the User
561     Voter voter = voterService.findVoterByUser(user)
562         ;
563     if (voter == null) {
564         return new ResponseEntity<>("Voter not found
           ", HttpStatus.NOT_FOUND);
565     }
566
567     boolean hasVoted = voterService.
568         hasVoterVotedInElection(voter, electionId);
569     return new ResponseEntity<>(hasVoted, HttpStatus
570         .OK);
571 }
572 }

```

Listing 2.4: Controller Layer Code

```

1  package com.example.votingSystem.dto;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Getter;
5  import lombok.NoArgsConstructor;
6  import lombok.Setter;
7
8  @AllArgsConstructor
9  @NoArgsConstructor
10 @Getter
11 @Setter
12 public class CandidateDTO {
13     private String name;
14     private String party;
15 }
16
17 package com.example.votingSystem.dto;
18
19 import lombok.AllArgsConstructor;
20 import lombok.Getter;
21 import lombok.NoArgsConstructor;
22 import lombok.Setter;
23

```

```

24 @AllArgsConstructor
25 @NoArgsConstructor
26 @Getter
27 @Setter
28 public class ElectionDTO {
29     private String name;
30     private boolean isActive;
31     private String startDate;
32     private String endDate;
33
34 }
35
36 package com.example.votingSystem.dto;
37
38 import com.example.votingSystem.model.Role;
39 import lombok.AllArgsConstructor;
40 import lombok.Getter;
41 import lombok.NoArgsConstructor;
42 import lombok.Setter;
43
44 @Getter
45 @Setter
46 @AllArgsConstructor
47 @NoArgsConstructor
48 public class VoterRequestDTO {
49
50     private String email;
51     private String password;
52     private String fullName;
53     private String mobile;
54     private Role role;
55     private String voterIdCode;
56
57 }

```

Listing 2.5: Data Transfer Objects

```

1 package com.example.votingSystem.config;
2
3 import jakarta.servlet.http.HttpServletRequest;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.
    Configuration;
6 import org.springframework.security.config.annotation.

```

```

    web.builders.HttpSecurity;
7  import org.springframework.security.config.http.
    SessionCreationPolicy;
8  import org.springframework.security.crypto.bcrypt.
    BCryptPasswordEncoder;
9  import org.springframework.security.crypto.password.
    PasswordEncoder;
10 import org.springframework.security.web.
    SecurityFilterChain;
11 import org.springframework.security.web.authentication.
    www.BasicAuthenticationFilter;
12 import org.springframework.web.cors.CorsConfiguration;
13 import org.springframework.web.cors.
    CorsConfigurationSource;
14
15 import java.util.Collections;
16 import java.util.List;
17
18 @Configuration
19 public class ApplicationConfig {
20
21     @SuppressWarnings("deprecation")
22     @Bean
23     SecurityFilterChain filterChain(HttpSecurity http)
24         throws Exception {
25         http.sessionManagement(management -> management.
26             sessionCreationPolicy(SessionCreationPolicy.
27                 STATELESS))
28             .authorizeRequests(
29                 authorize -> authorize
30                     .requestMatchers("/api/
31                         voter/register").
32                     permitAll() // Allow
33                         public access to
34                         register endpoint
35                     .requestMatchers("/api/
36                         auth/signin").
37                     permitAll() // Allow
38                         public access to
39                         login endpoint
40                     .requestMatchers("/api
41                         /**").authenticated()
42                     // Secure other API

```

```

30         endpoints
        .anyRequest().permitAll
        ()
31        .addFilterBefore(new JwtTokenValidator()
        , BasicAuthenticationFilter.class)
32        .csrf(csrf -> csrf.disable())
33        .cors(cors -> cors.configurationSource(
        corsConfigurationSource()));
34
35        return http.build();
36    }
37
38    private CorsConfigurationSource
39    corsConfigurationSource() {
40        return new CorsConfigurationSource() {
41            @Override
42            public CorsConfiguration
43            getCorsConfiguration(HttpServletRequest
44            request) {
45                CorsConfiguration ccfg = new
46                CorsConfiguration();
47                ccfg.setAllowedOrigins(List.of("http://
48                localhost:3000"));
49                ccfg.setAllowedMethods(Collections.
50                singletonList("*"));
51                ccfg.setAllowCredentials(true);
52                ccfg.setAllowedHeaders(Collections.
53                singletonList("*"));
54                ccfg.setExposedHeaders(List.of("
55                Authorization"));
56                ccfg.setMaxAge(3600L);
57                return ccfg;
58            }
59        };
60    }
61
62    @Bean
63    PasswordEncoder passwordEncoder() {
64        return new BCryptPasswordEncoder();
65    }
66
67    }
68
69    package com.example.votingSystem.config;

```

```

61
62 public class JwtConstant {
63     public static final String SECRET_KEY = "
        wpembytrwcvnryxksdbqwjebruyGHyudqgwveytrtrCSnwifoesarjbwe
        ";
64     public static final String JWT_HEADER = "
        Authorization";
65 }
66
67 package com.example.votingSystem.config;
68
69 import io.jsonwebtoken.Claims;
70 import io.jsonwebtoken.Jwts;
71 import io.jsonwebtoken.security.Keys;
72 import org.springframework.security.core.Authentication;
73 import org.springframework.security.core.
    GrantedAuthority;
74
75 import javax.crypto.SecretKey;
76 import java.util.Collection;
77 import java.util.Date;
78 import java.util.HashSet;
79 import java.util.Set;
80
81 public class JwtProvider {
82     static SecretKey key = Keys.hmacShaKeyFor(
        JwtConstant.SECRET_KEY.getBytes());
83
84     public static String generateToken(Authentication
        auth) {
85         Collection<? extends GrantedAuthority>
            authorities = auth.getAuthorities();
86         String roles = populateAuthorities(authorities);
87         @SuppressWarnings("deprecation")
88         String jwt = Jwts.builder()
89             .setIssuedAt(new Date())
90             .setExpiration(new Date(new Date().
                getTime()+86400000))
91             .claim("email", auth.getName())
92             .claim("authorities",roles)
93             .signWith(key)
94             .compact();
95         System.out.println("Token for parsing in

```



```

96         JwtProvider:␣" + jwt);
97         return jwt;
98     }
99
100     private static String populateAuthorities(Collection
101     <? extends GrantedAuthority> authorities) {
102         Set<String> auths = new HashSet<>();
103         for(GrantedAuthority authority: authorities) {
104             auths.add(authority.getAuthority());
105         }
106         return String.join(",",auths);
107     }
108
109     @SuppressWarnings("deprecation")
110     public static String getEmailFromJwtToken(String jwt
111     ) {
112         jwt = jwt.substring(7); // Assuming "Bearer " is
113         removed from the token
114         try {
115             //Claims claims=Jwts.parserBuilder().
116             setSigningKey(key).build().parseClaimsJws
117             (jwt).getBody();
118             Claims claims = Jwts.parser().setSigningKey(
119             key).build().parseClaimsJws(jwt).getBody
120             ();
121             String email = String.valueOf(claims.get("
122             email"));
123             System.out.println("Email␣extracted␣from␣JWT
124             :␣" + claims);
125             return email;
126         } catch (Exception e) {
127             System.err.println("Error␣extracting␣email␣
128             from␣JWT:␣" + e.getMessage());
129             e.printStackTrace();
130             return null;
131         }
132     }
133 }
134
135 package com.example.votingSystem.config;

```

