UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

"Election Database Management System"

Submitted by:

SUHAS.M	**PES1UG21CS310**
KRISHNA MODALA	**PES1UG21CS288**

6th Semester E Section

Prof. Course BHARGAVI MOKASHI
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

TABLE OF CONTENTS

1.SYNOPSIS:

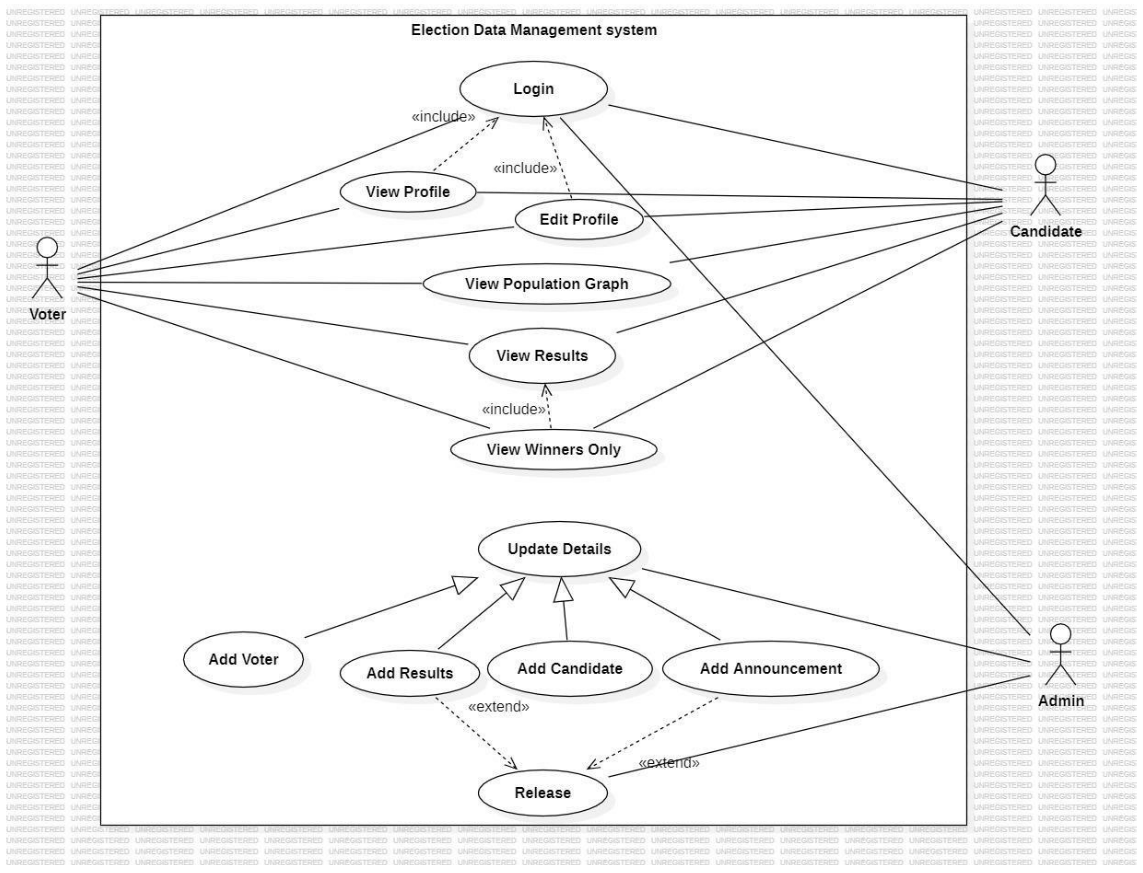
The Election Management System (EMS) is a web-based application designed to streamline electoral processes efficiently and transparently. Leveraging Java and the Spring Framework, the system offers a user-friendly interface for registration, voting, party and candidate exploration, and result tracking.

During registration, users provide their full name, username, and password to create accounts securely. Once registered, users can cast their votes, explore participating parties and candidates, and monitor election results.

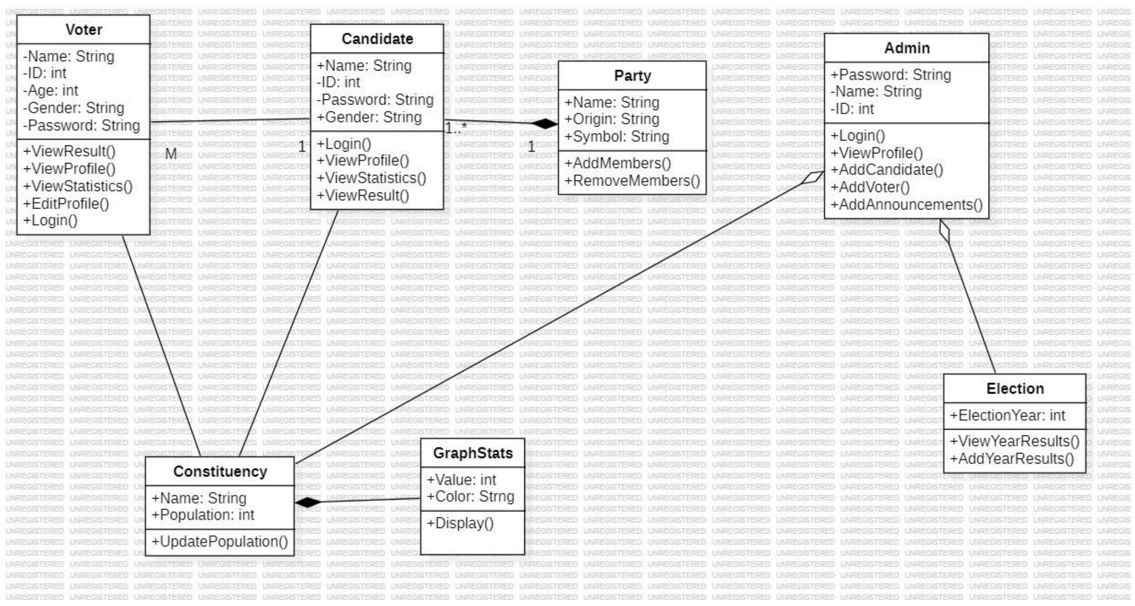
Administrators oversee the EMS, with the ability to appoint other administrators, initiate and conclude elections, and manage candidates and parties. The system prioritizes security, implementing authentication mechanisms for secure interactions.

Overall, the EMS is a concise, secure, and accessible platform for facilitating democratic processes using Java and the Spring Framework.

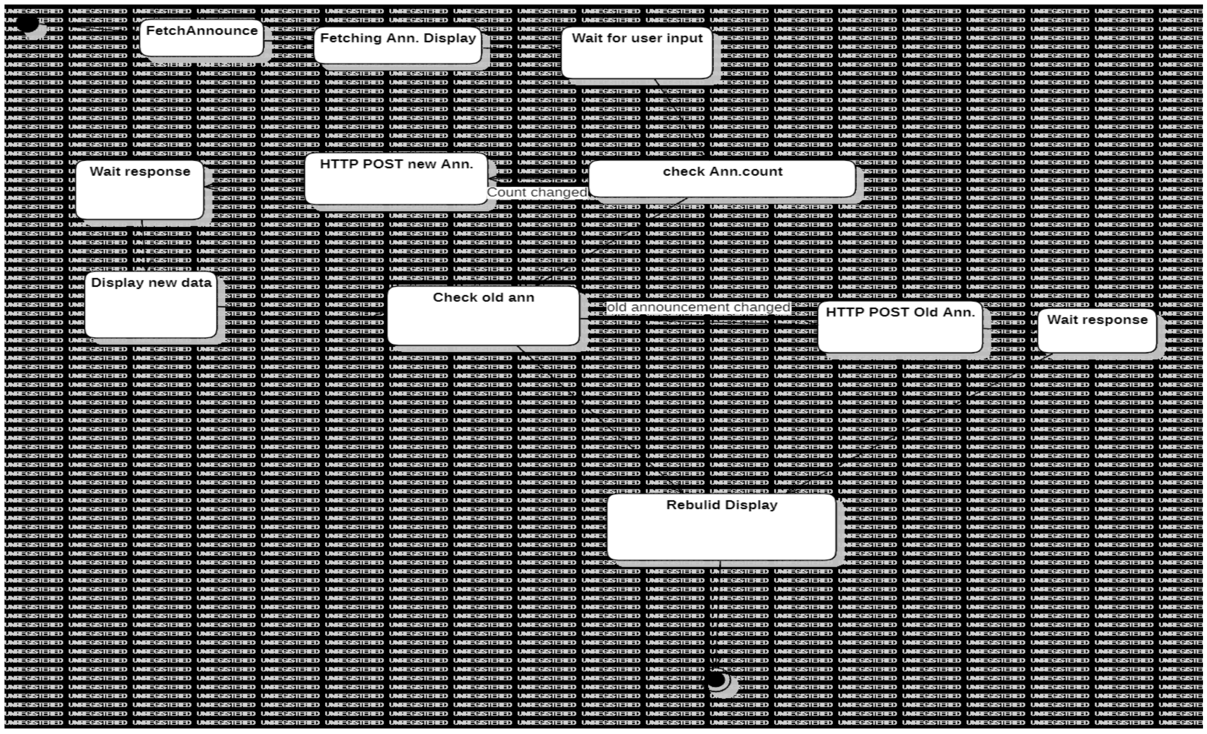
2. USE CASE DIAGRAM:

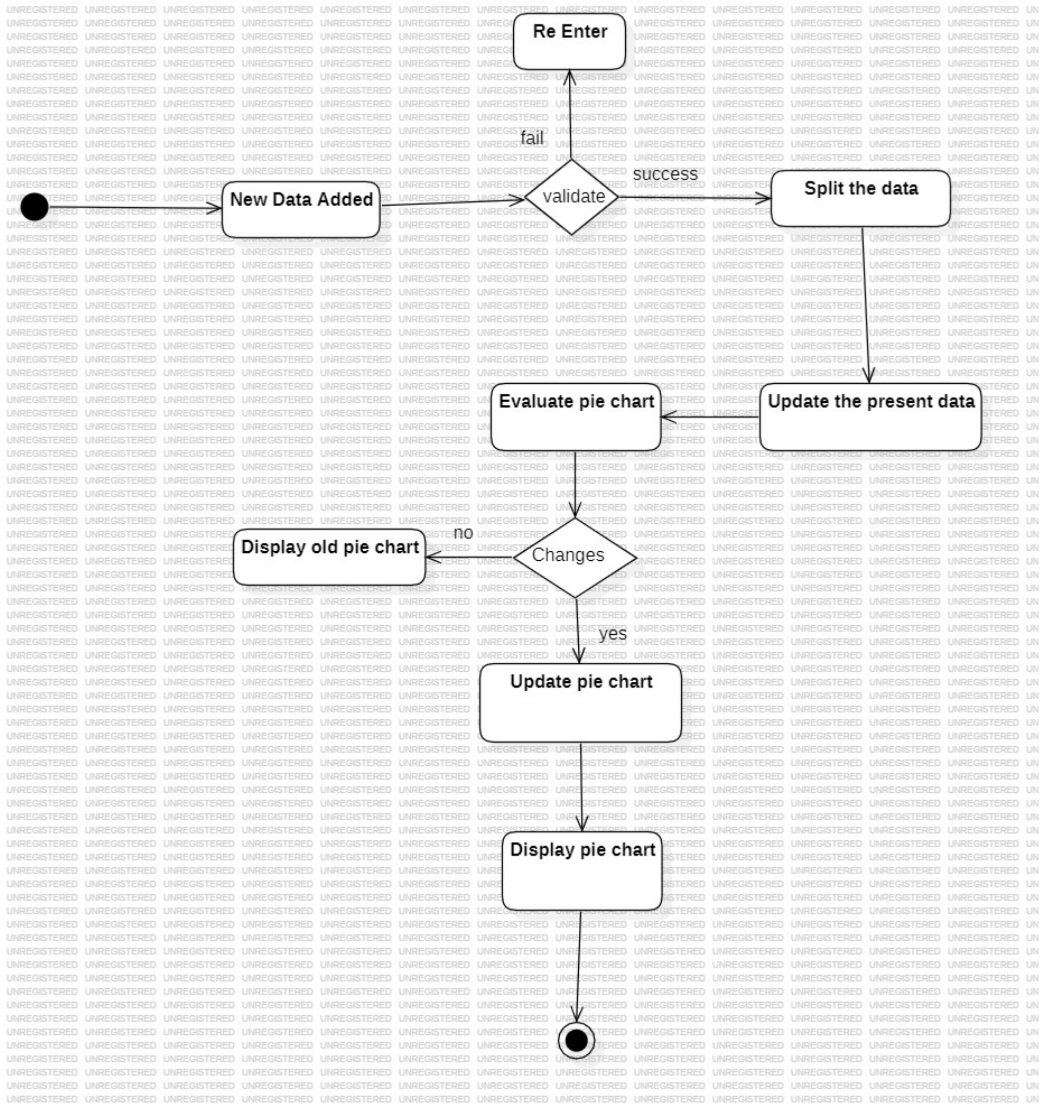


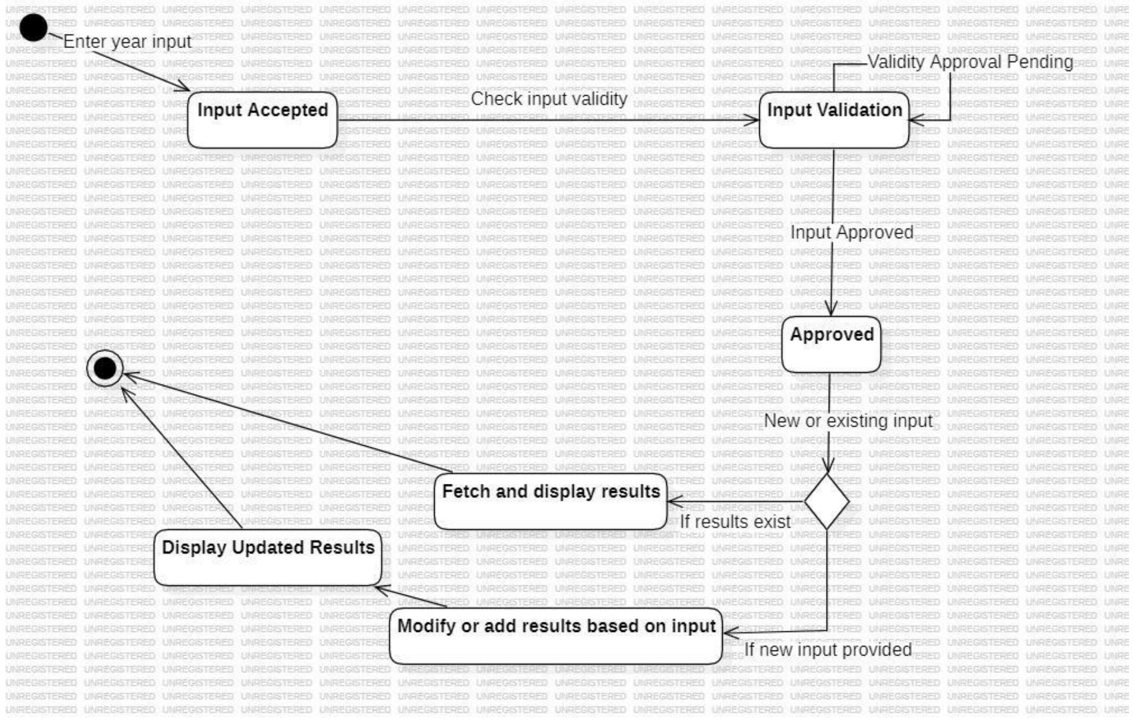
3. CLASS DIAGRAM:



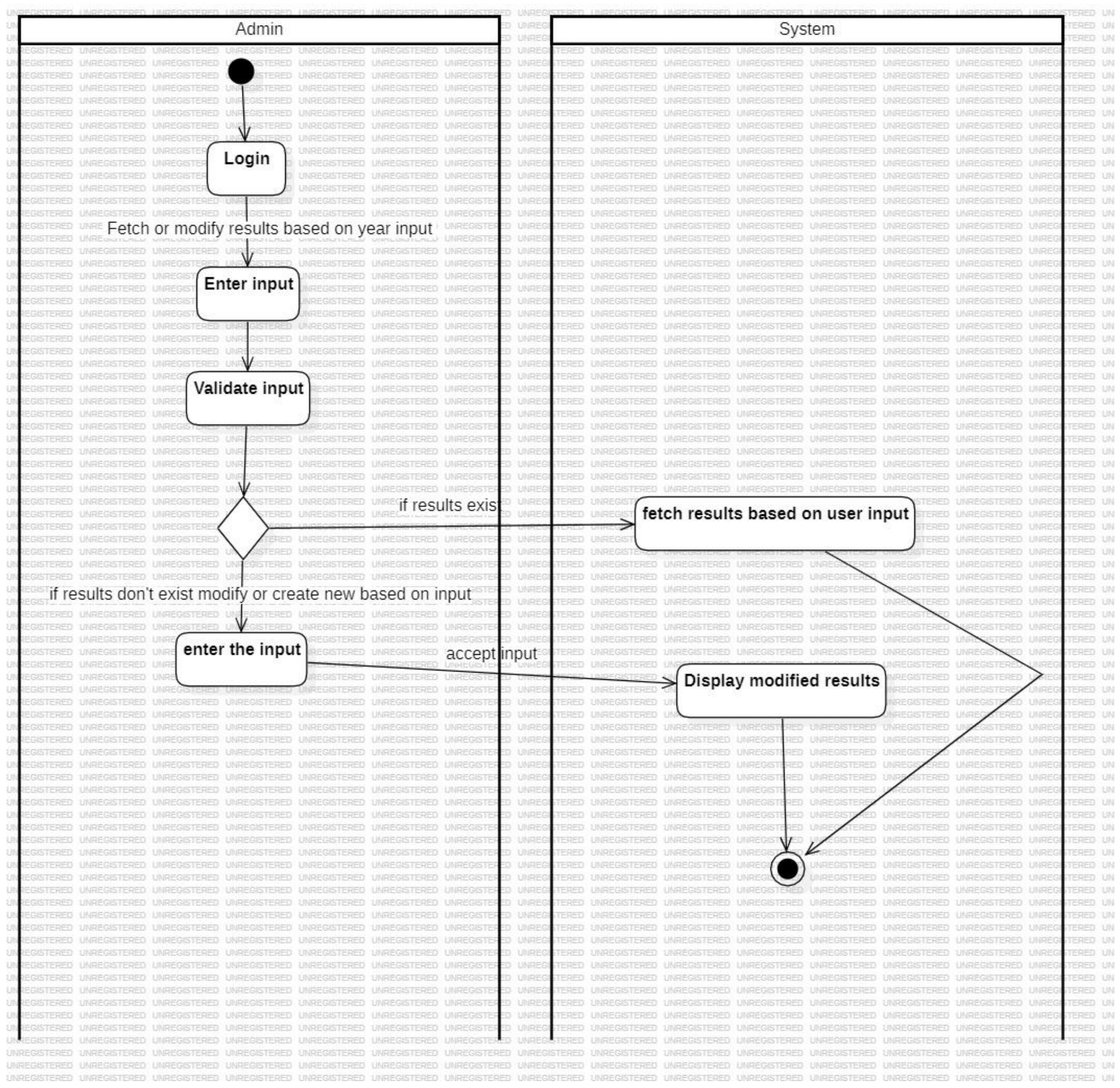
4. STATE DIAGRAM:

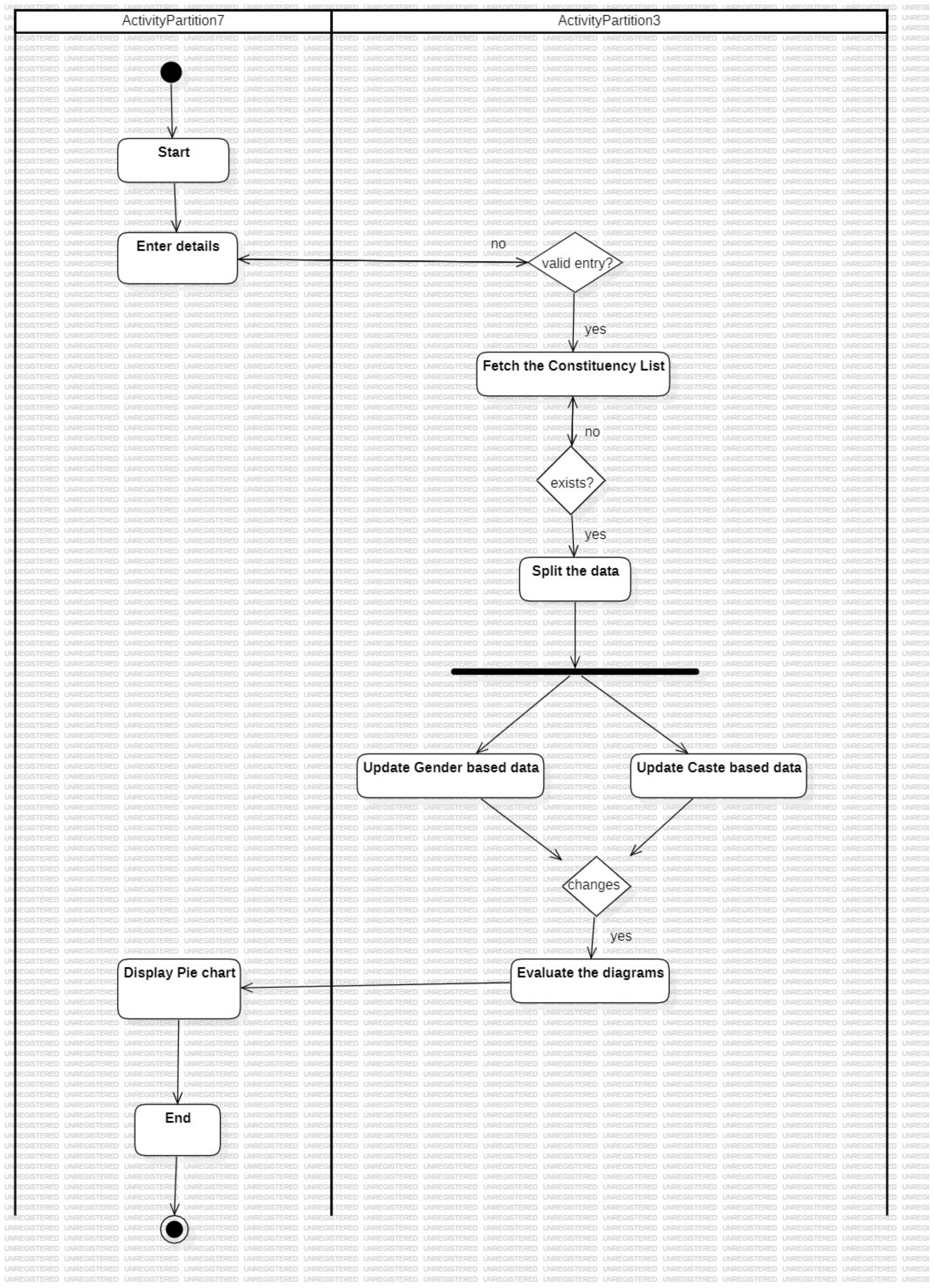



Re Enter
fail
success
New Data Added
validate
Split the data
Evaluate pie chart
Update the present data
no
Display old pie chart
Changes
yes
Update pie chart
Display pie chart

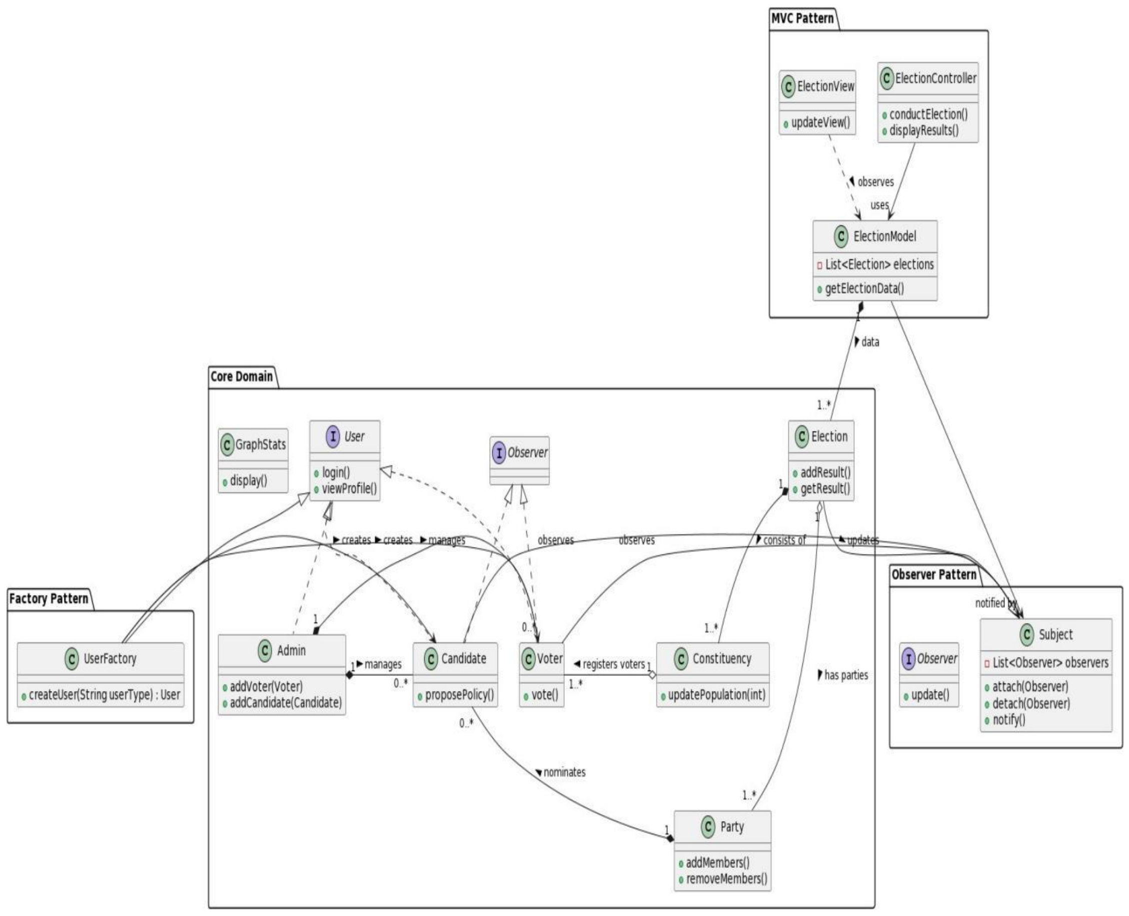

Enter year input
Input Accepted
Check input validity
Input Validation
Validity Approval Pending
Input Approved
Approved
New or existing input
Fetch and display results
If results exist
Display Updated Results
Modify or add results based on input
If new input provided

5. ACTIVITY DIAGRAM:


Admin
System
Login
Fetch or modify results based on year input
Enter input
Validate input
if results exist
fetch results based on user input
if results don't exist modify or create new based on input
enter the input
accept input
Display modified results


ActivityPartition7
ActivityPartition3
Start
Enter details
no
valid entry?
yes
Fetch the Constituency List
no
exists?
yes
Split the data
Update Gender based data
Update Caste based data
changes
yes
Evaluate the diagrams
Display Pie chart
End

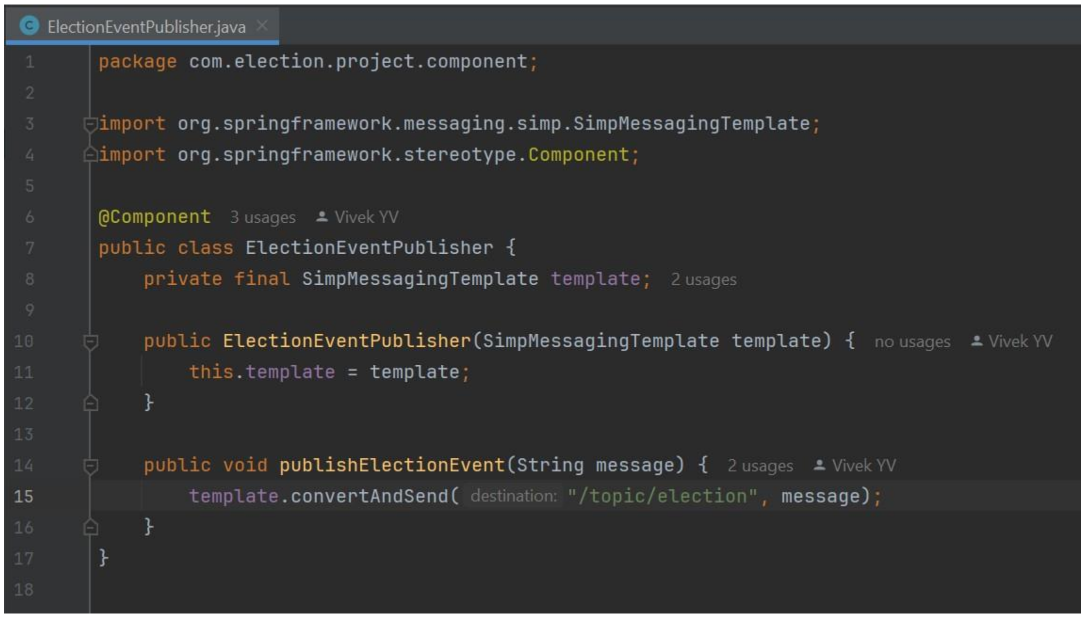
6. CLASS DIAGRAM WITH MVC ARCHITECTURE:



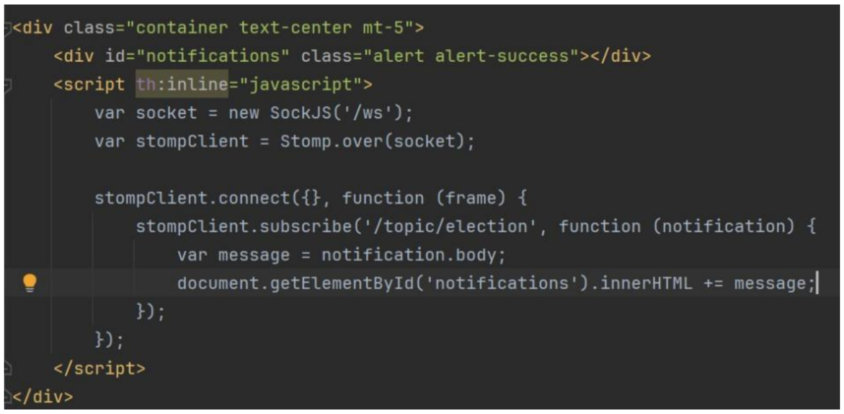
7. DESIGN PATTERNS

1) Behavioral Pattern: Observer

The code implements the observer pattern by using ElectionEventPublisher to send messages through SimpMessagingTemplate. When publishElectionEvent is called, it notifies all subscribers to the "/topic/election" channel with the provided message, similar to observers reacting to a subject's state change.

