A PROJECT REPORT

On

Online Voting Survey

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE

By:

AMANUL HAQUE (100210112)

MUKESH KUMAR (100210137)

SHAH ALAM KHAN (100210153)

Under the guidance of

Er. Beenu Yadav

(Assistant Professor, Computer Science)

NAAC A++ Accredited

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SIR CHHOTU RAM INSTITUTE OF ENGINEERING & TECHNOLOGY

CHAUDHARY CHARAN SINGH UNIVERSITY, MEERUT

(2024-25)

ACKNOWLEDGEMENT

First and foremost, I express my deepest gratitude to the almighty for his boundless blessing and guidance throughout the duration of this project. His unwavering support has been the cornerstone of my strength and perseverance.

We extend our thanks to **Prof. (Dr.) Niraj Singhal Director** and Co-ordinator **Er. Milind, Coordinator, Department of Computer Science, Sir Chhotu Ram Institute of Engineering & Technology, Chaudhary Charan Singh University, Meerut,** for giving us the motivation and guidance to complete this project. We are deeply indebted to our project guide **Er. Beenu Yadav, Department of Computer Science, Sir Chhotu Ram Institute of Engineering & Technology, Chaudhary Charan Singh University, Meerut** for the initial idea of the project and for all the guidance and encouragement she gave in the subsequent months. Her supervision and co-operation at every stage helped us in successfully completing the project. Whatever intellectual effort may be reflected from this project is the direct result of the informative and stimulating discussions and suggestion, she has given during the course of the project. Lastly, we would like to thank the entire staff of the Computer Science Department and our college for their support and co-operation in the completion of this project.

Mukesh Kumar (100210137)
mukeshkumarrajput0111@gamil.com
(Student Signature)

Amanul Haque (100210112)
amanulhaue376@gmail.com
(Student Signature)

Shah Alam Khan (100210153)
shahakmgs@gamil.com
(Student Signature)

SIR CHHOTU RAM INSTITUTE OF ENGINEERING & TECHNOLOGY

CHAUDHARY CHARAN SINGH UNIVERSITY, MEERUT

Approved by A.I.C.T.E., New Delhi

Student Declaration

This is to certify that the project entitled **"Online Voting Survey"** in Meerut is submitted in partial fulfillment of the requirement of the degree of **Bachelor of Technology** in **Computer Science** of **Sir Chhotu Ram Institute of Engineering and Technology, Chaudhary Charan Singh University Campus, Meerut (U.P)** under the supervision of Er. Beenu Yadav.

Student Name	Roll Number
Mukesh Kumar	100210137
Amanul Haque	100210112
Shah Alam Khan	100210153

SIR CHHOTU RAM INSTITUTE OF ENGINEERING & TECHNOLOGY

CHAUDHARY CHARAN SINGH UNIVERSITY, MEERUT

Approved by A.I.C.T.E., New Delhi

Certificate

The project report entitled **"Online Voting Survey"** in Meerut is submitted by Mukesh Kumar (100210137), Amanul Haque (100210112), Shah Alam Khan (100210153),It warrants its acceptance as a prerequisite for the degree of **Bachelor of Technology** in **Computer Science** of **Sir Chhotu Ram Institute of Engineering and Technology, Chaudhary Charan Singh University Campus, Meerut (U.P)**.

(Er. Ritu Sharma) **( )**

Internal Examiner **External Examiner**

(Er. Beenu Yadav)

Project Guide

(Er. Milind)

(Co-ordinator)

(Department of Computer Science)

ABSTRACT

The Voter Survey Application is a web-based platform designed to collect, analyse, and visualize voter opinions and feedback regarding political candidates, election issues, and public policies. The primary objective of the application is to provide a seamless and user-friendly experience for both voters and administrators, ensuring secure participation and real-time insight into survey results. The project is divided into two main modules: the User Module and the Admin Module. Users can register or log in securely, participate in active surveys, and view summarized survey outcomes once participation is completed. Administrators have the capability to create, edit, and delete surveys, monitor participation rates, and access detailed analytics through graphical dashboards. The application emphasizes security with proper authentication mechanisms like JWT (JSON Web Token) and ensures that user data is securely stored in the backend database.

Technologically, the frontend is built using HTML, CSS, JavaScript, and optionally React.js for a more dynamic experience. The backend is developed using Node.js or Java Spring Boot, with APIs to handle authentication, survey CRUD operations, and result aggregation. MongoDB or MySQL serves as the primary database to store user details, survey questions, and responses. For data visualization, libraries like Chart.js or Recharts are integrated to present real-time results in various formats such as bar charts and pie charts.

Throughout the project development, major software engineering practices like modular coding, API-based architecture, error handling, and database optimization have been followed. Testing is conducted using both manual and automated methods to ensure a bug-free and reliable system. Finally, the application is deployed on cloud platforms like Render or, ensuring high availability and scalability.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER - 1
INTRODUCTION

INTRODUCTION

The **Voting Survey** project is a comprehensive web-based solution that provides a RESTful API for managing various aspects of voting and survey systems. It is designed to ensure secure, efficient, and scalable handling of user data and electoral processes. The system supports key functionalities including.

1.1 INTRODUCTION OF THE PROJECT

This project is developed keeping in mind real-world electoral system needs, providing a robust backend system that can be easily connected with mobile or web interfaces. It aims to reduce manual work, enhance transparency, and ensure effective data handling in voting-related processes:

- **User Registration and Authentication**: Allows new users to register and existing users to log in securely using authentication protocols.
- **Constituency Management**: Administrators can create, update, and manage constituencies, enabling region-based organization of voting data.
- **Admin Functionality**: Includes features for administrative users to manage and monitor all user activities and data related to voting surveys.
- **Secure Data Handling**: Ensures user privacy and security with proper validation and encryption techniques.
- **RESTful API Architecture**: Enables seamless integration with frontend applications or third-party services, allowing flexible usage and expansion.

1.2 OBJECTIVES AND SCOPE OF THE PROJECT

The primary objective of the **Voting Survey** project is to develop a secure and efficient system for managing voter registration, authentication, and constituency-related data through a RESTful API. It aims to streamline the electoral process by offering a centralized platform for users and administrators to interact with voting and survey information. The system enhances transparency, reduces manual intervention, and promotes digital participation in democratic processes. By leveraging modern technologies, the project provides a scalable solution that ensures data integrity, role-

based access, and real-time survey insights to support informed decision-making and policy evaluation:

- To develop a secure RESTful API for user registration, login, and constituency management.
- To facilitate efficient survey and voting data collection and analysis.
- To implement role-based access for users and administrators to maintain data integrity.
- To support scalable and flexible integration with frontend or third-party applications.

These objectives collectively aim to enhance the accuracy, reliability, and efficiency of real time.

Scope

The Voting Survey project provides a secure and scalable platform for managing voter data, constituencies, and surveys. It supports user and admin roles, enables real-time data handling, and can be integrated with various frontend applications for wider accessibility:

- Supports secure user registration, login, and authentication.
- Enables administrators to manage constituencies and user data.
- Allows integration with frontend applications (web/mobile) via RESTful APIs.
- Facilitates real-time collection and analysis of voting and survey data.
- Implements role-based access control for users and administrators.
- Designed for scalability to accommodate increasing numbers of users and data.
- Useful for educational institutions, organizations, and local electoral bodies.