```

128
129
130 import io.jsonwebtoken.Claims;
131 import io.jsonwebtoken.Jwts;
132 import io.jsonwebtoken.security.Keys;
133 import jakarta.servlet.FilterChain;
134 import jakarta.servlet.ServletException;
135 import jakarta.servlet.http.HttpServletRequest;
136 import jakarta.servlet.http.HttpServletResponse;
137 import org.springframework.security.authentication.
    BadCredentialsException;
138 import org.springframework.security.authentication.
    UsernamePasswordAuthenticationToken;
139 import org.springframework.security.core.Authentication;
140 import org.springframework.security.core.
    GrantedAuthority;
141 import org.springframework.security.core.authority.
    AuthorityUtils;
142 import org.springframework.security.core.context.
    SecurityContextHolder;
143 import org.springframework.web.filter.
    OncePerRequestFilter;
144
145
146 import javax.crypto.SecretKey;
147 import java.io.IOException;
148 import java.util.List;
149
150 public class JwtTokenValidator extends
    OncePerRequestFilter {
151
152     @Override
153     protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain) throws ServletException,
        IOException {
154         String jwt = request.getHeader(JwtConstant.
            JWT_HEADER);
155         System.out.println("JWT_Token_in_
            JwtTokenValidator:_" + jwt);
156         if (jwt != null && jwt.startsWith("Bearer_")) {
157             jwt = jwt.substring(7);
158

```

```

159         System.out.println("JWT_Token_in_
        JwtTokenValidator:" + jwt);
160     try {
161         SecretKey key = Keys.hmacShaKeyFor(
            JwtConstant.SECRET_KEY.getBytes());
162         @SuppressWarnings("deprecation")
163         Claims claims = Jwts.parser().
            setSigningKey(key).build().
            parseClaimsJws(jwt).getBody();
164         System.out.print(claims);
165
166         String email = String.valueOf(claims.get(
            "email"));
167         System.out.print(email);
168         String authorities = String.valueOf(
            claims.get("authorities"));
169         List<GrantedAuthority> auth =
            AuthorityUtils.
                commaSeparatedStringToAuthorityList(
                    authorities);
170         Authentication authentication = new
            UsernamePasswordAuthenticationToken(
                email, null, auth);
171         SecurityContextHolder.getContext().
            setAuthentication(authentication);
172
173     } catch (Exception e) {
174         throw new BadCredentialsException("
            Invalid_token", e);
175     }
176 }
177
178     filterChain.doFilter(request, response);
179 }
180 }
181
182 package com.example.votingSystem.response;
183 import lombok.Data;
184
185 @Data
186 public class ApiResponse {
187     private String message;
188

```

```

189     private boolean status;
190     public ApiResponse(String string, boolean b) {
191     }
192     public String getMessage() {
193         return message;
194     }
195     public void setMessage(String message) {
196         this.message = message;
197     }
198     public boolean isStatus() {
199         return status;
200     }
201     public void setStatus(boolean status) {
202         this.status = status;
203     }
204 }
205
206 }
207
208 package com.example.votingSystem.response;
209
210
211 import lombok.AllArgsConstructor;
212 import lombok.NoArgsConstructor;
213
214 @AllArgsConstructor
215 @NoArgsConstructor
216 public class AuthResponse {
217     private String jwt;
218     private String message;
219     private Boolean status;
220
221     public String getJwt() {
222         return jwt;
223     }
224
225     public void setJwt(String jwt) {
226         this.jwt = jwt;
227     }
228
229     public String getMessage() {
230         return message;
231     }

```

```

232     public void setMessage(String message) {
233         this.message = message;
234     }
235
236     public Boolean getStatus() {
237         return status;
238     }
239
240     public void setStatus(Boolean status) {
241         this.status = status;
242     }
243 }
244
245
246 package com.example.votingSystem.exceptions;
247
248 import org.springframework.http.HttpStatus;
249 import org.springframework.http.ResponseEntity;
250 import org.springframework.web.bind.annotation.
    ControllerAdvice;
251 import org.springframework.web.bind.annotation.
   ExceptionHandler;
252
253 @ControllerAdvice
254 public class GlobalExceptionHandler {
255     @ExceptionHandler(RuntimeException.class)
256     public ResponseEntity<String> handleRuntimeException
    (RuntimeException ex) {
257         return new ResponseEntity<>(ex.getMessage(),
    HttpStatus.BAD_REQUEST);
258     }
259 }
260
261 package com.example.votingSystem;
262
263 import org.springframework.boot.SpringApplication;
264 import org.springframework.boot.autoconfigure.
    SpringBootApplication;
265
266 @SpringBootApplication
267 public class VotingSystemApplication {
268
269     public static void main(String[] args) {

```

```

270         SpringApplication.run(
                VotingSystemApplication.class, args);
271     }
272
273 }

```

Listing 2.6: Config Code

```

1  spring.application.name=votingSystem
2  spring.datasource.url=jdbc:postgresql://localhost:5432/
   votingSystem
3  spring.datasource.username=postgres
4  spring.datasource.password=admin
5  spring.jpa.show-sql=true
6  spring.jpa.hibernate.ddl-auto=update
7  spring.jpa.properties.hibernate.dialect=org.hibernate.
   dialect.PostgreSQLDialect
8  # Session Management Configuration
9  spring.security.filter.chain.content-negotiation.
   parameter-strategy=ignore
10 spring.security.filter.chain.any-request.authorized=
   permitAll
11 spring.security.filter.chain.request-matcher.path.
   pattern=/api/**
12 spring.security.filter.chain.request-matcher.path.
   authenticated=true
13 # CSRF Configuration
14 spring.security.csrf.disabled=true
15 # CORS Configuration
16 spring.security.cors.configurationSource.allowedOrigins=
   http://localhost:3000
17 spring.security.cors.configurationSource.allowedMethods
   =*
18 spring.security.cors.configurationSource.allowedHeaders
   =*
19 spring.security.cors.configurationSource.
   allowCredentials=true
20 spring.security.cors.configurationSource.exposedHeaders=
   Authorization
21 spring.security.cors.configurationSource.maxAge=3600
22 # Mail Configuration
23 spring.mail.host=smtp.gmail.com
24 spring.mail.port=587
25 spring.mail.username=

```

```

26 spring.mail.password=
27 spring.mail.properties.mail.smtp.auth=true
28 spring.mail.properties.mail.smtp.starttls.enable=true
29 spring.mail.properties.mail.smtp.ssl.trust=smtp.gmail.
    com

```

Listing 2.7: Application Properties

Listing 2.8: View Layer Code

```

1  import React from 'react';
2  import { BrowserRouter as Router, Route, Routes } from '
    react-router-dom';
3  import LoginPage from './components/LoginPage';
4  import SignupPage from './components/SignupPage';
5  import Dashboard from './components/Dashboard';
6  import ProfilePage from './components/ProfilePage';
7  import AdminDashboard from './components/AdminDashboard';
8  import ManageElections from './components/ManageElections'
    ;
9  import ManageCandidates from './components/
    ManageCandidates';
10 import './App.css'; // Import your CSS file here
11 import AdminRoute from './components/AdminRoute';
12 import ManageUsers from './components/ManageUsers';
13
14
15 function App() {
16   return (
17     <div className="App">
18       <Router>
19         <Routes>
20           <Route path="/" element={<LoginPage />} />
21           <Route path="/signup" element={<SignupPage
22             />} />
23           <Route path="/dashboard" element={<
24             Dashboard />} />
25           <Route path="/profile" element={<
26             ProfilePage />} />
27           <Route path="/admin/dashboard" element={<
28             AdminRoute element={<AdminDashboard />}
29             />} />
30           <Route path="/admin/manage-elections"
31             element={<AdminRoute element={<
32               ManageElections />} />} />
33           <Route path="/admin/manage-candidates"
34             element={<AdminRoute element={<
35               ManageCandidates />} />} />
36           <Route path="/admin/manage-users" element

```

```

                                ={<AdminRoute element={<ManageUsers />}
                                />} />
28         </Routes>
29     </Router>
30 </div>
31 );
32 }
33
34 export default App;
35
36 import React, { useState, useEffect } from 'react';
37 import axios from 'axios';
38 import { useNavigate } from 'react-router-dom';
39
40 function AdminDashboard() {
41     const [elections, setElections] = useState([]);
42     const [selectedElectionResults,
43         setSelectedElectionResults] = useState([]);
44     const [selectedElectionId, setSelectedElectionId] =
45         useState(null);
46     const navigate = useNavigate();
47
48     useEffect(() => {
49         // Fetch all elections
50         fetchElections();
51     }, []);
52
53     const fetchElections = async () => {
54         try {
55             const token = localStorage.getItem('token');
56             // Retrieve the token from local storage
57             const response = await axios.get("http://
58                 localhost:8080/api/admin/elections", {
59                 headers: {
60                     'Authorization': `Bearer ${token}` //
61                     Pass JWT token
62                 }
63             });
64             setElections(response.data);
65         } catch (error) {
66             console.error("Error fetching elections:",
67                 error);
68         }
69     };
70
71     const fetchElectionResults = async (electionId) => {
72         try {
73             const token = localStorage.getItem('token');
74             // Retrieve the token from local storage

```