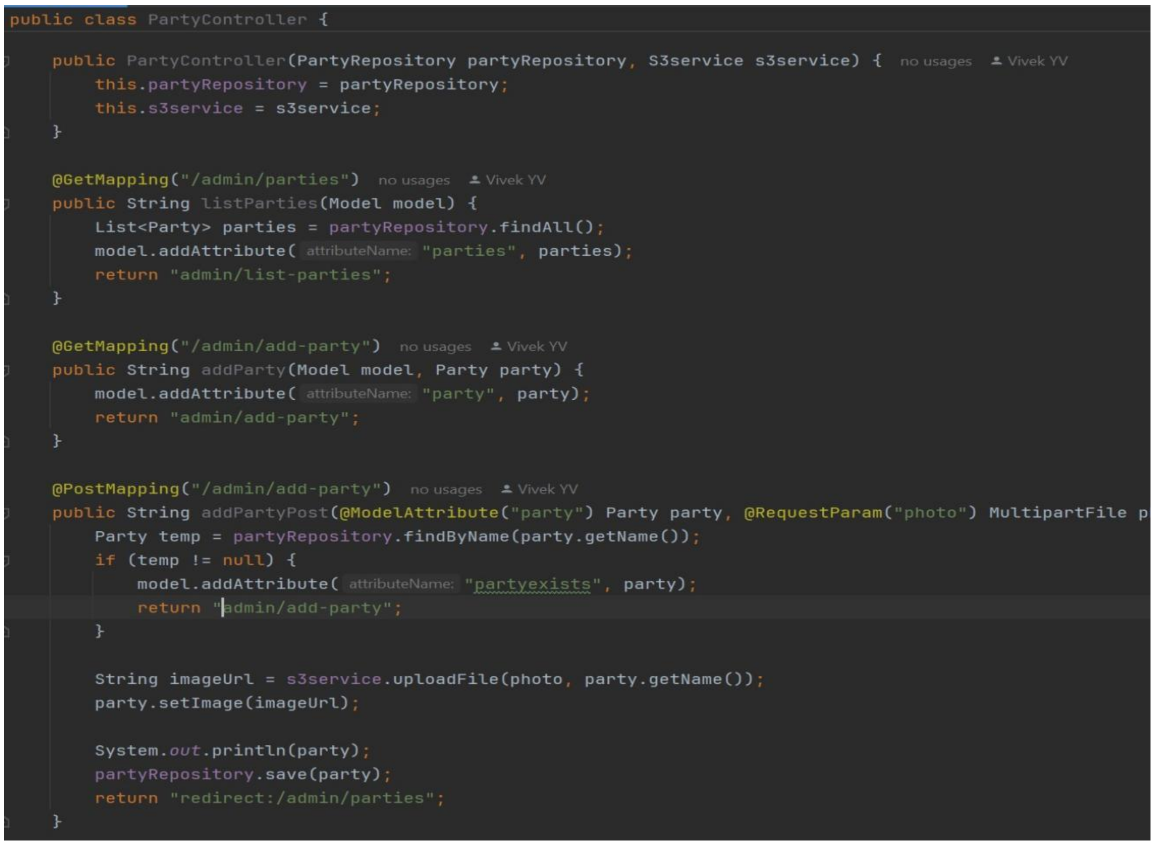


```java
package com.election.project.component;

import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Component;

@Component  3 usages  Vivek YV
public class ElectionEventPublisher {
    private final SimpMessagingTemplate template;  2 usages

    public ElectionEventPublisher(SimpMessagingTemplate template) {  no usages  Vivek YV
        this.template = template;
    }

    public void publishElectionEvent(String message) {  2 usages  Vivek YV
        template.convertAndSend( destination: "/topic/election", message);
    }
}
```


```
<div class="container text-center mt-5">
    <div id="notifications" class="alert alert-success"></div>
    <script th:inline="javascript">
        var socket = new SockJS('/ws');
        var stompClient = Stomp.over(socket);

        stompClient.connect({}, function (frame) {
            stompClient.subscribe('/topic/election', function (notification) {
                var message = notification.body;
                document.getElementById('notifications').innerHTML += message;
            });
        });
    </script>
</div>
```

2) Behavioral Pattern - Mediator

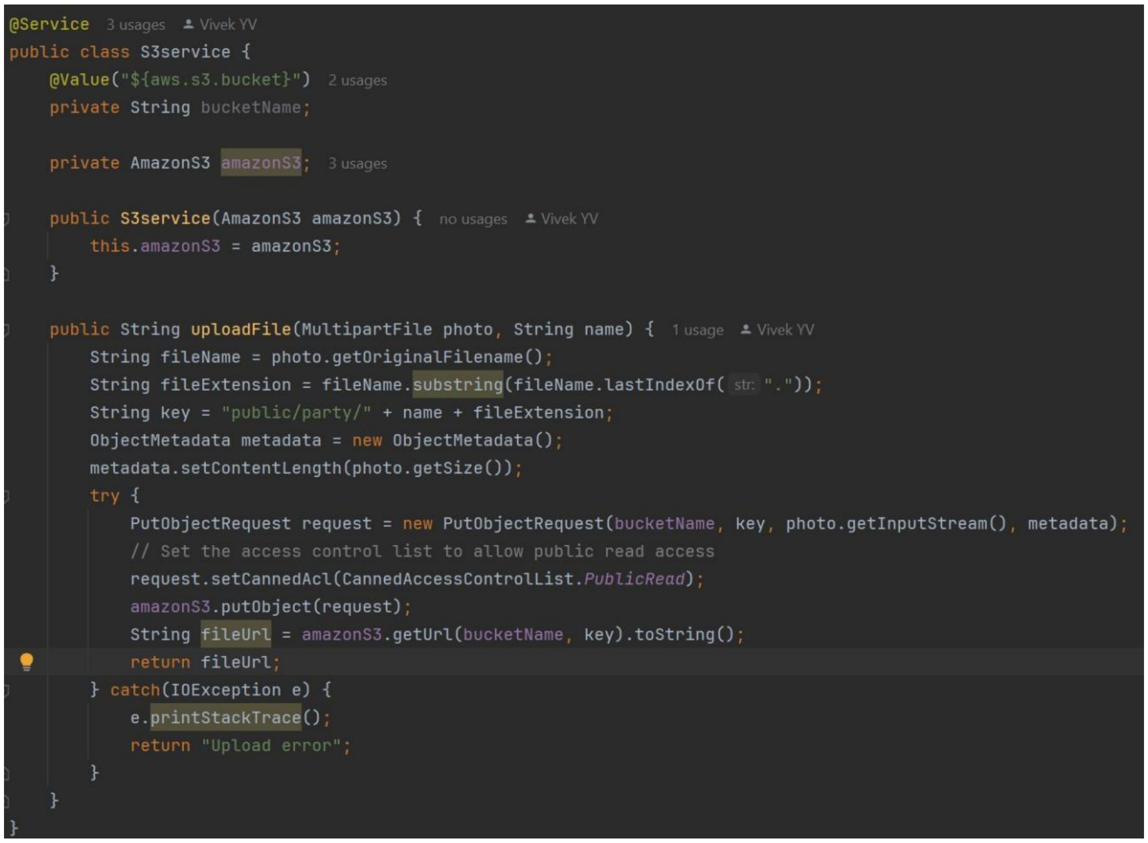
The `PartyController` acts as a mediator betweenthe UI layer and the service/data layers. It doesn't perform any operations on data by itself but delegatesto `PartyRepository` for data access and `S3Service` for file uploads. The controller mediates the input from the UI, processes it with the help of services, and then returns the appropriate view template or redirect, thereby orchestrating the flow ofdata and interactions across the system without directdependencies between the UI and data handling services.



```
public class PartyController {

    public PartyController(PartyRepository partyRepository, S3service s3service) {  no usages  Vivek YV
        this.partyRepository = partyRepository;
        this.s3service = s3service;
    }

    @GetMapping("/admin/parties")  no usages  Vivek YV
    public String listParties(Model model) {
        List<Party> parties = partyRepository.findAll();
        model.addAttribute( attributeName: "parties", parties);
        return "admin/list-parties";
    }

    @GetMapping("/admin/add-party")  no usages  Vivek YV
    public String addParty(Model model, Party party) {
        model.addAttribute( attributeName: "party", party);
        return "admin/add-party";
    }

    @PostMapping("/admin/add-party")  no usages  Vivek YV
    public String addPartyPost(@ModelAttribute("party") Party party, @RequestParam("photo") MultipartFile p
        Party temp = partyRepository.findByName(party.getName());
        if (temp != null) {
            model.addAttribute( attributeName: "partyexists", party);
            return "admin/add-party";
        }

        String imageUrl = s3service.uploadFile(photo, party.getName());
        party.setImage(imageUrl);

        System.out.println(party);
        partyRepository.save(party);
        return "redirect:/admin/parties";
    }
```