By defining the scope of the project, we aim to establish clear boundaries and deliverables, ensuring that the development process remains focused and aligned with the project objectives and requirements.

1.3 PROBLEM STATEMENT

In current electoral and survey systems, managing voter registrations, voting data, and constituency-related information is often cumbersome, error-prone, and manually intensive. There is a need for a secure, efficient, and scalable solution to streamline the process of

collecting and managing voting and survey data. Existing systems lack centralized, transparent, and easy-to-use platforms for administrators and users alike. The **Voting Survey** project aims to address these issues by providing a RESTful API that simplifies user authentication, constituency management, and data collection, thereby improving efficiency and transparency in voting-related processes.

In many existing electoral and survey systems, managing voter registrations, voting processes, and constituency data is a complex and error-prone task. Traditional methods often involve manual data entry, which can lead to inaccuracies, inefficiencies, and delays. Additionally, these systems are not always secure, making them vulnerable to fraud or unauthorized access. Many solutions lack flexibility and are unable to scale efficiently, especially when handling large volumes of data or real-time interactions. The absence of a centralized, transparent platform data is a complex and error-prone task. Traditional methods often involve manual data entry, which can lead to inaccuracies, inefficiencies, and delays. Additionally, these systems are not always secure, making them vulnerable to fraud or unauthorized access. Many solutions makes it difficult for both administrators and voters to interact effectively with the system.

Given these challenges, there is an urgent need for an automated, secure, and scalable solution that simplifies the registration, login, and voting processes. The **Voting Survey** project aims to address these issues by providing a RESTful API that ensures secure user authentication, efficient management of voting data, and streamlined administration of constituencies. With this approach, the project seeks to eliminate manual processes, enhance transparency, and facilitate better decision-making by making voting data easily accessible and manageable in a centralized platform.

- **Manual Processes and Errors**: Traditional voting systems rely heavily on manual entry and physical documentation, leading to potential human errors, inefficiencies, and delays in data processing.
- **Data Inaccessibility and Lack of Transparency**: In many electoral systems, data related to voter registrations and surveys is fragmented, inaccessible, and not easily shareable, reducing transparency and trust in the system.

- **Security Concerns**: Many existing systems do not implement strong security measures, making them susceptible to data breaches, fraud, and unauthorized access, threatening the integrity of the voting process.
- **Lack of Scalability**: Most legacy voting systems struggle to handle large-scale elections or surveys, making it difficult to manage a growing number of voters or survey participants efficiently.
- **Inefficient Data Management**: Without a centralized platform, tracking voting patterns, survey responses, or constituency-specific information becomes cumbersome, leading to a slower response time in decision-making.

CHAPTER - 2

PROPOSED SYSTEM

PROPOSED SYSTEM

The Voting Survey system is designed to address the key challenges in traditional voting and survey systems by providing an integrated, secure, and scalable solution. It leverages a RESTful API to manage all core functionalities, ensuring smooth interaction between the backend and frontend applications. The system consists of several key components.

3.1 SYSTEM DESCRIPTION

The **Voting Survey** system is a modern, secure, and efficient platform designed to streamline the process of managing voter data, conducting surveys, and managing constituencies through a RESTful API. The system is modular and scalable, ensuring it can meet the needs of both small-scale elections and larger voting or survey processes.

The **Voting Survey** system offers an integrated, scalable, and secure solution for managing voter data and conducting surveys. By leveraging modern web technologies, RESTful APIs, and cloud infrastructure, the system process of managing voter data, conducting surveys, and managing constituencies through a RESTful API ensures that voting and survey processes are efficient, secure, and easy to manage, making it suitable leveraging modern web technologies, RESTful APIs, and cloud infrastructure, the system for a wide range of applications in elections, surveys, and organizational decision-making.

3.2 FEATURES

The Proposed Real-Time Application offers a range of features to provide seamless, secure, and interactive communication:

- **Secure and Scalable**: Focuses on security with encrypted data storage and transfer, and scalability to handle large-scale voter data.
- **Transparency**: Ensures transparent voting and survey results through accurate, real-time data analysis.
- **Role-Based Permissions**: Different user roles (admin, voter) to ensure proper access control and data security.

This comprehensive set of features ensures that the application is efficient, secure, and adaptable, making it suitable for various real-time communication needs.

3.3 SYSTEM ADVANTAGES

The **Voting Survey** system offers several advantages that make it a robust solution for managing voting and survey data. Firstly, it ensures **enhanced security** by utilizing encrypted data storage and secure communication channels, protecting voter information and preventing unauthorized access. The system is highly **scalable**, designed to handle a growing number of users and data, which is crucial during large elections or extensive surveys. With **real-time data collection**, administrators can monitor results immediately, enabling timely decision-making. Additionally, the system offers **role-based access control**, ensuring that only authorized users can access sensitive data:

- **Enhanced Security**: Utilizes encrypted user data and secure communication channels (HTTPS) to ensure the integrity and privacy of voting and survey data, minimizing the risk of fraud or unauthorized access.
- **Scalability**: The system is designed to scale seamlessly, supporting a growing number of users and data through cloud-based infrastructure, ensuring performance remains optimal even during large elections or extensive surveys.
- **Efficient Data Management**: By centralizing voter registrations, voting data, and surveys, the system allows for efficient management and easy retrieval of information, significantly reducing administrative overhead.
- **Role-Based Access Control (RBAC)**: Provides secure, granular control over user permissions, ensuring that different types of users (e.g., voters, administrators) can access only the information they are authorized to view or modify.

These advantages ensure that application is efficient, secure, and adaptable to a wide range of use cases.

CHAPTER - 3

SYSTEM ANALYSIS

SYSTEM ANALYSIS

System analysis is the process of understanding and evaluating how the **Voting Survey** system will function to meet user requirements, improve efficiency, and solve real-world problems related to managing voter registrations, surveys, and constituency data. This section includes analysis of the existing problems, user requirements, system feasibility, and proposed system design.

3.1 Problem Identification

- Data is often managed manually, which is **time-consuming and error-prone**.
- There is **limited security** and a high risk of data tampering or unauthorized access.
- Lack of **real-time data visibility** for administrators.
- Difficult to **scale** or modify based on requirements.

System Requirements Analysis

User Interface (UI): The client-side architecture includes the user interface components built using HTML, CSS, and JavaScript. This encompasses the layout, design, and interactivity of the chat application, including features such as message input, message display, and user authentication.

Server-Side Architecture

In the server-side architecture of the real-time chat application project, several components work together to facilitate communication between clients, manage user connections, and handle message transmission. Here's a description of the server-side architecture based on the provided code:

Node.js Server

The server-side architecture is powered by Node.js, a runtime environment for executing JavaScript code. The Node.js server hosts the application logic, handles client requests, and facilitates real-time communication between clients.