```

68     const response = await axios.get(`http://
        localhost:8080/api/admin/elections/${
        electionId}/vote-counts`, {
69         headers: {
70             'Authorization': `Bearer ${token}` //
                Pass JWT token
71         }
72     });
73     console.log("Election results fetched:",
        response.data);
74     const results = Object.entries(response.data).
        map(([candidate, voteCount]) => ({
75         candidate,
76         voteCount
77     }));
78     setSelectedElectionResults(results);
79 } catch (error) {
80     console.error("Error fetching election
        results:", error);
81     setSelectedElectionResults([]); // Ensure
        selectedElectionResults is an array even if
        there's an error
82 }
83 };
84
85 const handleElectionClick = (electionId) => {
86     setSelectedElectionId(electionId);
87     fetchElectionResults(electionId);
88 };
89
90 return (
91     <div>
92         <nav className="navbar navbar-expand-lg navbar
            -light bg-light">
93             <div className="container-fluid">
94                 <a className="navbar-brand" href="#">
                    Admin Dashboard</a>
95                 <button className="navbar-toggler"
                    type="button" data-bs-toggle="
                    collapse" data-bs-target="#
                    navbarNav" aria-controls="navbarNav
                    " aria-expanded="false" aria-label="
                    Toggle navigation">
96                     <span className="navbar-toggler-
                        icon"></span>
97                 </button>
98                 <div className="collapse navbar-
                    collapse" id="navbarNav">
99                     <ul className="navbar-nav ms-auto"

```

```

100     >
101     <li className="nav-item">
102         <button className="btn btn
103             -link nav-link" onClick
104             ={{() => navigate('/
105                 admin/manage-users')}}>
106             Manage Users</button>
107     </li>
108     <li className="nav-item">
109         <button className="btn btn
110             -link nav-link" onClick
111             ={{() => navigate('/
112                 admin/manage-elections'
113                 )}}>Manage Elections</
114             button>
115     </li>
116     <li className="nav-item">
117         <button className="btn btn
118             -link nav-link" onClick
119             ={{() => navigate('/
120                 admin/manage-candidates
121                 ')}}>Manage Candidates</
122             button>
123     </li>
124     <li className="nav-item">
125         <button className="btn btn
126             -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(

```

```

127         election.id))>
        {election.name} (from {
            election.startDate} to
            {election.endDate})
128     {election.id ===
        selectedElectionId &&
        selectedElectionResults.length
        > 0 && (
129         <ul>
130             {
                selectedElectionResults.map
                ((result, index
                ) => (
131                 <li key={index
                    }>
132                     {
                        result.candidate
                    } - {
                        result.voteCount
                    } votes
133                 </li>
134             )}}
135         </ul>
136     )}
137     </li>
138     )}}
139 </ul>
140 </div>
141 </div>
142 </div>
143 );
144 }
145
146 export default AdminDashboard;
147
148 import React, { useEffect, useState } from 'react';
149 import { Navigate } from 'react-router-dom';
150 import { getCurrentUser } from './UserService';
151
152 function AdminRoute({ element: Component, ...rest }) {
153     const [isAdmin, setIsAdmin] = useState(false);
154     const [loading, setLoading] = useState(true);
155
156     useEffect(() => {
157         const checkUserRole = async () => {
158             try {
159                 const user = await getCurrentUser();
160                 setIsAdmin(user.role === 'ADMIN');
161             } catch (error) {

```

```

162         setIsAdmin(false);
163     } finally {
164         setLoading(false);
165     }
166 };
167
168     checkUserRole();
169 }, []);
170
171     if (loading) {
172         return <div>Loading...</div>; // or a loading
            spinner
173     }
174
175     return isAdmin ? Component : <Navigate to="/dashboard"
        />;
176 }
177
178 export default AdminRoute;
179
180 import React, { useState, useEffect } from 'react';
181 import axios from 'axios';
182 import { useNavigate } from 'react-router-dom';
183
184 function Dashboard() {
185     const [activeElections, setActiveElections] = useState
        ([]);
186     const [filter, setFilter] = useState('all'); // 'all'
        or 'notVoted'
187     const [error, setError] = useState('');
188     const navigate = useNavigate();
189
190     useEffect(() => {
191         // Fetch active elections
192         fetchActiveElections();
193     }, [filter]);
194
195     const fetchActiveElections = async () => {
196         try {
197             const token = localStorage.getItem('token');
198             // Retrieve the token from local storage
199             const endpoint = filter === 'all' ? "http://
                localhost:8080/api/voter/active-elections"
                : "http://localhost:8080/api/voter/active-
                    elections-not-voted";
200             const response = await axios.get(endpoint, {
201                 headers: {
                    'Authorization': `Bearer ${token}` //
                        Pass JWT token

```

```

202         }
203     });
204     console.log("Active elections fetched:",
        response.data);
205     const electionsWithVoteStatus = await
        Promise.all(response.data.map(async (
            election) => {
206         const electionDetails = await axios.get(`
            http://localhost:8080/api/voter/active-
            elections/${election.id}`, {
207             headers: {
208                 'Authorization': `Bearer ${token}`
                // Pass JWT token
209             }
210         });
211         return { ...election, hasVoted:
            electionDetails.data };
212     }));
213     setActiveElections(electionsWithVoteStatus);
214     console.log("Active elections with vote
        status:", electionsWithVoteStatus);
215 } catch (error) {
216     console.error("Error fetching active
        elections:", error);
217     setActiveElections([]); // Ensure
        activeElections is an array even if there's
        an error
218 }
219 };
220
221 const handleVote = async (electionId, candidateId) =>
    {
222     try {
223         const token = localStorage.getItem('token');
                // Retrieve the token from local storage
224         const response = await axios.post(`http://
            localhost:8080/api/voter/generate-token/${
            electionId}`, {}, {
225             headers: {
226                 'Authorization': `Bearer ${token}` //
                Pass JWT token
227             }
228         });
229         alert(response.data);
230         const voteToken = prompt("Enter the vote
            token:");
231         const voteResponse = await axios.post(`http://
            localhost:8080/api/voter/vote/${voteToken}/
            ${candidateId}`, {}, {

```

```

232         headers: {
233             'Authorization': `Bearer ${token}` //
                Pass JWT token
234         }
235     });
236     console.log("Vote successful:",
        voteResponse.data);
237     setError("");
238     alert(voteResponse.data);
239     fetchActiveElections(); // Refresh the
        elections after voting
240 } catch (error) {
241     console.error("Error casting vote:", error);
242     setError("Failed to cast vote.");
243 }
244 };
245
246 const toggleFilter = () => {
247     setFilter(filter === 'all' ? 'notVoted' : 'all');
248 };
249
250 return (
251     <div>
252         <nav className="navbar navbar-expand-lg navbar
            -light bg-light">
253             <div className="container-fluid">
254                 <a className="navbar-brand" href="#">
                    Voter Dashboard</a>
255                 <button className="navbar-toggler"
                    type="button" data-bs-toggle="
                        collapse" data-bs-target="#
                            navbarNav" aria-controls="navbarNav
                                " aria-expanded="false" aria-label=
                                    "Toggle navigation">
256                     <span className="navbar-toggler-
                        icon"></span>
257                 </button>
258                 <div className="collapse navbar-
                    collapse" id="navbarNav">
259                     <ul className="navbar-nav ms-auto"
                        >
260                         <li className="nav-item">
261                             <button className="btn btn
                                -link nav-link" onClick
                                    ={(()) => navigate('/
                                        profile')}>Profile</
                                        button>
262                             <button className="btn btn
                                -link nav-link" onClick

```