3) Behavioral Pattern: Chain of Responsibility

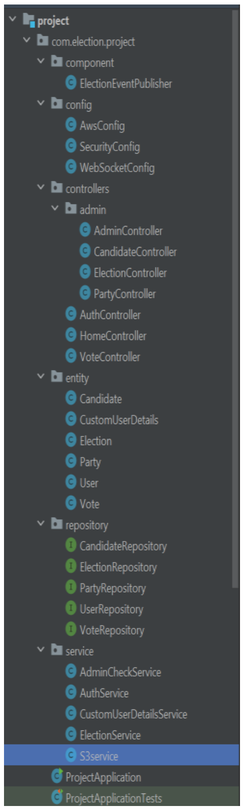
In our project, the Chain of Responsibility pattern is implemented by a series of components that pass requests along a chain until processed. Each component handles a specific task, and if unable to process the request, forwards it to the next component, streamlining request handling.


```java
@Service  3 usages  Vivek YV
public class S3service {
    @Value("${aws.s3.bucket}")  2 usages
    private String bucketName;

    private AmazonS3 amazonS3;  3 usages

    public S3service(AmazonS3 amazonS3) {  no usages  Vivek YV
        this.amazonS3 = amazonS3;
    }

    public String uploadFile(MultipartFile photo, String name) {  1 usage  Vivek YV
        String fileName = photo.getOriginalFilename();
        String fileExtension = fileName.substring(fileName.lastIndexOf( str: "."));
        String key = "public/party/" + name + fileExtension;
        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setContentLength(photo.getSize());
        try {
            PutObjectRequest request = new PutObjectRequest(bucketName, key, photo.getInputStream(), metadata);
            // Set the access control list to allow public read access
            request.setCannedAcl(CannedAccessControlList.PublicRead);
            amazonS3.putObject(request);
            String fileUrl = amazonS3.getUrl(bucketName, key).toString();
            return fileUrl;
        } catch(IOException e) {
            e.printStackTrace();
            return "Upload error";
        }
    }
}
```

8. DESIGN PRINCIPLES

1) Single Responsibility Principle

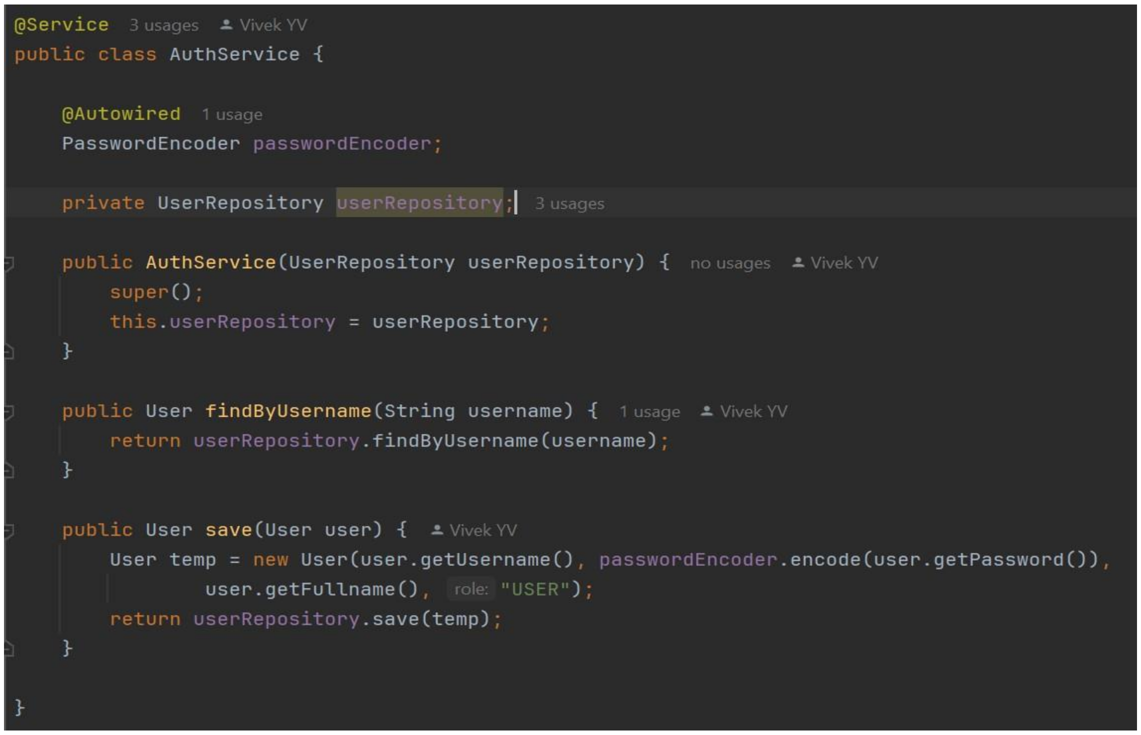
In our project, the Single Responsibility Principle is adhered to by designing each class to handle a single part of the functionality. For instance, `S3Service` exclusively manages file uploads to AWS S3, while other classes focus on database interactions or user input handling. This ensures that each class has one reason to change, simplifying maintenance and scalability.



```
project
  com.election.project
    component
      ElectionEventPublisher
    config
      AwsConfig
      SecurityConfig
      WebSocketConfig
    controllers
      admin
        AdminController
        CandidateController
        ElectionController
        PartyController
      AuthController
      HomeController
      VoteController
    entity
      Candidate
      CustomUserDetails
      Election
      Party
      User
      Vote
    repository
      CandidateRepository
      ElectionRepository
      PartyRepository
      UserRepository
      VoteRepository
    service
      AdminCheckService
      AuthService
      CustomUserDetailsService
      ElectionService
      S3service
    ProjectApplication
    ProjectApplicationTests
```

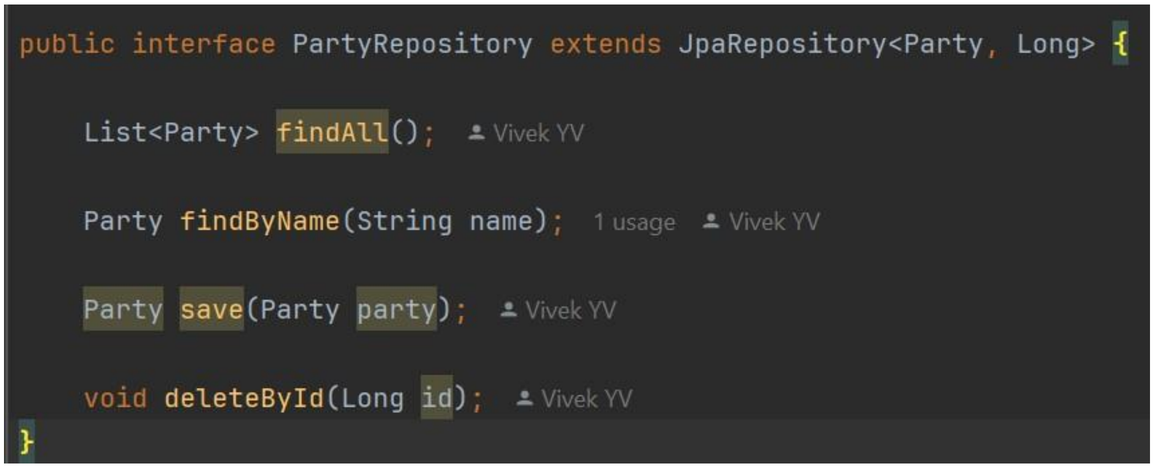
2) Open-Close Principle

We demonstrate the Open/Closed Principle by definingthe `AuthService` class, which is open for extension but closed for modification. The `findbyUsername` and `save` methods implement consistent behaviors foruser authentication. If new authentication methods are needed, instead of modifying these existing methods, we would extend the service with new methods or subclasses, ensuring the original `AuthService` remains unchanged. This approach adheres to the principle by allowing new functionality to be added withminimal impact on existing code.


```
@Service  3 usages  Vivek YV
public class AuthService {

    @Autowired  1 usage
    PasswordEncoder passwordEncoder;

    private UserRepository userRepository;  3 usages

    public AuthService(UserRepository userRepository) {  no usages  Vivek YV
        super();
        this.userRepository = userRepository;
    }

    public User findByUsername(String username) {  1 usage  Vivek YV
        return userRepository.findByUsername(username);
    }

    public User save(User user) {  Vivek YV
        User temp = new User(user.getUsername(), passwordEncoder.encode(user.getPassword()),
                user.getFullname(),  role: "USER");
        return userRepository.save(temp);
    }

}
```

3) Liskov Substitution Principle

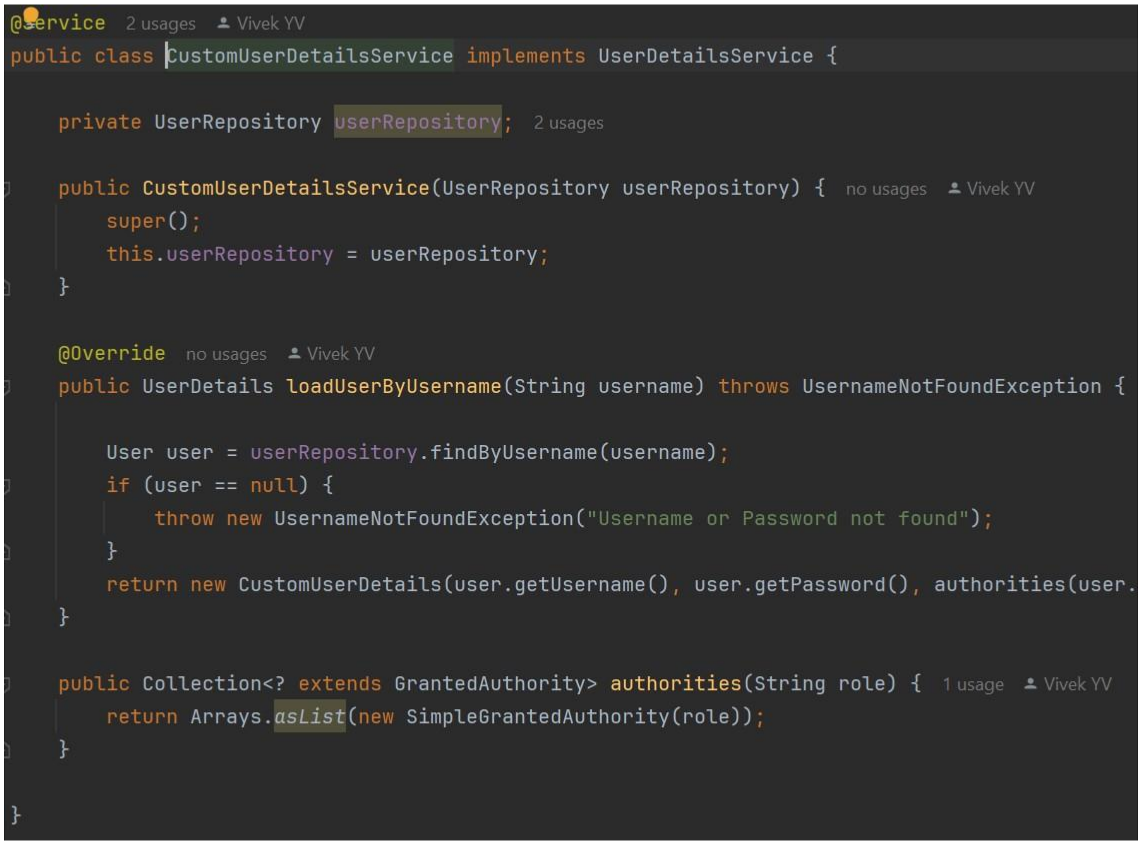
The `PartyRepository` interface in the provided code exemplifies the Liskov Substitution Principle (LSP) by extending `JpaRepository`. Any class that implements `PartyRepository` can be substituted for `JpaRepository` without altering the correctness of the program. This means instances where `JpaRepository` is expected can seamlessly work with `PartyRepository`, as it promises to fulfill the contract defined by `JpaRepository`, ensuring that the behaviors of the methods like `findAll`, `findByName`, `save`, and `deleteById` are consistent with those defined in the superinterface.