Non-Functional Requirements

- High security and privacy for all data transactions.
- Fast response time and high availability.
- Scalability to handle growing user demands.
- Cross-platform support for accessibility on desktop and mobile devices.

3.2 Technologies used in the application

The Real-Time Chat Application leverages a combination of modern web technologies to deliver its functionality. Here are the key technologies used in the development of the application:

3.2.1. Front-End Technologies

Front-end technologies are the tools and languages used to develop the user interface (UI) and user experience (UX) of a web application. These technologies focus on what the user interacts with directly in their web browser. Here's a description of some key front-end technologies commonly used in web development:

- **HTML (Hyper Text Markup Language)**

  In this real-time chat application project, HTML (Hyper Text Markup Language) is used to structure the content and layout of web pages.

- **CSS (Cascading Style Sheets)**

  Used for styling and presentation of HTML elements. Defines the visual appearance, layout, and responsiveness of the chat interface. CSS is applied to the entire layout of the application, including the navigation, message container, input form, and buttons. Styles are used to define the overall appearance, such as background colours, font families, and spacing., padding, margin, and overflow are used to create a visually pleasing container for displaying chat messages. Background colour, font size, and font colour are adjusted to improve readability and aesthetics.

- **JavaScript**

JavaScript is a scripting language used to add interactivity and dynamic behavior to web pages. In your project, JavaScript plays a crucial role in handling client-side functionality such as:

Appending new messages to the message container dynamically without reloading the page. Handling user interactions like submitting messages, joining the chat, and notifying users about new messages. Initiating audio calls through WebRTC (Real-Time Communication) for peer-to-peer communication.

3.2.2 Back-End Technologies

For the back-end of your real-time chat application project, you've already provided the server-side code using Node.js and Socket.IO. Let's describe these technologies in more detail:

- **Node.js**

  Node.js is a runtime environment that allows you to run JavaScript on the server-side. It provides an event-driven, non-blocking I/O model that makes it lightweight and efficient, ideal for real-time applications.

- **Express.js**

  A web application framework for Node.js used to build robust APIs and web applications. Simplifies routing, middleware integration, and handling HTTP requests and responses.

- **Socket.IO**

A library that enables real-time, bidirectional, and event-based communication between web clients and servers. Facilitates instant messaging and updates among connected clients in the chat application.

3.3 Feasibility Study

The feasibility study assesses the practicality and viability of developing and deploying the Voting Survey Application. The technical feasibility is strong, as the project utilizes widely adopted technologies like HTML, CSS, JavaScript, and a backend framework (e.g.,

Node.js, Python Django/Flask) that are well-supported and compatible with most systems. The application is lightweight and can be hosted on affordable cloud platforms, ensuring high availability and scalability. From a economic feasibility perspective, the cost of development is minimal, as it primarily involves open-source tools and free-tier cloud services. This makes it budget-friendly for educational institutions or small organizations. The operational feasibility is also positive, since the system offers a user-friendly interface for voters and administrators, reducing the need for extensive training. It supports easy deployment and maintenance, ensuring it can be managed with limited technical expertise. Finally, legal and ethical feasibility is ensured by incorporating secure login mechanisms, protecting user data, and allowing transparent voting processes. Overall, the system is both feasible and sustainable for real-world use:

- **Operational Feasibility**: The system is easy to operate, with intuitive interfaces and clear workflows, ensuring it can be used by both tech-savvy and non-technical users.
- **Economic Feasibility**: Since the system reduces manual tasks, printing, and administrative costs, it is economically viable and cost-efficient in the long term.
- **Legal Feasibility**: The system can comply with data privacy laws (e.g., GDPR, if needed) by ensuring user data is encrypted and access is controlled.

CHAPTER - 4

SYSTEM DESIGN

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE DIAGRAM

The system architecture diagram of the **Voting Survey** project represents how different components interact with each other to form a complete, secure, and efficient system. Here's a breakdown of the main elements.

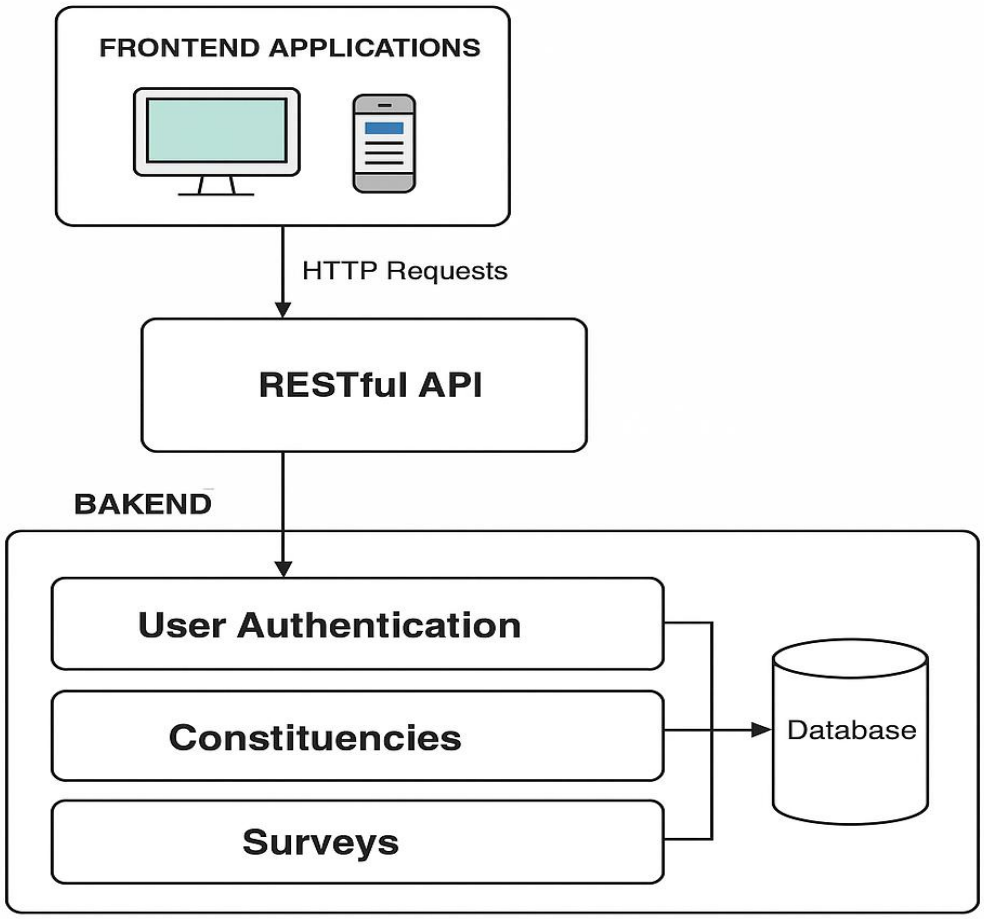

Fig.4.1 System Architecture Design

Reference: https://www.interviewbit.com/blog/system-architecture/

4.2 DATA FLOW DIAGRAM

Textual description of a **Data Flow Diagram (DFD)** for your **Voting Survey System**. You can use this to visualize or draw the DFD at **Level 0 and Level 1**.