```

263         = {() => {
                localStorage.removeItem
                    ('token');
264                 navigate('/');
265             }}>Logout</button>
266         </li>
267     </ul>
268 </div>
269 </div>
270 </nav>
271 <div className="container mt-5">
272     <h1>Active Elections</h1>
273     {error && <p style={{ color: 'red' }}>{
        error}</p>}
274     <button className="btn btn-secondary mb-3"
        onClick={toggleFilter}>
275         {filter === 'all' ? 'Show Elections
            Not Voted' : 'Show All Elections'}
276     </button>
277     <ul className="list-group">
278         {activeElections.map((election) => (
279             <li key={election.id} className="
                list-group-item">
280                 <h3>{election.name} (from {
                    election.startDate} to {
                    election.endDate})</h3>
281                 <ul>
282                     {election.candidates.map((
                        candidate) => (
283                         <li key={candidate.id
                            }>
284                             {candidate.name}
                                ({
                                    candidate.party
                                })
285                             {!
                                election.hasVoted
                                    && (
286                                 <button
                                    className="
                                        btn btn-
                                        primary ml
                                        -3" onClick
                                        = {() =>
                                            handleVote(
                                                election.id
                                                ,
                                                candidate.id
                                            )}>Vote</

```

```

287                                     button>
288                                     }}
289                                     </li>
290                                 )))
291                             </ul>
292                         </li>
293                     )))
294                 </ul>
295             </div>
296         </div>
297     );
298 }
299 export default Dashboard;
300
301 import React, { useState } from 'react';
302 import axios from 'axios';
303 import { useNavigate } from 'react-router-dom';
304 import {
305     MDBContainer,
306     MDBInput,
307     MDBBtn,
308 } from 'mdb-react-ui-kit';
309 import { getCurrentUser } from '../UserService';
310
311 function LoginPage() {
312     const [email, setEmail] = useState('');
313     const [password, setPassword] = useState('');
314     const [error, setError] = useState('');
315     const navigate = useNavigate();
316
317     const handleLogin = async () => {
318         try {
319             if (!email || !password) {
320                 setError('Please enter both email and password.');
```



```

330         login
const userResponse = await getCurrentUser();
    // Await the user data
331 console.log('User:', userResponse);
332
333     // Navigate based on user role
334     if (userResponse.role === 'ADMIN') {
335         navigate('/admin/dashboard');
336     } else {
337         navigate('/dashboard');
338     }
339 } catch (error) {
340     console.error('Login failed:', error.response
        ? error.response.data : error.message);
341     setError('Invalid email or password.');
```

```

342     }
343 };
344 return (
345     <div className="d-flex justify-content-center
        align-items-start vh-100">
346         <div className="border rounded-lg p-4" style
            ={{ maxWidth: '400px', width: '100%' }}>
347             <MDBContainer className="p-3">
348                 <h2 className="mb-4 text-center">Login
                    </h2>
349                 <MDBInput
350                     wrapperClass='mb-4'
351                     placeholder='Email'
352                     id='email'
353                     value={email}
354                     type='email'
355                     onChange={(e) => setEmail(
                        e.target.value)}
356                 />
357                 <MDBInput
358                     wrapperClass='mb-4'
359                     placeholder='Password'
360                     id='password'
361                     type='password'
362                     value={password}
363                     onChange={(e) => setPassword(
                        e.target.value)}
364                 />
365                 {error && <p className="text-danger
                    text-center">{error}</p>}
366                 <MDBBtn
367                     className="mb-4 w-100"
368                     color="primary"
369                     size="lg"

```

```

370             onClick={handleLogin}
371         >
372             Sign In
373         </MDBBtn>
374         <div className="text-center">
375             <p>Not a member? <a href="/signup"
376                 >Register</a></p>
377         </div>
378     </MDBContainer>
379 </div>
380 );
381 }
382
383 export default LoginPage;
384
385 import React, { useState, useEffect } from 'react';
386 import axios from 'axios';
387 import { useNavigate } from 'react-router-dom';
388
389 function ManageCandidates() {
390     const [candidates, setCandidates] = useState([]);
391     const [newCandidateName, setNewCandidateName] =
392         useState("");
393     const [newCandidateParty, setNewCandidateParty] =
394         useState("");
395     const [editCandidateId, setEditCandidateId] = useState
396         (null);
397     const [editCandidateName, setEditCandidateName] =
398         useState("");
399     const [editCandidateParty, setEditCandidateParty] =
400         useState("");
401     const [error, setError] = useState("");
402     const navigate = useNavigate();
403
404     useEffect(() => {
405         // Fetch all candidates
406         fetchCandidates();
407     }, []);
408
409     const fetchCandidates = async () => {
410         try {
411             const token = localStorage.getItem('token');
412             // Retrieve the token from local storage
413             const response = await axios.get("http://
414                 localhost:8080/api/admin/candidates", {
415                 headers: {
416                     'Authorization': `Bearer ${token}` //
417                     Pass JWT token

```

```

410         }
411     });
412     setCandidates(response.data);
413 } catch (error) {
414     console.error("Error fetching candidates:",
415         error);
416 }
417 };
418
419 const handleCreateCandidate = async () => {
420     if (!newCandidateName || !newCandidateParty) {
421         setError("All fields are required.");
422         return;
423     }
424     try {
425         const token = localStorage.getItem('token');
426         // Retrieve the token from local storage
427         const candidateData = {
428             name: newCandidateName,
429             party: newCandidateParty
430         };
431         console.log("Creating candidate:",
432             candidateData);
433         const response = await axios.post("http://
434             localhost:8080/api/admin/candidates/create"
435             , candidateData, {
436                 headers: {
437                     'Authorization': `Bearer ${token}` //
438                     Pass JWT token
439                 }
440             });
441         console.log("Candidate created successfully:",
442             response.data);
443         setCandidates([...candidates, response.data]);
444         setNewCandidateName("");
445         setNewCandidateParty("");
446         setError("");
447     } catch (error) {
448         console.error("Error creating candidate:",
449             error);
450         setError("Failed to create candidate.");
451     }
452 };
453
454 const handleDeleteCandidate = async (candidateId) => {
455     try {
456         const token = localStorage.getItem('token');
457         // Retrieve the token from local storage
458         await axios.delete(`http://localhost:8080/api/

```

```

450         admin/candidates/${candidateId}`, {
451             headers: {
452                 'Authorization': `Bearer ${token}` //
453                 Pass JWT token
454             }
455         });
456         setCandidates(candidates.filter(candidate =>
457             candidate.id !== candidateId));
458     } catch (error) {
459         console.error("Error deleting candidate:",
460             error);
461         alert("Failed to delete candidate. ");
462         setError("Failed to delete candidate.");
463     }
464 };
465
466 const handleEditCandidate = async () => {
467     if (!editCandidateName || !editCandidateParty) {
468         setError("All fields are required.");
469         return;
470     }
471     try {
472         const token = localStorage.getItem('token');
473         // Retrieve the token from local storage
474         const candidateData = {
475             name: editCandidateName,
476             party: editCandidateParty
477         };
478         console.log("Editing candidate:",
479             candidateData);
480         const response = await axios.put(`http://
481             localhost:8080/api/admin/candidates/${
482             editCandidateId}`, candidateData, {
483             headers: {
484                 'Authorization': `Bearer ${token}` //
485                 Pass JWT token
486             }
487         });
488         console.log("Candidate edited successfully:",
489             response.data);
490         setCandidates(candidates.map(candidate =>
491             candidate.id === editCandidateId ?
492             response.data : candidate));
493         setEditCandidateId(null);
494         setEditCandidateName("");
495         setEditCandidateParty("");
496         setError("");
497     } catch (error) {
498         console.error("Error editing candidate:",

```