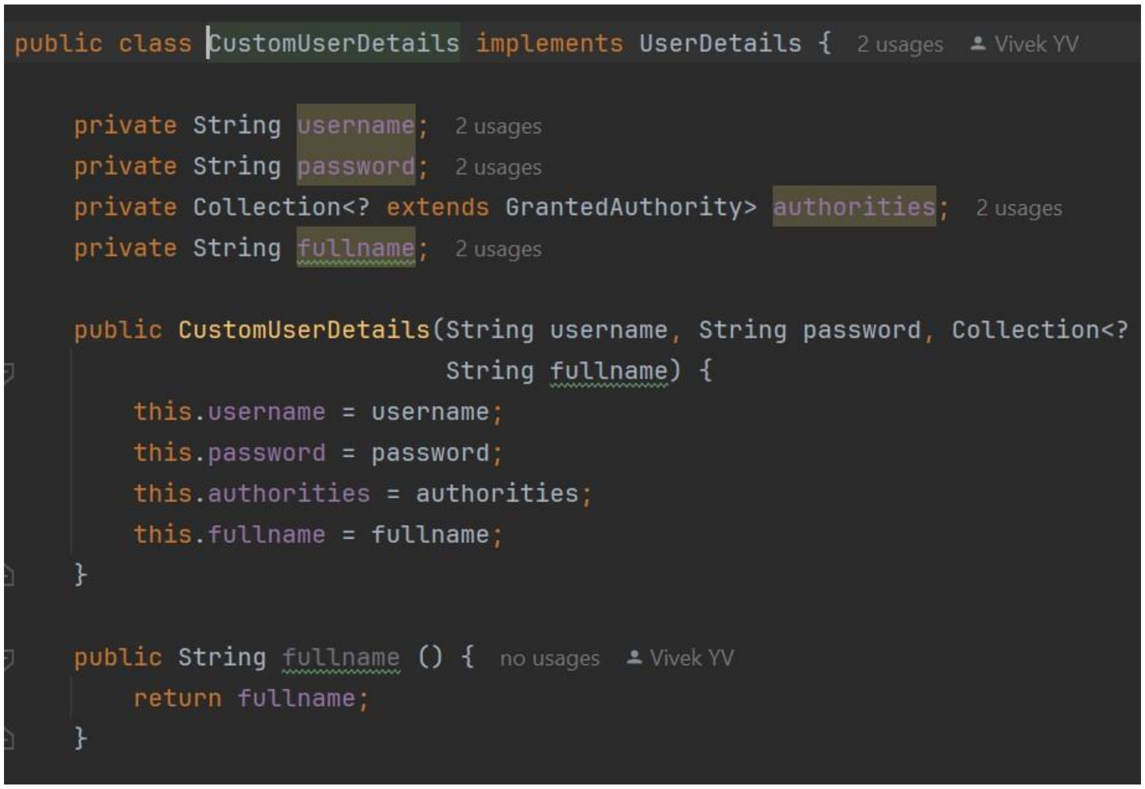
```
public interface PartyRepository extends JpaRepository<Party, Long> {

    List<Party> findAll();  Vivek YV

    Party findByName(String name);  1 usage  Vivek YV

    Party save(Party party);  Vivek YV

    void deleteById(Long id);  Vivek YV
}
```

4) Dependency Inversion Principle

The `CustomUserDetailsService` class exemplifies the Dependency Inversion Principle. This class depends on the abstraction `UserRepository` rather than concrete implementations for user data retrieval. By injecting `UserRepository` via the constructor, `CustomUserDetailsService` can work with any implementation of `UserRepository`, allowing for flexible and interchangeable backend data sources without needing to change the service layer code.



```
@Service  2 usages  Vivek YV
public class CustomUserDetailsService implements UserDetailsService {

    private UserRepository userRepository;  2 usages

    public CustomUserDetailsService(UserRepository userRepository) {  no usages  Vivek YV
        super();
        this.userRepository = userRepository;
    }

    @Override  no usages  Vivek YV
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("Username or Password not found");
        }
        return new CustomUserDetails(user.getUsername(), user.getPassword(), authorities(user.
    }

    public Collection<? extends GrantedAuthority> authorities(String role) {  1 usage  Vivek YV
        return Arrays.asList(new SimpleGrantedAuthority(role));
    }

}
```

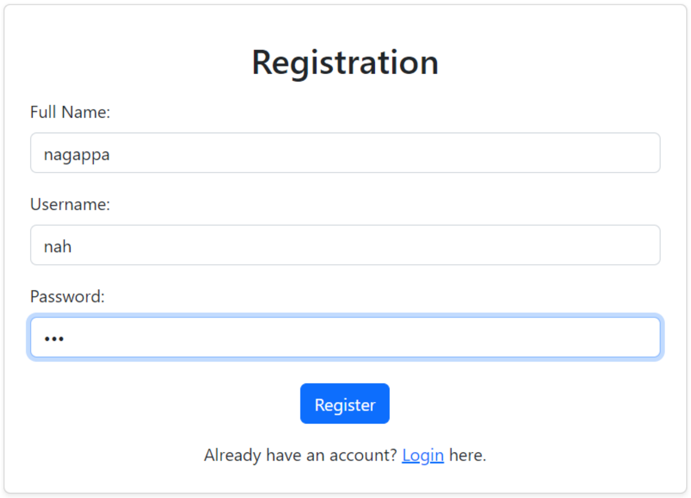


```
public class CustomUserDetails implements UserDetails {   2 usages   Vivek YV

    private String username;   2 usages
    private String password;   2 usages
    private Collection<? extends GrantedAuthority> authorities;   2 usages
    private String fullname;   2 usages

    public CustomUserDetails(String username, String password, Collection<?
                             String fullname) {
        this.username = username;
        this.password = password;
        this.authorities = authorities;
        this.fullname = fullname;
    }

    public String fullname () {   no usages   Vivek YV
        return fullname;
    }
```

9. SAMPLE OUTPUT DEMO SCREENSHOTS:

User Interface Screenshots:

1. Registration page



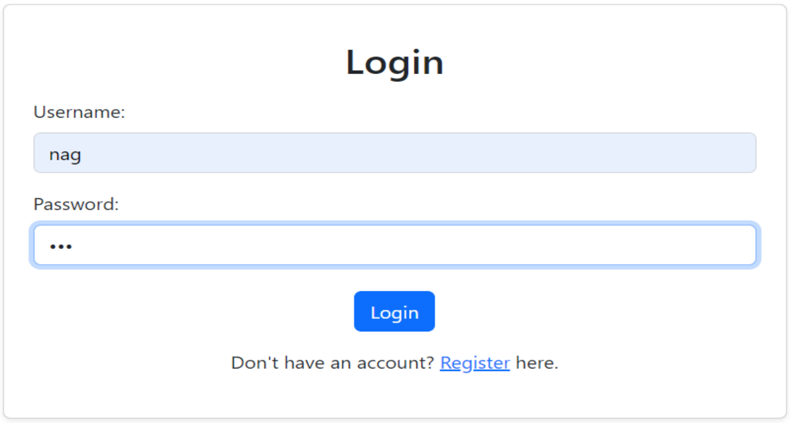
Registration

Full Name: nagappa

Username: nah

Password: •••

2. Login Page

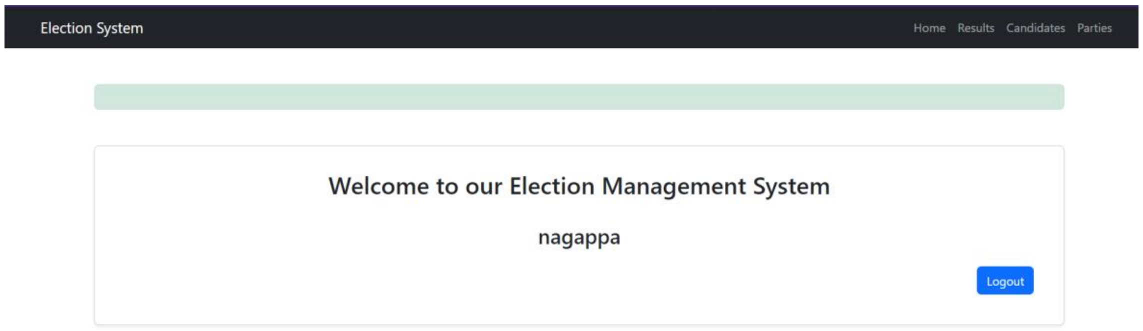


Login

Username: nag

Password: •••

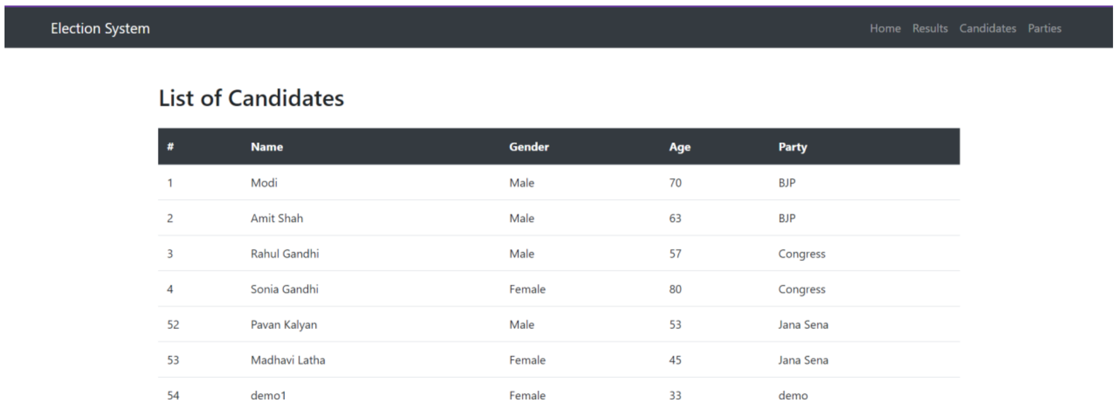
2. User home page:



Welcome to our Election Management System

nagappa

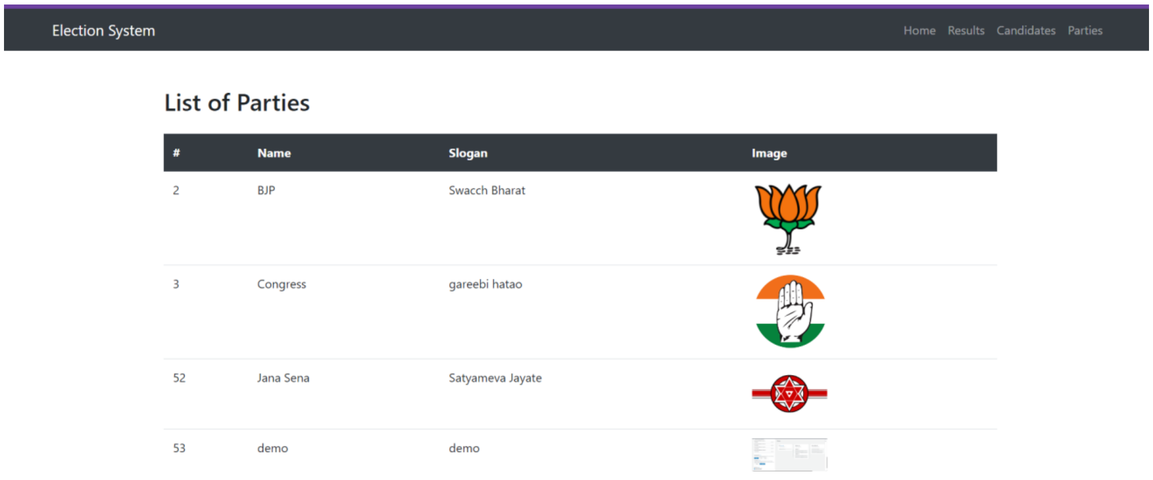
3. Candidates page:



List of Candidates

#	Name	Gender	Age	Party
1	Modi	Male	70	BJP
2	Amit Shah	Male	63	BJP
3	Rahul Gandhi	Male	57	Congress
4	Sonia Gandhi	Female	80	Congress
52	Pavan Kalyan	Male	53	Jana Sena
53	Madhavi Latha	Female	45	Jana Sena
54	demo1	Female	33	demo

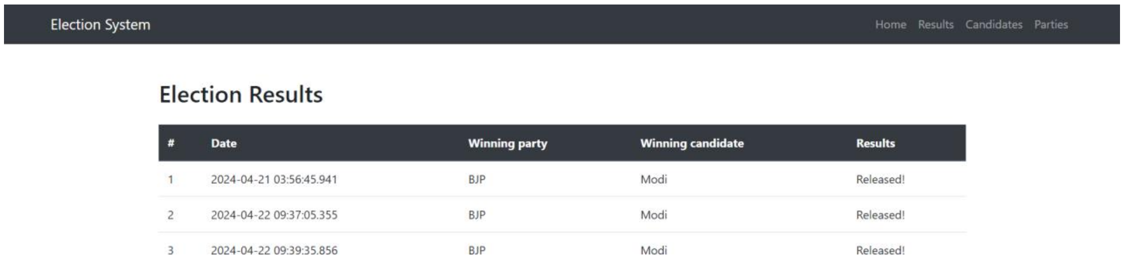
4. Party page:



List of Parties

#	Name	Slogan	Image
2	BJP	Swacch Bharat	
3	Congress	gareebi hatao	
52	Jana Sena	Satyameva Jayate	
53	demo	demo	

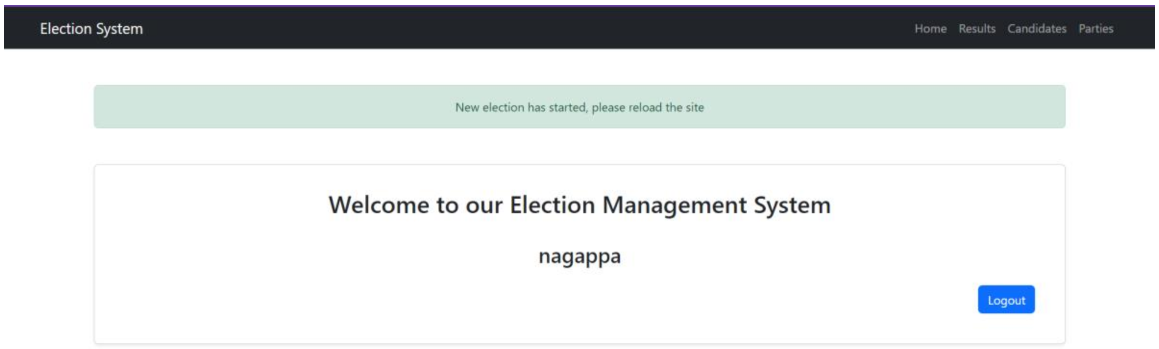
5. Results page:



Election Results

#	Date	Winning party	Winning candidate	Results
1	2024-04-21 03:56:45.941	BJP	Modi	Released!
2	2024-04-22 09:37:05.355	BJP	Modi	Released!
3	2024-04-22 09:39:35.856	BJP	Modi	Released!

6. Interface after hosting election:


Election System

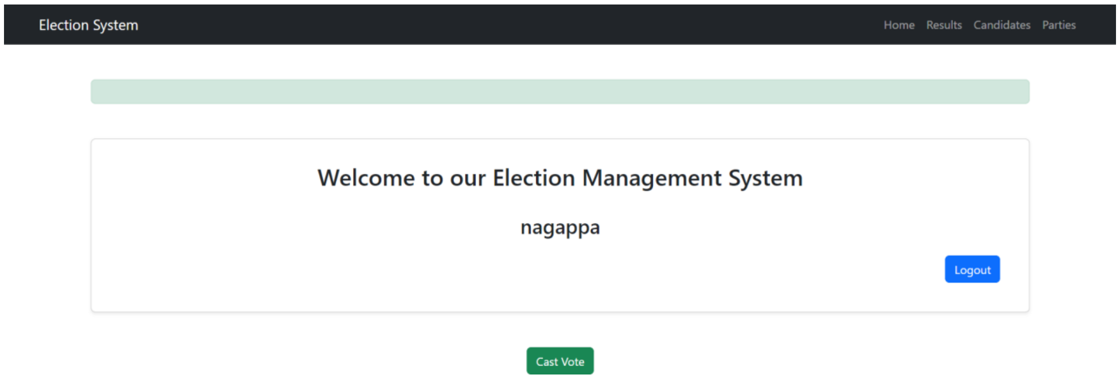
Home Results Candidates Parties

New election has started, please reload the site

Welcome to our Election Management System

nagappa

Logout


Election System

Home Results Candidates Parties

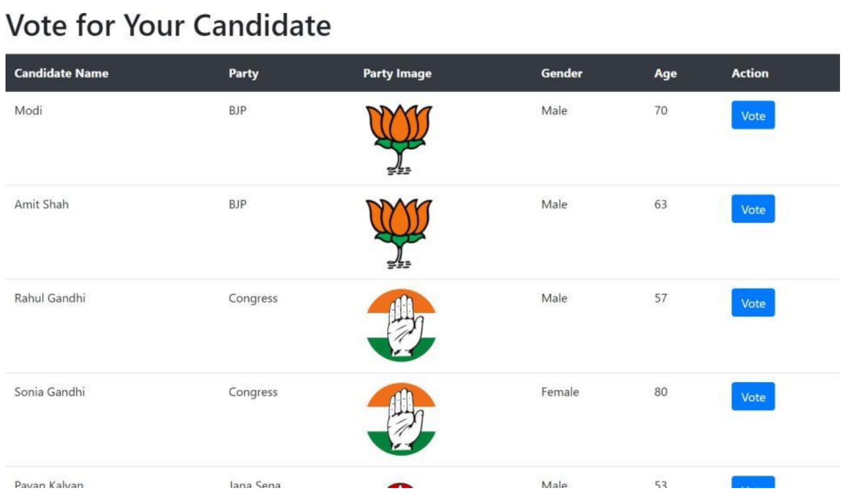
Welcome to our Election Management System

nagappa

Logout

Cast Vote

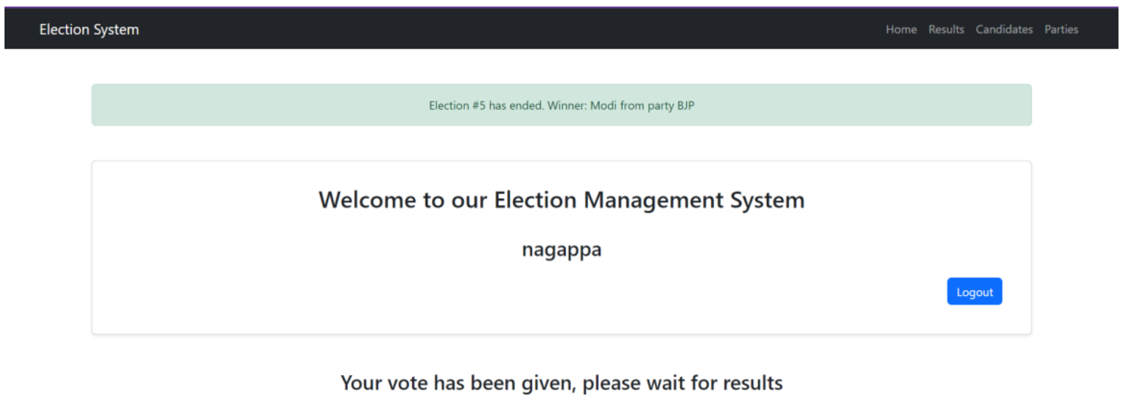
7. Polling page:



Vote for Your Candidate

Candidate Name	Party	Party Image	Gender	Age	Action
Modi	BJP		Male	70	Vote
Amit Shah	BJP		Male	63	Vote
Rahul Gandhi	Congress		Male	57	Vote
Sonia Gandhi	Congress		Female	80	Vote
Pavan Kalyan	Jana Sena		Male	53	

8. Page after end of election:



Election System

Election #5 has ended. Winner: Modi from party BJP

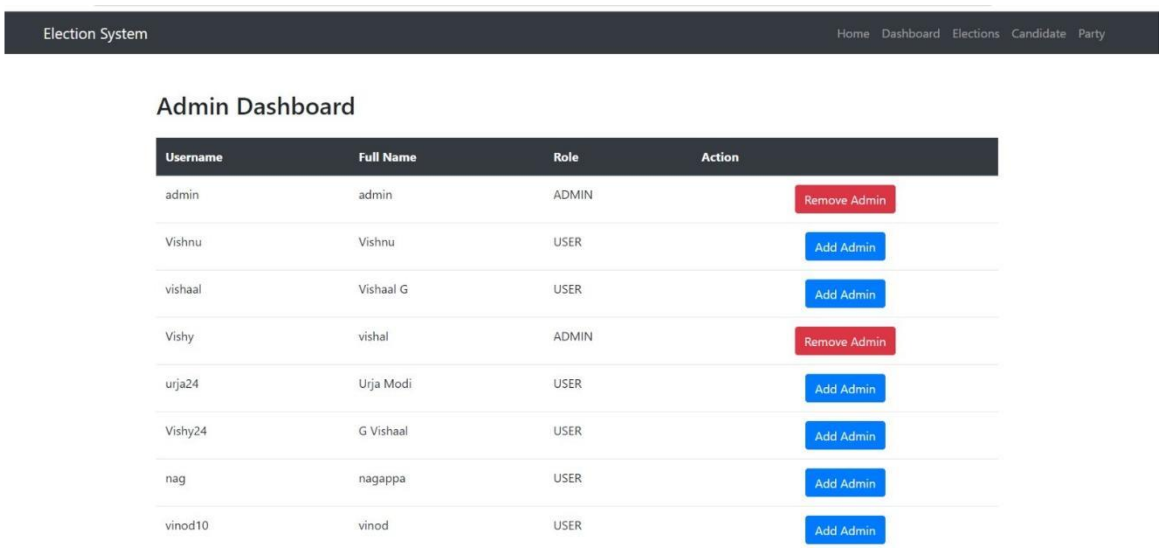
Welcome to our Election Management System

nagappa

Your vote has been given, please wait for results

Admin Interface Screenshots:

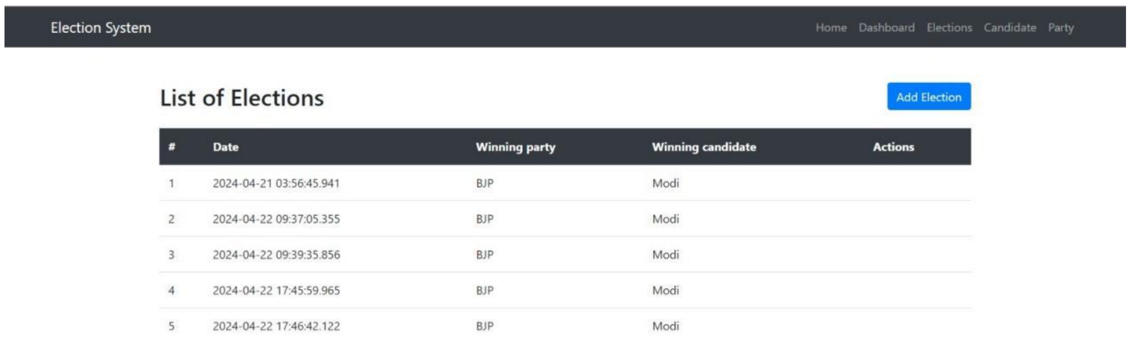
9. Admin Dashboard:



Admin Dashboard

Username	Full Name	Role	Action
admin	admin	ADMIN	Remove Admin
Vishnu	Vishnu	USER	Add Admin
vishaal	Vishaal G	USER	Add Admin
Vishy	vishal	ADMIN	Remove Admin
urja24	Urja Modi	USER	Add Admin
Vishy24	G Vishaal	USER	Add Admin
nag	nagappa	USER	Add Admin
vinod10	vinod	USER	Add Admin

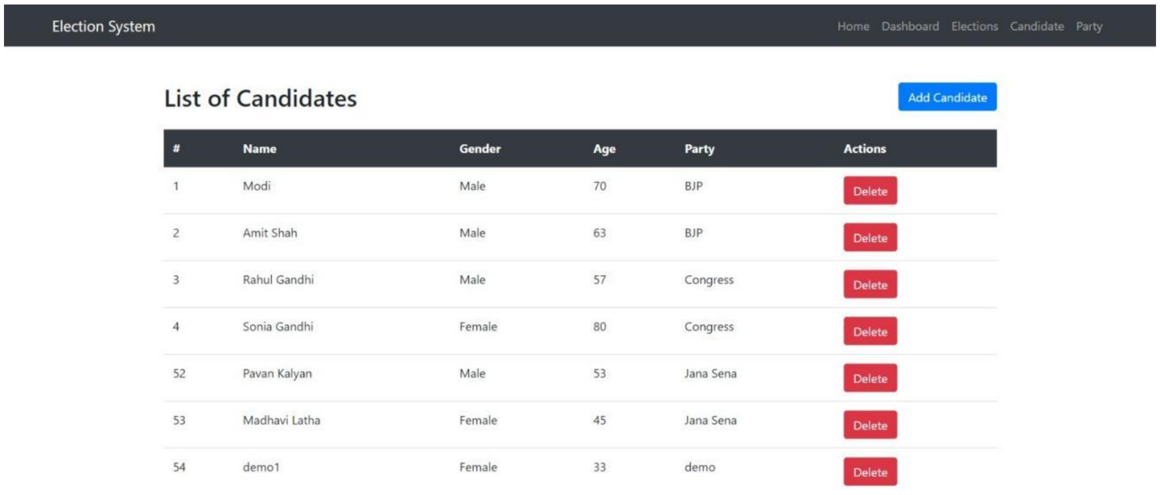
8. Election page (Admin perspective):



List of Elections

#	Date	Winning party	Winning candidate	Actions
1	2024-04-21 03:56:45.941	BJP	Modi	
2	2024-04-22 09:37:05.355	BJP	Modi	
3	2024-04-22 09:39:35.856	BJP	Modi	
4	2024-04-22 17:45:59.965	BJP	Modi	
5	2024-04-22 17:46:42.122	BJP	Modi	

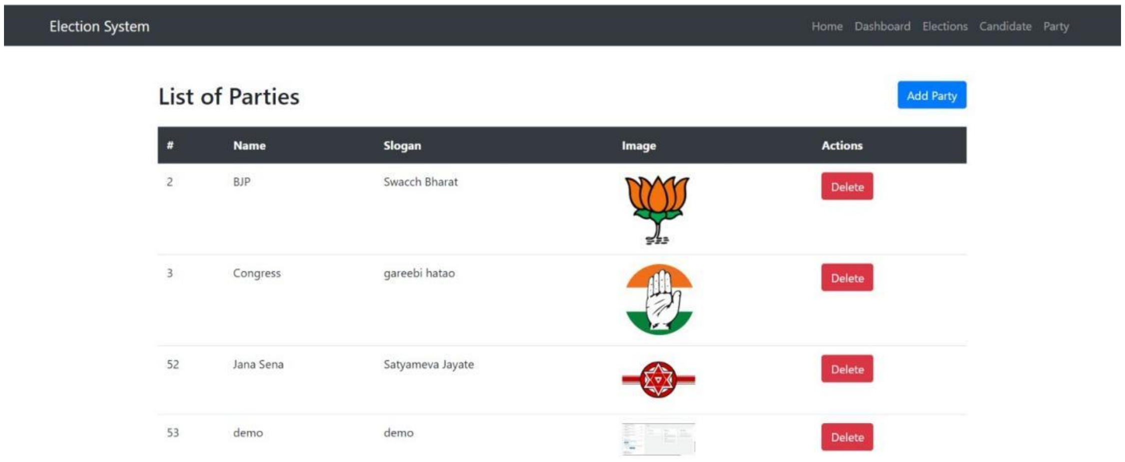
9. Add candidate:



List of Candidates

#	Name	Gender	Age	Party	Actions
1	Modi	Male	70	BJP	Delete
2	Amit Shah	Male	63	BJP	Delete
3	Rahul Gandhi	Male	57	Congress	Delete
4	Sonia Gandhi	Female	80	Congress	Delete
52	Pavan Kalyan	Male	53	Jana Sena	Delete
53	Madhavi Latha	Female	45	Jana Sena	Delete
54	demo1	Female	33	demo	Delete

10. Add party:



List of Parties

#	Name	Slogan	Image	Actions
2	BJP	Swacch Bharat		Delete
3	Congress	gareebi hatao		Delete
52	Jana Sena	Satyameva Jayate		Delete
53	demo	demo		Delete

Github link:

https://github.com/wolverkm/ElectionManagementSystem_JavaSpringBoot/tree/master/Spring-Project-main