Level 0 Diagram Structure:

This is the highest-level view of the system. It shows how the system interacts with external entities.

- User submits registration/login → System
- User sends vote/survey responses → System
- Admin sends/receives constituency & result data → System
- System sends results/status back to Admin
- System fetches/stores data in Databases

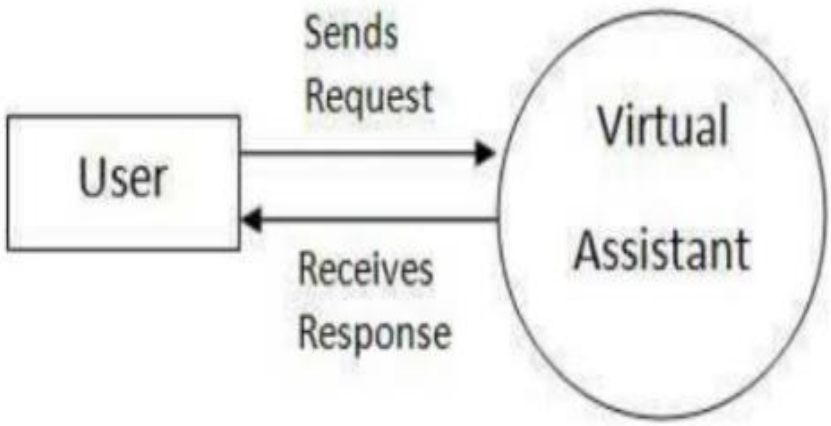

Fig. 4.2 Data Flow Diagram

Reference: https://www.ibm.com/think/topics/data-flow-diagram

Level 1 DFD:

A Level 1 DFD expands on the context-level (Level 0) diagram by breaking down the main system into sub-processes, showing the flow of data between users, subsystems, and data stores.

Level 1 expands the main system into sub-processes:

Processes:

1. 1.0 User Registration & Login
2. 2.0 Cast Vote / Fill Survey
3. 3.0 Admin Management
4. 4.0 Result Processing

Data Stores:

- D1: User DB
- D2: Survey & Voting DB
- D3: Constituency DB

4.3 WORK FLOW DIAGRAM

The workflow of the Voting Survey Application begins with two primary user roles: regular users and administrators. Regular users first go through the registration or login process to access the platform. Once logged in, they are directed to a dashboard where they can view and select from available surveys or voting polls.

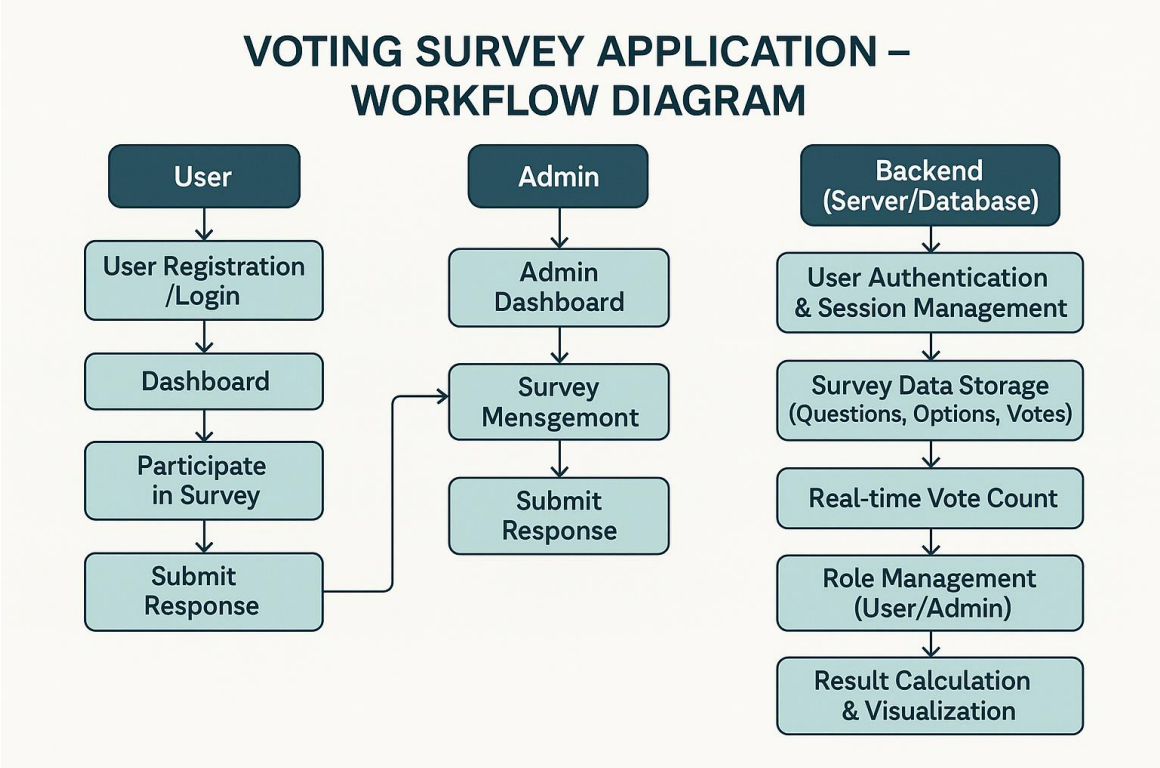

Fig.4.3 Work Flow Diagram

Reference: https://www.creatio.com/glossary/workflow-diagram

4.4 Entity Relationship Diagram

This ERD (Entity Relationship Diagram) explains how a voting system is used legally. It illustrates the relationship between different sections of the voting process, such as casting votes, counting and verifying them, tabulating the results, and announcing the winner. The data displayed on the ERD makes it easy to identify the different entities, attributes, and constraints associated with each step of the voting process. By analyzing this ERD, one can gain a comprehensive understanding of the workings of a modern voting system, helping to ensure that the voting process is carried out according to the law.