```

    error);
487     setError("Failed to edit candidate.");
488   }
489 };
490
491 const startEditCandidate = (candidate) => {
492   setEditCandidateId(candidate.id);
493   setEditCandidateName(candidate.name);
494   setEditCandidateParty(candidate.party);
495 };
496
497 return (
498   <div>
499     <nav className="navbar navbar-expand-lg navbar
500       -light bg-light">
501       <div className="container-fluid">
502         <a className="navbar-brand" href="#">
503           Manage Candidates</a>
504         <button className="navbar-toggler"
505           type="button" data-bs-toggle="
506             collapse" data-bs-target="#
507               navbarNav" aria-controls="navbarNav
508                 " aria-expanded="false" aria-label=
509                   "Toggle navigation">
510           <span className="navbar-toggler-
511             icon"></span>
512         </button>
513         <div className="collapse navbar-
514           collapse" id="navbarNav">
515           <ul className="navbar-nav ms-auto"
516             >
517             <li className="nav-item">
518               <button className="btn btn
519                 -link nav-link" onClick
520                   ={(e) => navigate('/
521                     admin/dashboard')}>Back
522                     to Dashboard</button>
523             </li>
524           </ul>
525         </div>
526       </div>
527     </nav>
528     <div className="container mt-5">
529       <h1>Manage Candidates</h1>
530       <div className="mt-4">
531         <h2>Add New Candidate</h2>
532         {error && <p style={{ color: 'red'
533           }}>{error}</p>}
534         <form onSubmit={(e) => {

```

```

e.preventDefault();
handleCreateCandidate(); }>
520 <div className="mb-3">
521   <label className="form-label">
      Candidate Name</label>
522   <input
523     type="text"
524     className="form-control"
525     value={newCandidateName}
526     onChange={(e) =>
      setNewCandidateName(
        e.target.value)}
      />
527   </div>
528   <div className="mb-3">
529     <label className="form-label">
      Party</label>
531     <input
532       type="text"
533       className="form-control"
534       value={newCandidateParty}
535       onChange={(e) =>
        setNewCandidateParty(
          e.target.value)}
      />
536     </div>
537     <button type="submit" className="
538       btn btn-primary">Add Candidate
      </button>
539   </form>
540 </div>
541 <div className="mt-5">
542   <h2>All Candidates</h2>
543   <ul className="list-group">
544     {candidates.map((candidate) => (
545       <li key={candidate.id}
        className="list-group-item
        d-flex justify-content-
        between align-items-center"
        >
546         <span>{candidate.name} ({
          candidate.party}</span>
        >
547         <div>
548           <button className="btn
            btn-secondary btn-
            sm me-2" onClick
            ={} =>
              startEditCandidate(

```

```

        candidate)}}>Edit</
        button>
549     <button className="btn
           btn-danger btn-sm"
           onClick={() =>
             handleDeleteCandidate
             (candidate.id)}>
             Delete</button>
550   </div>
551 </li>
552   )}}
553 </ul>
554 </div>
555 {editCandidateId && (
556   <div className="mt-5">
557     <h2>Edit Candidate</h2>
558     <form onSubmit={(e) => {
559       e.preventDefault();
560       handleEditCandidate(); }}>
561       <div className="mb-3">
562         <label className="form-
563           label">Candidate Name</
564           label>
565         <input
566           type="text"
567           className="form-
568             control"
569           value={
570             editCandidateName}
571           onChange={(e) =>
572             setEditCandidateName
573             (e.target.value)}
574         />
575       </div>
576       <div className="mb-3">
577         <label className="form-
           label">Party</label>
578         <input
579           type="text"
580           className="form-
581             control"
582           value={
583             editCandidateParty}
584           onChange={(e) =>
585             setEditCandidateParty
586             (e.target.value)}
587         />
588       </div>
589       <button type="submit"

```

```

                    className="btn btn-primary"
                    >Save</button>
578         <button type="button"
                    className="btn btn-
                    secondary ms-2" onClick={()
                        => setEditCandidateId(null
                    )}>Cancel</button>
579     </form>
580 </div>
581     })
582 </div>
583 </div>
584 );
585 }
586
587 export default ManageCandidates;
588
589 import React, { useState, useEffect } from 'react';
590 import axios from 'axios';
591 import { useNavigate } from 'react-router-dom';
592
593 function ManageElections() {
594     const [elections, setElections] = useState([]);
595     const [candidates, setCandidates] = useState([]);
596     const [newElectionName, setNewElectionName] = useState
        ('');
597     const [newElectionStartDate, setNewElectionStartDate]
        = useState('');
598     const [newElectionEndDate, setNewElectionEndDate] =
        useState('');
599     const [selectedElectionId, setSelectedElectionId] =
        useState('');
600     const [selectedCandidateId, setSelectedCandidateId] =
        useState('');
601     const [error, setError] = useState('');
602     const navigate = useNavigate();
603
604     useEffect(() => {
605         fetchElections();
606         fetchCandidates();
607     }, []);
608
609     const fetchElections = async () => {
610         try {
611             const token = localStorage.getItem('token');
612             // Retrieve the token from local storage
613             const response = await axios.get("http://
                localhost:8080/api/admin/elections", {
                headers: {

```



```

614         'Authorization': `Bearer ${token}` //
           Pass JWT token
615     }
616   });
617   setElections(response.data);
618 } catch (error) {
619   console.error("Error fetching elections:",
        error);
620 }
621 };
622
623 const fetchCandidates = async () => {
624   try {
625     const token = localStorage.getItem('token');
626     // Retrieve the token from local storage
627     const response = await axios.get("http://
        localhost:8080/api/admin/candidates", {
628       headers: {
629         'Authorization': `Bearer ${token}` //
           Pass JWT token
630       }
631     });
632     setCandidates(response.data);
633   } catch (error) {
634     console.error("Error fetching candidates:",
        error);
635   }
636 };
637
638 const handleCreateElection = async () => {
639   if (!newElectionName || !newElectionStartDate || !
        newElectionEndDate) {
640     setError("All fields are required.");
641     return;
642   }
643   try {
644     const token = localStorage.getItem('token');
645     // Retrieve the token from local storage
646     const electionData = {
647       name: newElectionName,
648       startDate: newElectionStartDate,
649       endDate: newElectionEndDate
650     };
651     console.log("Creating election:", electionData
        );
        const response = await axios.post("http://
        localhost:8080/api/admin/elections/create",
        electionData, {
        headers: {

```

```

652         'Authorization': `Bearer ${token}` //
           Pass JWT token
653     }
654 });
655     console.log("Election created successfully:",
           response.data);
656     setElections([...elections, response.data]);
657     setNewElectionName("");
658     setNewElectionStartDate("");
659     setNewElectionEndDate("");
660     setError("");
661 } catch (error) {
662     console.error("Error creating election:",
           error);
663     setError("Failed to create election.");
664 }
665 };
666
667 const handleAddCandidateToElection = async () => {
668     if (!selectedElectionId || !selectedCandidateId) {
669         setError("All fields are required.");
670         return;
671     }
672     try {
673         const token = localStorage.getItem('token');
674         // Retrieve the token from local storage
675         const response = await axios.post(`http://
           localhost:8080/api/admin/elections/${
           selectedElectionId}/add-candidate`, {
           candidateId: selectedCandidateId }, {
676             headers: {
677                 'Authorization': `Bearer ${token}` //
678                 Pass JWT token
679             }
680         });
681         console.log("Candidate added to election
           successfully:", response.data);
682         fetchElectionCandidates(selectedElectionId);
683         setSelectedCandidateId("");
684         setError("");
685     } catch (error) {
686         console.error("Error adding candidate to
           election:", error);
687         setError("Failed to add candidate to election.
           ");
688     }
689 };
690
691 const handleRemoveCandidateFromElection = async (

```