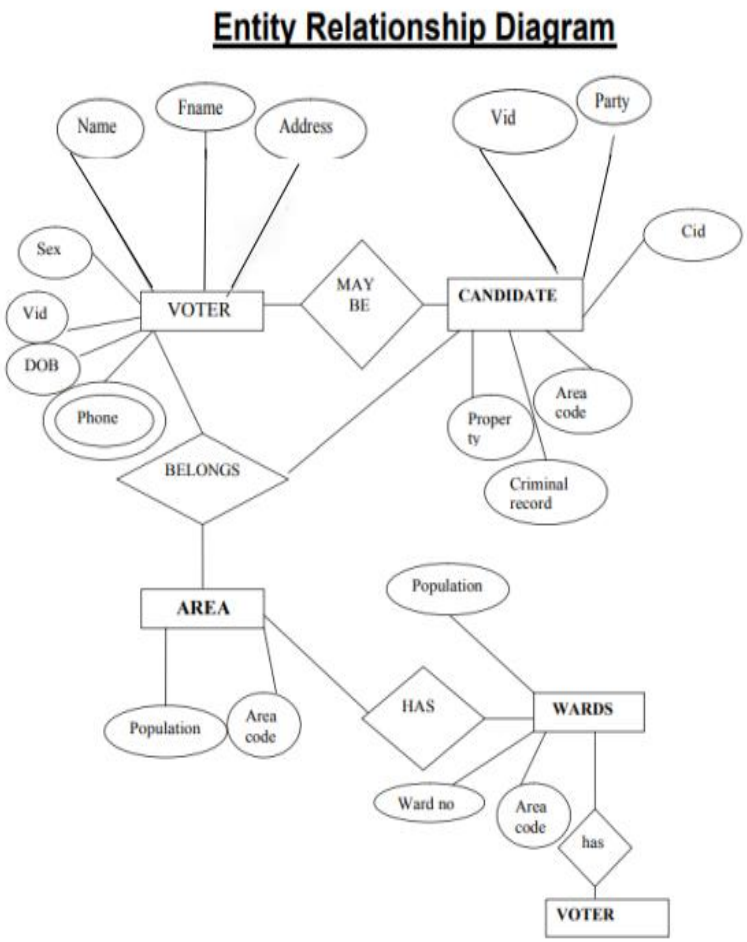

Fig.3.4 Entity Relationship Diagram

Reference: https://www.edrawsoft.com/er-diagram-examples.html

4.5 Activity Diagram

The activity begins when a user accesses the voting survey application. The first step involves user authentication, where the user either logs in with existing credentials or registers for a new account. If authentication fails, the system prompts the user to retry or register. Once successfully logged in, the system displays a list of available surveys. The user selects a desired survey to participate in, after which the system shows the survey details and available voting options.

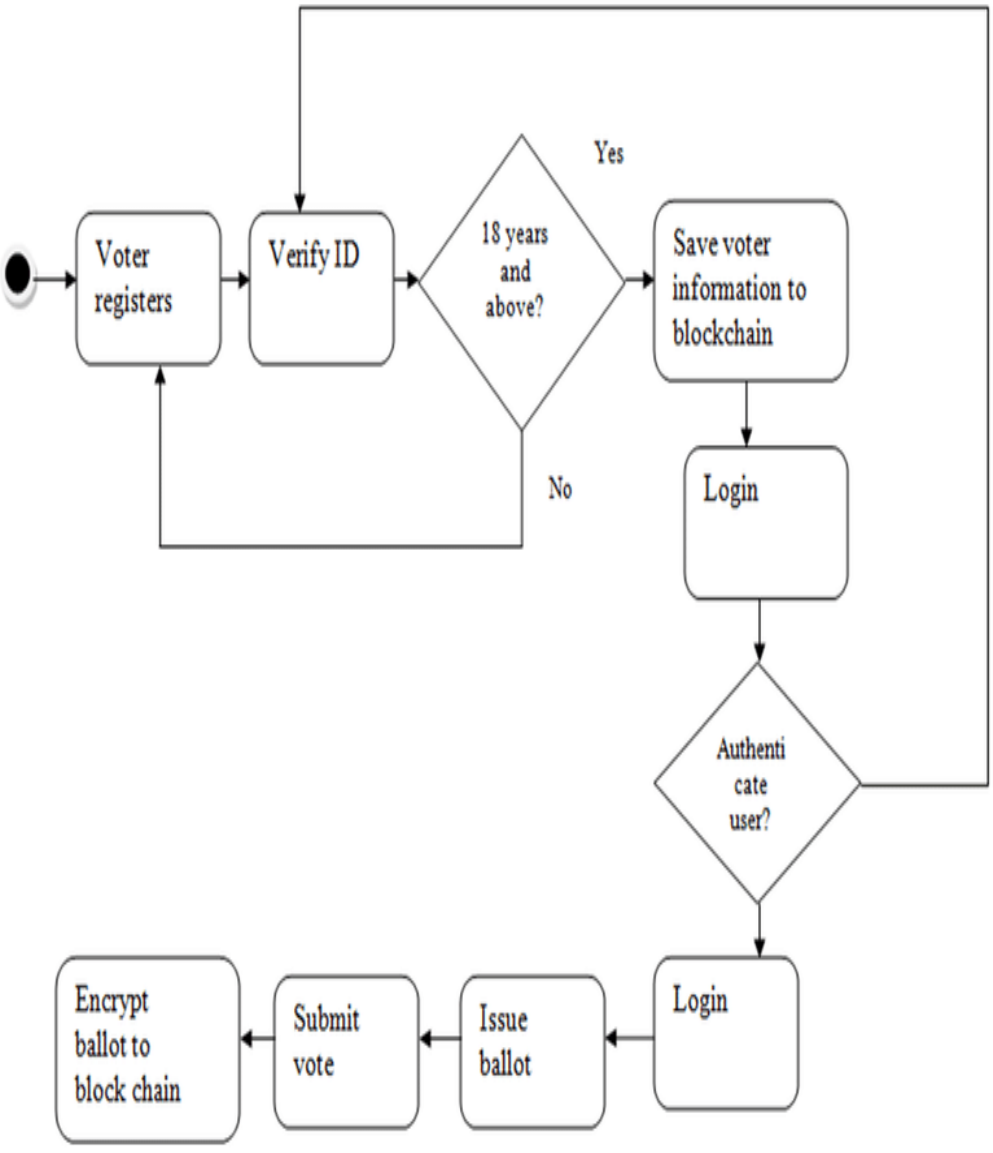

fig.4.5 Activity Diagram

Reference: https://medium.com/@nynptel/understanding-activity-diagrams-a-visual-guide-to-system-process-modeling-ee46a0759e10

4.6 FUNCTIONALITY OVERVIEW

The **Voting Survey System** is designed to facilitate secure, efficient, and user-friendly digital voting and survey management. The system allows users to register and log in with unique credentials, ensuring that only authenticated voters can participate. Once logged in, users can view available surveys assigned to their constituency and cast their vote with a single click. The system ensures one-person-one-vote integrity by disabling further access to the same poll after submission. Administrators have elevated privileges that enable them to create, edit, and delete surveys, define constituencies, and manage voter access. They can also monitor real-time voting statistics and analyse results through visual reports. The platform supports role-based access control, ensuring that only authorized users can perform certain operations. It offers functionalities for both users (voters) and administrators, providing distinct features based on roles:

User Interaction

- **User Registration**
  Allows new users to create an account by providing required credentials.
- **User Login & Authentication**
  Secure login system to access voting and survey features.
- **View Constituency Details**
  Users can view their assigned constituency and related information.
- **Participate in Voting**
  Users can securely cast their votes within assigned constituencies.
- **Participate in Surveys**
  Users can fill out available surveys related to political, social, or election feedback.
- **View Voting/Survey Status**
  Users can check whether their vote/survey has been submitted.

Administrator Functionality

- **Admin Login**
  Secure admin authentication to manage system components.

- **Manage Users**

  View, verify, or remove users; assign users to constituencies.
- **Create and Manage Constituencies**

  Add or update constituency data and assign voters.
- **Create and Manage Surveys**

  Add survey questions, define deadlines, and manage responses.
- **Monitor Voting Activity**

  View live status of votes cast in each constituency.
- **Analyze Survey Results**

  Review and analyze survey responses in real-time or after submission.

System Features

- **RESTful API Integration**

  Enables smooth communication between frontend and backend.
- **Data Encryption & Secure Access**

  Ensures data privacy and system integrity.
- **Role-Based Access Control**

  Restricts features based on whether the user is a voter or an admin.
- **Real-Time Updates**

  Live tracking of vote counts and survey responses.
- **Scalability & Modularity**

  Easily extendable to accommodate new features or large user bases.

CHAPTER - 5

CODING

CODING

5.1 Controller

a) Admin Controller

```
package com.jspider.votingsurvey.controller;

import java.util.Map;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.jspider.votingsurvey.entity.Admin;
import com.jspider.votingsurvey.services.AdminService;

@RestController
@RequestMapping(value = "/api/admin")
@CrossOrigin(value = "http://localhost:5173")
public class AdminController {

    @Autowired
    private AdminService service; // Add @Autowired

    @GetMapping("/{id}") // Fix missing path parameter
    public Optional<Admin> getUserByIdController(@PathVariable(name = "id") Long
id) {
        return service.getAdminById(id);
    }
```

```
    @PostMapping(value = "/auth")
    public boolean authAdminByIdAndPassword(@RequestBody Map<String, String>
logingRequest) {
     Long id = Long.valueOf(logingRequest.get("id").toString());
     String password = logingRequest.get("password").toString();

     System.out.println("From Controller: Voter ID = " + id + ", Password = " +
password);
     return service.authAdminByIdAndPassword(id, password);
    }
   }
```

b) User Controller

```
package com.jspider.votingsurvey.controller;

import java.util.List;
import java.util.Map;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.HttpStatus;

import com.jspider.votingsurvey.entity.User;
import com.jspider.votingsurvey.services.UsersService;
```

```
@RestController
@RequestMapping(value = "/api/user")
@CrossOrigin(value = "http://localhost:5173")
public class UserController {

@Autowired
private UsersService service;

@PostMapping(value = "/register")
public ResponseEntity<?> saveUserController(@RequestBody User user) {
   if (user.getAge() < 18) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("You must be at least 18 years old to register.");
   }

   long vid = user.getVoterId();
   Optional<User> optional = service.getUserByVoterId(vid);

   if (optional.isPresent()) {
      return ResponseEntity.status(HttpStatus.CONFLICT).body("Voter ID already exists. Please use a different Voter ID.");
   }

   User savedUser = service.saveUser(user);
   return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}


@GetMapping
public List<User> getAllUserController(){
    return service.getAllUsers();
}
public class AdminController {

   @Autowired
   private AdminService service; // Add @Autowired
```

```
    @GetMapping("/{id}") // Fix missing path parameter
    public Optional<Admin> getUserByIdController(@PathVariable(name = "id") Long
id) {
        return service.getAdminById(id);
    }

    @PostMapping(value = "/auth")
    public boolean authAdminByIdAndPassword(@RequestBody Map<String, String>
logingRequest) {
     Long id = Long.valueOf(logingRequest.get("id").toString());
     String password = logingRequest.get("password").toString();

     System.out.println("From Controller: Voter ID = " + id + ", Password = " +
password);
     return service.authAdminByIdAndPassword(id, password);
    }
   }
```

c) User Controller

```
package com.jspider.votingsurvey.controller;

import java.util.List;
import java.util.Map;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import org.springframework.http.HttpStatus;

import com.jspider.votingsurvey.entity.User;
import com.jspider.votingsurvey.services.UsersService;

@RestController
@RequestMapping(value = "/api/user")
@CrossOrigin(value = "http://localhost:5173")
public class UserController {

@Autowired
private UsersService service;

@PostMapping(value = "/register")
public ResponseEntity<?> saveUserController(@RequestBody User user) {
   if (user.getAge() < 18) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("You must be at least 18 years old to register.");
   }

   long vid = user.getVoterId();
   Optional<User> optional = service.getUserByVoterId(vid);

   if (optional.isPresent()) {
      return ResponseEntity.status(HttpStatus.CONFLICT).body("Voter ID already exists. Please use a different Voter ID.");
   }

   User savedUser = service.saveUser(user);
   return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}


@GetMapping
public List<User> getAllUserController(){
    return service.getAllUsers();
}
```

```
@GetMapping(value = "/{id}")
public Optional<User> getUserByIdController(@PathVariable int id) {
    return service.getUserById(id);
}

@GetMapping(value = "/email/{email}")
public Optional<User> getUserByEmailController(@PathVariable(name = "email")
String email){
    return service.getUserByEmail(email);
}

@GetMapping(value =  "/voterId/{vid}")
public Optional<User> getUserByVoterId(@PathVariable Long vid) {
    return service.getUserByVoterId(vid);
}
    public class AdminController {

    @Autowired
    private AdminService service; // Add @Autowired

    @GetMapping("/{id}") // Fix missing path parameter
    public Optional<Admin> getUserByIdController(@PathVariable(name = "id") Long
id) {
       return service.getAdminById(id);
    }

    @PostMapping(value = "/auth")
    public boolean authAdminByIdAndPassword(@RequestBody Map<String, String>
logingRequest) {
     Long id = Long.valueOf(logingRequest.get("id").toString());
     String password = logingRequest.get("password").toString();

     System.out.println("From Controller: Voter ID = " + id + ", Password = " +
password);
     return service.authAdminByIdAndPassword(id, password);
    }
```

```
}
```