```

candidateId) => {
690   try {
691     const token = localStorage.getItem('token');
        // Retrieve the token from local storage
692     await axios.delete(`http://localhost:8080/api/
        admin/elections/${selectedElectionId}/
        remove-candidate/${candidateId}`, {
693       headers: {
694         'Authorization': `Bearer ${token}` //
        Pass JWT token
695       }
696     });
697     console.log("Candidate removed from election
        successfully");
698     fetchElectionCandidates(selectedElectionId);
699   } catch (error) {
700     console.error("Error removing candidate from
        election:", error);
701     setError("Failed to remove candidate from
        election.");
702   }
703 };
704
705 const handleStartElection = async (electionId) => {
706   try {
707     const token = localStorage.getItem('token');
        // Retrieve the token from local storage
708     await axios.put(`http://localhost:8080/api/
        admin/elections/${electionId}/start`, {}, {
709       headers: {
710         'Authorization': `Bearer ${token}` //
        Pass JWT token
711       }
712     });
713     console.log("Election started successfully");
714     fetchElections();
715   } catch (error) {
716     console.error("Error starting election:",
        error);
717     setError("Failed to start election.");
718   }
719 };
720
721 const handleEndElection = async (electionId) => {
722   try {
723     const token = localStorage.getItem('token');
        // Retrieve the token from local storage
724     await axios.put(`http://localhost:8080/api/
        admin/elections/${electionId}/end`, {}, {

```

```

725         headers: {
726             'Authorization': `Bearer ${token}` //
              Pass JWT token
727         }
728     });
729     console.log("Election ended successfully");
730     fetchElections();
731 } catch (error) {
732     console.error("Error ending election:", error)
733     ;
734     setError("Failed to end election.");
735 }
736 };
737 const fetchElectionCandidates = async (electionId) =>
738 {
739     try {
740         const token = localStorage.getItem('token');
741         // Retrieve the token from local storage
742         const response = await axios.get(`http://
localhost:8080/api/admin/elections/${
electionId}/candidates`, {
743             headers: {
744                 'Authorization': `Bearer ${token}` //
              Pass JWT token
745             }
746         });
747         setCandidates(response.data);
748     } catch (error) {
749         console.error("Error fetching election
candidates:", error);
750     }
751 };
752 const getAvailableCandidates = () => {
753     return candidates.filter(candidate => !
candidate.elections || !
candidate.elections.includes(selectedElectionId
));
754 };
755 return (
756     <div>
757         <nav className="navbar navbar-expand-lg navbar
-light bg-light">
758             <div className="container-fluid">
759                 <a className="navbar-brand" href="#">
Manage Elections</a>
760                 <button className="navbar-toggler"

```

```

761         type="button" data-bs-toggle="
collapse" data-bs-target="#
navbarNav" aria-controls="navbarNav
" aria-expanded="false" aria-label=
"Toggle navigation">
762         <span className="navbar-toggler-
icon"></span>
763     </button>
764     <div className="collapse navbar-
collapse" id="navbarNav">
765         <ul className="navbar-nav ms-auto"
766             >
767             <li className="nav-item">
768                 <button className="btn btn
-link nav-link" onClick
={() => navigate('/
admin/dashboard')}>Back
to Dashboard</button>
769             </li>
770         </ul>
771     </div>
772 </nav>
773 <div className="container mt-5">
774     <h1>Manage Elections</h1>
775     <div className="mt-4">
776         <h2>Create New Election</h2>
777         {error && <p style={{ color: 'red'
}}>{error}</p>}
778         <form onSubmit={(e) => {
779             e.preventDefault();
780             handleCreateElection(); }}>
781             <div className="mb-3">
782                 <label className="form-label">
783                     Election Name</label>
784                 <input
785                     type="text"
786                     className="form-control"
787                     value={newElectionName}
788                     onChange={(e) =>
789                         setNewElectionName(
790                             e.target.value)}
791                 </div>
792                 <div className="mb-3">
793                     <label className="form-label">
794                         Start Date</label>
795                     <input
796                         type="date"

```

```

791         className="form-control"
792         value={
793             newElectionStartDate
794             onChange={(e) =>
795                 setNewElectionStartDate
796                 (e.target.value)}
797             />
798         </div>
799         <div className="mb-3">
800             <label className="form-label">
801                 End Date</label>
802             <input
803                 type="date"
804                 className="form-control"
805                 value={newElectionEndDate}
806                 onChange={(e) =>
807                     setNewElectionEndDate(
808                         e.target.value)}
809             />
810             </div>
811             <button type="submit" className="
812                 btn btn-primary">Create
813                 Election</button>
814         </form>
815     </div>
816     <div className="mt-5">
817         <h2>Add Candidate to Election</h2>
818         {error && <p style={{ color: 'red'
819             }}>{error}</p>}
820         <form onSubmit={(e) => {
821             e.preventDefault();
822             handleAddCandidateToElection(); }}>
823             <div className="mb-3">
824                 <label className="form-label">
825                     Select Election</label>
826                 <select
827                     className="form-control"
828                     value={selectedElectionId}
829                     onChange={(e) => {
830                         setSelectedElectionId(
831                             e.target.value);
832                         fetchElectionCandidates
833                         (e.target.value);
834                     }}
835                 >
836                     <option value="">Select an
837                         election</option>
838                     {Array.isArray(elections)
839                     && elections.map((

```

```

824         election) => (
            <option key={
                election.id} value
                ={{election.id}}>
825             {{election.name}} (
                from {
                    election.startDate
                } to {
                    election.endDate
                })
826         </option>
827     )}}
828 </select>
829 </div>
830 <div className="mb-3">
831     <label className="form-label">
        Select Candidate</label>
832     <select
833         className="form-control"
834         value={{selectedCandidateId
            }}
835         onChange={{(e) =>
            setSelectedCandidateId(
                e.target.value)}}
836     >
837         <option value="">Select a
            candidate</option>
838         {{Array.isArray(
            getAvailableCandidates
            ()) &&
            getAvailableCandidates
            ().map((candidate) => (
839             <option key={
                candidate.id} value
                ={{candidate.id}}>
840                 {{candidate.name}}
                ({{
                    candidate.party
                }})
            </option>
841             )}}
842         </select>
843 </div>
844 <button type="submit" className="
845     btn btn-primary">Add Candidate
    </button>
846 </form>
847 </div>
848 <div className="mt-5">

```

```

849      <h2>Manage Elections</h2>
850      <ul className="list-group">
851          {elections.map((election) => (
852              <li key={election.id}
                  className="list-group-item"
                  >
853                  <h3>{election.name} (from
                      {election.startDate} to
                      {election.endDate})</h3>
854                  <button className="btn btn
                      -success me-2" onClick
                      =={() =>
                          handleStartElection(
                              election.id)}>Start
                          Election</button>
855                  <button className="btn btn
                      -danger me-2" onClick
                      =={() =>
                          handleEndElection(
                              election.id)}>End
                          Election</button>
856                  <h4>Candidates</h4>
857                  <ul className="list-group"
                      >
858                      {
                        election.candidates.map
                        ((candidate) => (
859                          <li key={
                              candidate.id}
                              className="list
                              -group-item d-
                              flex justify-
                              content-between
                              align-items-
                              center">
860                              <span>{
                                  candidate.name
                              } ({
                                  candidate.party
                              })</span>
861                              <button
                                  className="
                                  btn btn-
                                  danger btn-
                                  sm" onClick
                                  =={() =>
                                      handleRemoveCandidateFromEle
                                      (

```



```

candidate.id
    >>Remove</
button>
862                                     </li>
863                                 )}}
864                             </ul>
865                         </li>
866                     )}}
867                 </ul>
868             </div>
869         </div>
870     </div>
871 );
872 }
873
874 export default ManageElections;
875
876 import React, { useState, useEffect } from 'react';
877 import axios from 'axios';
878 import { useNavigate } from 'react-router-dom';
879
880 function ManageUsers() {
881     const [users, setUsers] = useState([]);
882     const [editUserId, setEditUserId] = useState(null);
883     const [editUserFullName, setEditUserFullName] =
884         useState('');
885     const [editUserEmail, setEditUserEmail] = useState('')
886         ;
887     const [editUserMobile, setEditUserMobile] = useState('')
888         ;
889     const [editUserRole, setEditUserRole] = useState('
890         VOTER');
891     const [error, setError] = useState('');
892     const navigate = useNavigate();
893
894     useEffect(() => {
895         fetchUsers();
896     }, []);
897
898     const fetchUsers = async () => {
899         try {
900             const token = localStorage.getItem('token');
901             // Retrieve the token from local storage
902             const response = await axios.get("http://
903                 localhost:8080/api/admin/users", {
904                 headers: {
905                     'Authorization': `Bearer ${token}` //
906                     Pass JWT token
907                 }
908             }

```