d) User Controller

```
package com.jspider.votingsurvey.controller;

import java.util.List;
import java.util.Map;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.HttpStatus;

import com.jspider.votingsurvey.entity.User;
import com.jspider.votingsurvey.services.UsersService;

@RestController
@RequestMapping(value = "/api/user")
@CrossOrigin(value = "http://localhost:5173")
public class UserController {

@Autowired
private UsersService service;

@PostMapping(value = "/register")
public ResponseEntity<?> saveUserController(@RequestBody User user) {
```

```
    if (user.getAge() < 18) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("You must be at least 18 years old to register.");
    }

    long vid = user.getVoterId();
    Optional<User> optional = service.getUserByVoterId(vid);

    if (optional.isPresent()) {
        return ResponseEntity.status(HttpStatus.CONFLICT).body("Voter ID already exists. Please use a different Voter ID.");
    }

    User savedUser = service.saveUser(user);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}



@GetMapping
public List<User> getAllUserController(){
    return service.getAllUsers();
}

@GetMapping("/constituency/{constituency}")
    public List<User> getUsersByConstituencyController(@PathVariable String constituency) {
    return service.getUsersByConstituency(constituency);
}

@DeleteMapping(value = "/{id}")
public boolean deleteUserById(@PathVariable int id) {
    return service.deleteUserById(id);
}

@PutMapping(value = "/{id}")
public Optional<User> updateUserById(@PathVariable int id, @RequestBody User user) {
    return service.updateUserById(id, user);
```

```
}

@PostMapping(value = "/login")
public boolean loginUserByVoterIdAndPassword(@RequestBody Map<String, Object> loginRequest) {
   Long voterId = Long.valueOf(loginRequest.get("voterId").toString());
   String password = loginRequest.get("password").toString();

   System.out.println("From Controller: Voter ID = " + voterId + ", Password = " + password);
   return service.loginUserByVoterIdAndPassword(voterId, password);
}



@PutMapping(value = "/constituency/{constituency}/vote-status/{hasVoted}")
   public List<User> updateVotingStatus(@PathVariable String constituency, @PathVariable boolean hasVoted) {
    return service.updateVotingStatus(constituency, hasVoted);
}
@Autowired
private UsersService service;

@PostMapping(value = "/register")
public ResponseEntity<?> saveUserController(@RequestBody User user) {
   if (user.getAge() < 18) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("You must be at least 18 years old to register.");
   }

   long vid = user.getVoterId();
   Optional<User> optional = service.getUserByVoterId(vid);

   if (optional.isPresent()) {
      return ResponseEntity.status(HttpStatus.CONFLICT).body("Voter ID already exists. Please use a different Voter ID.");
   }

   User savedUser = service.saveUser(user);
```

```
    return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
  }


  @GetMapping
  public List<User> getAllUserController(){
     return service.getAllUsers();
  }

  @GetMapping(value = "/{id}")
  public Optional<User> getUserByIdController(@PathVariable int id) {
     return service.getUserById(id);
  }

  @GetMapping(value = "/email/{email}")
  public Optional<User> getUserByEmailController(@PathVariable(name = "email")
  String email){
     return service.getUserByEmail(email);
  }
  public class AdminController {

     @Autowired
     private AdminService service; // Add @Autowired

     @GetMapping("/{id}") // Fix missing path parameter
     public Optional<Admin> getUserByIdController(@PathVariable(name = "id") Long
id) {
        return service.getAdminById(id);
     }

     @PostMapping(value = "/auth")
     public boolean authAdminByIdAndPassword(@RequestBody Map<String, String>
logingRequest) {
      Long id = Long.valueOf(logingRequest.get("id").toString());
      String password = logingRequest.get("password").toString();

      System.out.println("From Controller: Voter ID = " + id + ", Password = " +
password);
```

```
    return service.authAdminByIdAndPassword(id, password);
   }
  }
```

e) User Controller

```
package com.jspider.votingsurvey.controller;

import java.util.List;
import java.util.Map;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.HttpStatus;

import com.jspider.votingsurvey.entity.User;
import com.jspider.votingsurvey.services.UsersService;

@RestController
@RequestMapping(value = "/api/user")
@CrossOrigin(value = "http://localhost:5173")
public class UserController {

@Autowired
private UsersService service;
```

```
@PostMapping(value = "/register")
public ResponseEntity<?> saveUserController(@RequestBody User user) {
   if (user.getAge() < 18) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("You must be at least 18 years old to register.");
   }

   long vid = user.getVoterId();
   Optional<User> optional = service.getUserByVoterId(vid);

   if (optional.isPresent()) {
      return ResponseEntity.status(HttpStatus.CONFLICT).body("Voter ID already exists. Please use a different Voter ID.");
   }

   User savedUser = service.saveUser(user);
   return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}


@GetMapping
public List<User> getAllUserController(){
    return service.getAllUsers();
}


@GetMapping(value =  "/voterId/{vid}")
public Optional<User> getUserByVoterId(@PathVariable Long vid) {
    return service.getUserByVoterId(vid);
}


@PutMapping(value = "/voterId/{vid}/vote-status/{hasVoted}")
public User updateVotingStatusByUserVoterIdController(@PathVariable Long vid, @PathVariable boolean hasVoted) {
    return service.updateVotingStatusByUserVoterId(vid, hasVoted);
}
```

```
//   @PutMapping("/reset-votes/{constituencyNumber}")
//                public ResponseEntity<String> resetVotes(@PathVariable Long constituencyNumber) {
//        boolean isUpdated = service.resetVotesByConstituency(constituencyNumber);
//        if (isUpdated) {
//                return ResponseEntity.ok("Votes reset successfully for constituency: " + constituencyNumber);
//        } else {
//           return ResponseEntity.badRequest().body("No users found with hasVoted=true in constituency: " + constituencyNumber);
//        }
//    }

@PutMapping("/reset-votes/{constituencyNumber}")
public boolean resetVotesByConstituency(@PathVariable Long constituencyNumber) {
    return service.resetVotesByConstituency(constituencyNumber);
}
public ResponseEntity<?> saveUserController(@RequestBody User user) {
   if (user.getAge() < 18) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("You must be at least 18 years old to register.");
   }

   long vid = user.getVoterId();
   Optional<User> optional = service.getUserByVoterId(vid);

   if (optional.isPresent()) {
      return ResponseEntity.status(HttpStatus.CONFLICT).body("Voter ID already exists. Please use a different Voter ID.");
   }

   User savedUser = service.saveUser(user);
   return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}


@GetMapping
```

```
public List<User> getAllUserController(){
    return service.getAllUsers();
}


@GetMapping(value =  "/voterId/{vid}")
public Optional<User> getUserByVoterId(@PathVariable Long vid) {
    return service.getUserByVoterId(vid);
}


@PutMapping(value = "/voterId/{vid}/vote-status/{hasVoted}")
public User updateVotingStatusByUserVoterIdController(@PathVariable Long vid, @PathVariable boolean hasVoted) {
    return service.updateVotingStatusByUserVoterId(vid, hasVoted);
}

//   @PutMapping("/reset-votes/{constituencyNumber}")
//                public ResponseEntity<String> resetVotes(@PathVariable Long constituencyNumber) {
//        boolean isUpdated = service.resetVotesByConstituency(constituencyNumber);
//        if (isUpdated) {
//                return ResponseEntity.ok("Votes reset successfully for constituency: " + constituencyNumber);
//        } else {
//           return ResponseEntity.badRequest().body("No users found with hasVoted=true in constituency: " + constituencyNumber);
//        }
//    }

@PutMapping("/reset-votes/{constituencyNumber}")
public boolean resetVotesByConstituency(@PathVariable Long constituencyNumber) {
    return service.resetVotesByConstituency(constituencyNumber);
}
}
```