```

901         });
902         console.log("Users fetched:", response.data);
903         setUsers(response.data);
904     } catch (error) {
905         console.error("Error fetching users:", error);
906     }
907 };
908
909 const handleDeleteUser = async (userId) => {
910     try {
911         const token = localStorage.getItem('token');
912         // Retrieve the token from local storage
913         await axios.delete(`http://localhost:8080/api/
914             admin/users/${userId}`, {
915             headers: {
916                 'Authorization': `Bearer ${token}` //
917                 Pass JWT token
918             }
919         });
920         setUsers(users.filter(user => user.id !==
921             userId));
922     } catch (error) {
923         console.error("Error deleting user:", error);
924         setError("Failed to delete user.");
925     }
926 };
927
928 const handleEditUser = async () => {
929     if (!editUserFullName || !editUserEmail || !
930         editUserRole) {
931         setError("All fields are required.");
932         return;
933     }
934     try {
935         const token = localStorage.getItem('token');
936         // Retrieve the token from local storage
937         const userData = {
938             fullName: editUserFullName,
939             email: editUserEmail,
940             mobile: editUserMobile,
941             role: editUserRole
942         };
943         console.log("Editing user:", userData);
944         const response = await axios.put(`http://
945             localhost:8080/api/admin/users/${editUserId
946             }`, userData, {
947             headers: {
948                 'Authorization': `Bearer ${token}` //
949                 Pass JWT token

```

```

941         }
942     });
943     console.log("User edited successfully:",
944         response.data);
945     setUsers(users.map(user => user.id ===
946         editUserId ? response.data : user));
947     setEditUserId(null);
948     setEditUserFullName("");
949     setEditUserEmail("");
950     setEditUserRole("VOTER");
951     setError("");
952     } catch (error) {
953         console.error("Error editing user:", error);
954         setError("Failed to edit user.");
955     }
956 };
957
958 const startEditUser = (user) => {
959     setEditUserId(user.id);
960     setEditUserFullName(user.fullName);
961     setEditUserEmail(user.email);
962     setEditUserMobile(user.mobile);
963     setEditUserRole(user.role);
964 };
965
966 return (
967     <div>
968         <nav className="navbar navbar-expand-lg navbar
969             -light bg-light">
970             <div className="container-fluid">
971                 <a className="navbar-brand" href="#">
972                     Manage Users</a>
973                 <button className="navbar-toggler"
974                     type="button" data-bs-toggle="
975                     collapse" data-bs-target="#
976                     navbarNav" aria-controls="navbarNav
977                     " aria-expanded="false" aria-label=
978                     "Toggle navigation">
979                     <span className="navbar-toggler-
980                         icon"></span>
981                 </button>
982                 <div className="collapse navbar-
983                     collapse" id="navbarNav">
984                     <ul className="navbar-nav ms-auto"
985                         >
986                         <li className="nav-item">
987                             <button className="btn btn
988                                 -link nav-link" onClick
989                                 =`() => navigate('/

```

```

                                admin/dashboard'))}>Back
                                to Dashboard</button>
976                                </li>
977                                </ul>
978                                </div>
979                                </div>
980                            </nav>
981                            <div className="container mt-5">
982                                <h1>Manage Users</h1>
983                                <div className="mt-5">
984                                    <h2>All Users</h2>
985                                    {error && <p style={{ color: 'red'
986                                        }}>{error}</p>}
987                                    <ul className="list-group">
988                                        {users.map((user) => (
989                                            <li key={user.id} className="
990                                                list-group-item d-flex
991                                                justify-content-between
992                                                align-items-center">
993                                                <span>{user.fullName} ({
994                                                    user.email}) - {
995                                                    user.role}</span>
996                                                <div>
997                                                    <button className="btn
998                                                        btn-secondary btn-
999                                                        sm me-2" onClick
1000                                                        ={{() =>
1001                                                            startEditUser(user)
1002                                                        }}>Edit</button>
1003                                                    <button className="btn
1004                                                        btn-danger btn-sm"
1005                                                        onClick={{() =>
1006                                                            handleDeleteUser(
1007                                                                user.id)}}>Delete</
1008                                                        button>
1009                                                </div>
1010                                            </li>
1011                                        )}}
1012                                    </ul>
1013                                </div>
1014                                {editUserId && (
1015                                    <div className="mt-5">
1016                                        <h2>Edit User</h2>
1017                                        <form onSubmit={{(e) => {
1018                                            e.preventDefault();
1019                                            handleEditUser(); }}>
1020                                            <div className="mb-3">
1021                                                <label className="form-
1022                                                    label">Full Name</label>

```

```

1004         >
1005         <input
1006             type="text"
1007             className="form-
1008                 control"
1009             value={
1010                 editUserFullName}
1011             onChange={(e) =>
1012                 setEditUserFullName
1013                 (e.target.value)}
1014         />
1015     </div>
1016     <div className="mb-3">
1017         <label className="form-
1018             label">Email</label>
1019         <input
1020             type="email"
1021             className="form-
1022                 control"
1023             value={editUserEmail}
1024             onChange={(e) =>
1025                 setEditUserEmail(
1026                     e.target.value)}
1027         />
1028     </div>
1029     <div className="mb-3">
1030         <label className="form-
1031             label">Mobile</label>
1032         <input
1033             type="text"
1034             className="form-
1035                 control"
1036             value={editUserMobile}
1037             onChange={(e) =>
1038                 setEditUserMobile(
1039                     e.target.value)}
1040         />
1041     </div>
1042     <div className="mb-3">
1043         <label className="form-
1044             label">Role</label>
1045         <select
1046             className="form-
1047                 control"
1048             value={editUserRole}
1049             onChange={(e) =>
1050                 setEditUserRole(
1051                     e.target.value)}
1052         >

```

```

1036             <option value="VOTER">
1037                 VOTER</option>
1038             <option value="ADMIN">
1039                 ADMIN</option>
1040             </select>
1041         </div>
1042         <button type="submit"
1043             className="btn btn-primary"
1044             >Save</button>
1045         <button type="button"
1046             className="btn btn-
1047                 secondary ms-2" onClick={()
1048                 => setEditUserId(null)}>
1049             Cancel</button>
1050     </form>
1051 </div>
1052     })
1053 </div>
1054 </div>
1055 );
1056 }
1057 export default ManageUsers;
1058
1059 import React, { useState, useEffect } from 'react';
1060 import { getCurrentUser } from './UserService';
1061 import axios from 'axios';
1062 import { useNavigate } from 'react-router-dom';
1063
1064 function ProfilePage() {
1065     const navigate = useNavigate();
1066     const [userData, setUserData] = useState({
1067         id: '',
1068         fullName: '',
1069         email: '',
1070         mobile: '',
1071         voterIdCode: '',
1072     });
1073     const [error, setError] = useState('');
1074
1075     useEffect(() => {
1076         fetchUserProfile();
1077     }, []);
1078
1079     const fetchUserProfile = async () => {
1080         try {
1081             const data = await getCurrentUser();
1082             console.log('User data:', data);
1083             const token = localStorage.getItem('token');

```

```

1077         // Retrieve the token from local storage
const response = await axios.get(`http://
localhost:8080/api/voter/voterIdCode/${
data.id}` , {
1078
1079         headers: {
1080             'Authorization': `Bearer ${token}` //
                Pass JWT token
1081         });
1082
1083         console.log('Voter ID Code:', response.data);
1084         data.voterIdCode = response.data;
1085         setUserData(data);
1086
1087     } catch (err) {
1088         console.error(err);
1089         setError('Failed to fetch profile');
1090     }
1091 };
1092
1093 const handleBackToDashboard = () => {
1094     navigate('/dashboard');
1095 };
1096
1097 return (
1098     <div>
1099         <h1>Profile Page</h1>
1100         {error && <p style={{ color: 'red' }}>{error
            }</p>}
1101         <div>
1102             <p><strong>Full Name:</strong> {
                userData.fullName}</p>
1103             <p><strong>Email:</strong> {userData.email
                }</p>
1104             <p><strong>Mobile:</strong> {
                userData.mobile}</p>
1105             <p><strong>Voter ID Code:</strong> {
                userData.voterIdCode}</p>
1106             <button onClick={handleBackToDashboard}>
                Back to Dashboard</button>
1107         </div>
1108     </div>
1109 );
1110 }
1111
1112 export default ProfilePage;
1113
1114 import React, { useState } from 'react';
1115 import axios from 'axios';

```

```

1116 import { useNavigate } from 'react-router-dom';
1117 import { MDBContainer, MDBInput } from 'mdb-react-ui-kit';
1118
1119 function SignupPage() {
1120     const [fullName, setFullName] = useState('');
1121     const [email, setEmail] = useState('');
1122     const [password, setPassword] = useState('');
1123     const [confirmPassword, setConfirmPassword] = useState
        ('');
1124     const [mobile, setMobile] = useState(''); // renamed
        state
1125     const [role] = useState('VOTER'); // role remains
        fixed as 'VOTER'
1126     const [voterIdCode, setVoterIdCode] = useState('');
1127     const [error, setError] = useState(''); // State to
        manage error messages
1128     const history = useNavigate(); // Navigation hook
1129
1130     const handleSignup = async () => {
1131         try {
1132             // Check for empty fields
1133             if (!fullName || !email || !password || !
                confirmPassword || !mobile || !voterIdCode)
                {
1134                 setError('Please fill in all fields.');
```



```

1156         response.data.jwt);
1157         //Show success message
1158         alert('Signup successful!');
1159
1160         // Redirect to login page after successful
1161         signup
1162         history('/'); // Redirect after successful
1163         signup
1164     } catch (error) {
1165         // Handle signup error
1166         console.error('Signup failed:', error.response
1167             ? error.response.data : error.message);
1168         setError(error.response ? error.response.data
1169             : error.message);
1170     }
1171 };
1172
1173 return (
1174     <div className="d-flex justify-content-center
1175         align-items-center vh-100">
1176         <div className="border rounded-lg p-4" style
1177             ={{ width: '600px', height: 'auto' }}>
1178             <MDBContainer className="p-3">
1179                 <h2 className="mb-4 text-center">Sign
1180                 Up as a Voter</h2>
1181                 {error && <p className="text-danger">{
1182                 error}</p>}
1183                 <MDBInput
1184                     wrapperClass="mb-3"
1185                     id="fullName"
1186                     placeholder="Full Name"
1187                     value={fullName}
1188                     type="text"
1189                     onChange={(e) => setFullName(
1190                         e.target.value)}
1191                 />
1192                 <MDBInput
1193                     wrapperClass="mb-3"

```

```

1194         placeholder="Password"
1195         id="password"
1196         type="password"
1197         value={password}
1198         onChange={(e) => setPassword(
1199             e.target.value)}
1200     />
1201     <MDBInput
1202         wrapperClass="mb-3"
1203         placeholder="Confirm Password"
1204         id="confirmPassword"
1205         type="password"
1206         value={confirmPassword}
1207         onChange={(e) =>
1208             setConfirmPassword(
1209                 e.target.value)}
1210     />
1211     <MDBInput
1212         wrapperClass="mb-2"
1213         placeholder="Mobile Number"
1214         id="mobile"
1215         value={mobile}
1216         type="text" // Changed to 'tel'
1217                     type for better validation
1218         onChange={(e) => setMobile(
1219             e.target.value)}
1220     />
1221     <MDBInput
1222         wrapperClass="mb-2"
1223         placeholder="Voter ID Code"
1224         id="voterIdCode"
1225         value={voterIdCode}
1226         type="text"
1227         onChange={(e) => setVoterIdCode(
1228             e.target.value)}
1229     />
1230     <button
1231         className="mb-4 d-block mx-auto
1232             fixed-action-btn btn-primary"
1233         style={{ height: '40px', width: '
1234             100%' }}
1235         onClick={handleSignup}
1236     >
1237         Sign Up
1238     </button>
1239     <div className="text-center">
1240         <p>Already registered? <a href="/"
1241             >Login</a></p>
1242     </div>

```

```

1234             </MDBContainer>
1235         </div>
1236     </div>
1237     );
1238 }
1239
1240 export default SignupPage;
1241
1242 import axios from 'axios';
1243
1244 export const getCurrentUser = async () => {
1245     const token = localStorage.getItem('token');
1246     try {
1247         const response = await axios.get('http://
1248             localhost:8080/api/auth/me', {
1249             headers: {
1250                 'Authorization': `Bearer ${token}`,
1251             },
1252         });
1253         return response.data;
1254     } catch (error) {
1255         console.error('Error fetching current user:',
1256             error);
1257         throw error;
1258     }
1259 };

```

Chapter 3

Appendix: Mini Project

```
1 package com.example.votingSystem.model;
2
3 import jakarta.persistence.*;
4
5 import java.io.Serializable;
6
7 @Entity
8 public class Demo implements Serializable {
9     @Id
10    @GeneratedValue(strategy = GenerationType.AUTO)
11    private Long id;
12    private String demoString;
13
14    public Demo() {
15
16    }
17
18    public Demo(String demoString) {
19        this.demoString = demoString;
20    }
21
22    public String getDemoString() {
23        return demoString;
24    }
25
26    public void setDemoString(String demoString) {
27        this.demoString = demoString;
28    }
29 }
```

```

30
31 package com.example.votingSystem.repo;
32
33 import com.example.votingSystem.model.Demo;
34 import org.springframework.data.jpa.repository.
    JpaRepository;
35
36 public interface DemoRepo extends JpaRepository<Demo,
    Long> {
37 }
38 package com.example.votingSystem.service;
39
40 import com.example.votingSystem.model.Demo;
41 import com.example.votingSystem.repo.DemoRepo;
42 import org.springframework.beans.factory.annotation.
   Autowired;
43 import org.springframework.stereotype.Service;
44 import org.springframework.transaction.annotation.
   Transactional;
45
46
47 @Service
48 public class DemoService {
49     private final DemoRepo demoRepo;
50
51     @Autowired
52     public DemoService(DemoRepo demoRepo) {
53         this.demoRepo = demoRepo;
54     }
55
56     @Transactional(readOnly = true)
57     public Demo findDemoById(Long demoId) {
58         return demoRepo.findById(demoId).orElse(null);
59     }
60 }
61
62 package com.example.votingSystem.controller;
63
64 import com.example.votingSystem.model.Demo;
65 import com.example.votingSystem.service.DemoService;
66 import org.springframework.http.HttpStatus;
67 import org.springframework.http.ResponseEntity;

```

```

68 import org.springframework.web.bind.annotation.
    CrossOrigin;
69 import org.springframework.web.bind.annotation.
    GetMapping;
70 import org.springframework.web.bind.annotation.
    RestController;
71
72
73 @RestController
74 public class DemoController {
75     private final DemoService demoService;
76
77     public DemoController(DemoService demoService) {
78         this.demoService = demoService;
79     }
80
81     @GetMapping("/demo")
82     @CrossOrigin(origins = "http://localhost:3000")
83     public ResponseEntity<Demo> getDemoById() {
84         Demo demo = demoService.findDemoById(1L);
85         return new ResponseEntity<>(demo, HttpStatus.OK)
            ;
86     }
87
88 }
89
90 import React, { useEffect, useState } from 'react';
91
92 function App() {
93     const [message, setMessage] = useState("");
94
95     useEffect(() => {
96         fetch("http://localhost:8080/demo")
97             .then((response) => response.json())
98             .then((data) => setMessage(data.demoString))
99             .catch((error) => console.error("Error␣
            fetching␣message:", error));
100     }, []);
101
102     return (
103         <div>
104             <h1>Backend Message:</h1>
105             <p>{message}</p>

```

```
106         </div>
107     );
108 }
109
110 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/>.