f) Constituency Controller

```
package com.jspider.votingsurvey.controller;

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.ResponseEntity;

import com.jspider.votingsurvey.entity.Constituency;
import com.jspider.votingsurvey.services.ConstituencysService;

@RestController
@RequestMapping(value = "/api/constituency")
@CrossOrigin(value = "http://localhost:5173")
public class ConstituencyController {

@Autowired
private ConstituencysService service;

@PostMapping
public Constituency saveConstituency(@RequestBody Constituency constituency) {
    Optional<Constituency> optional = service.getConstituencyById(constituency.getId());
    if (optional.isPresent()) {
```

```
            return null;
    }else
            return service.saveConstituency(constituency);
}

@PostMapping("/all")
public List<Constituency> saveMultipleConstituencyController(@RequestBody List<Constituency> constituencyList) {
   List<Constituency> savedConstituencies = new ArrayList<>();

   for (Constituency constituency : constituencyList) {
      if (constituency.getId() != null && service.getConstituencyById(constituency.getId()).isPresent()) {
         continue; // Skip if ID is already present
      }
      savedConstituencies.add(service.saveConstituency(constituency));
   }

   return savedConstituencies;
}

Optional<Constituency> getConstituencyByName(@PathVariable(name = "name") String name){
    return service.getConstituencyByName(name);
}
    public ResponseEntity<?> saveUserController(@RequestBody User user) {
   if (user.getAge() < 18) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("You must be at least 18 years old to register.");
   }

   long vid = user.getVoterId();
   Optional<User> optional = service.getUserByVoterId(vid);

   if (optional.isPresent()) {
      return ResponseEntity.status(HttpStatus.CONFLICT).body("Voter ID already exists. Please use a different Voter ID.");
   }
```

```
    User savedUser = service.saveUser(user);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}


@GetMapping
public List<User> getAllUserController(){
    return service.getAllUsers();
}


@GetMapping(value =  "/voterId/{vid}")
public Optional<User> getUserByVoterId(@PathVariable Long vid) {
    return service.getUserByVoterId(vid);
}


@PutMapping(value = "/voterId/{vid}/vote-status/{hasVoted}")
public User updateVotingStatusByUserVoterIdController(@PathVariable Long vid,
@PathVariable boolean hasVoted) {
    return service.updateVotingStatusByUserVoterId(vid, hasVoted);
}

//  @PutMapping("/reset-votes/{constituencyNumber}")
//                  public ResponseEntity<String> resetVotes(@PathVariable Long
constituencyNumber) {
//      boolean isUpdated = service.resetVotesByConstituency(constituencyNumber);
//      if (isUpdated) {
//                  return ResponseEntity.ok("Votes reset successfully for constituency: " +
constituencyNumber);
//      } else {
//          return ResponseEntity.badRequest().body("No users found with hasVoted=true
in constituency: " + constituencyNumber);
//      }
//  }

@PutMapping("/reset-votes/{constituencyNumber}")
```

```
public boolean resetVotesByConstituency(@PathVariable Long constituencyNumber) {
    return service.resetVotesByConstituency(constituencyNumber);
}
@GetMapping("/active")
   public List<Constituency> getActiveConstituenciesController() {
    return service.getActiveConstituencies();
}

@GetMapping("/state/{state}")
   public List<Constituency> getConstituenciesByStateController(@PathVariable
String state) {
    return service.getConstituenciesByState(state);
}

@GetMapping("/allConstituencyByIdOrName")
public ResponseEntity<?> getConstituenciesByNameOrIdController(
       @RequestParam(required = false) Long id,
       @RequestParam(required = false) String name) {

   if (id == null && name == null) {
      return ResponseEntity.badRequest().body(Collections.singletonMap("error",
"Either 'id' or 'name' is required"));
   }

   List<Constituency> constituencies = service.getConstituencyByIdOrName(id,
name);
   return ResponseEntity.ok(constituencies);
}



@PutMapping("/{id}/election-status/{electionActive}")
public Constituency updateElectionStatusController(@PathVariable Long id,
@PathVariable boolean electionActive) {
    return service.updateElectionStatus(id, electionActive);
}
Optional<Constituency> getConstituencyByName(@PathVariable(name = "name")
String name){
    return service.getConstituencyByName(name);
```

```
}

@GetMapping("/active")
   public List<Constituency> getActiveConstituenciesController() {
    return service.getActiveConstituencies();
}

@GetMapping("/state/{state}")
   public List<Constituency> getConstituenciesByStateController(@PathVariable
String state) {
    return service.getConstituenciesByState(state);
}

@GetMapping("/allConstituencyByIdOrName")
public ResponseEntity<?> getConstituenciesByNameOrIdController(
       @RequestParam(required = false) Long id,
       @RequestParam(required = false) String name) {

   if (id == null && name == null) {
      return ResponseEntity.badRequest().body(Collections.singletonMap("error",
"Either 'id' or 'name' is required"));
   }

   List<Constituency> constituencies = service.getConstituencyByIdOrName(id,
name);
   return ResponseEntity.ok(constituencies);
}



@PutMapping("/{id}/election-status/{electionActive}")
public Constituency updateElectionStatusController(@PathVariable Long id,
@PathVariable boolean electionActive) {
    return service.updateElectionStatus(id, electionActive);
}

@GetMapping
List<Constituency> getAllConstituencyController(){
    return service.getAllConstituency();
```

```
}

@GetMapping(value = "/{id}")
Optional<Constituency> getConstituencyByIdController(@PathVariable(name = "id")
Long id){
    return service.getConstituencyById(id);
}

@GetMapping(value = "/name/{name}")
Optional<Constituency>  getConstituencyByName(@PathVariable(name  =  "name")
String name){
    return service.getConstituencyByName(name);
}

@GetMapping("/active")
   public List<Constituency> getActiveConstituenciesController() {
    return service.getActiveConstituencies();
}

@GetMapping("/state/{state}")
   public  List<Constituency>  getConstituenciesByStateController(@PathVariable
String state) {
    return service.getConstituenciesByState(state);
}

@GetMapping("/allConstituencyByIdOrName")
public ResponseEntity<?> getConstituenciesByNameOrIdController(
       @RequestParam(required = false) Long id,
       @RequestParam(required = false) String name) {

   if (id == null && name == null) {
      return          ResponseEntity.badRequest().body(Collections.singletonMap("error",
"Either 'id' or 'name' is required"));
   }

   List<Constituency>  constituencies  =  service.getConstituencyByIdOrName(id,
name);
   return ResponseEntity.ok(constituencies);
```

```
}


@PutMapping("/{id}/election-status/{electionActive}")
public Constituency updateElectionStatusController(@PathVariable Long id,
@PathVariable boolean electionActive) {
    return service.updateElectionStatus(id, electionActive);
}
}
```

g) Party Controller

```
package com.jspider.votingsurvey.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.ResponseEntity;

import com.jspider.votingsurvey.entity.Party;
import com.jspider.votingsurvey.services.PartysService;

@RestController
@RequestMapping(value = "/api/party")
@CrossOrigin(value = "http://localhost:5173")
public class PartyController {
```

```
@Autowired
private PartysService service;

@PostMapping
public Party savePartyController(@RequestBody Party party) {
   return service.saveParty(party);
}

@GetMapping
public List<Party> getAllPartyscontroller() {
    return service.getAllPartys();
}

@GetMapping(value = "/constituency-number/{constituencyId}")
public List<Party> getPartiesByConstituency(@PathVariable Long constituencyId){
    return service.getPartiesByConstituency(constituencyId);
}

@GetMapping(value = "/activeConstituenciePartys")
public List<Party> getAllPartysByActiveConstituenciesNumberController() {
    return service.getAllPartysByActiveConstituenciesNumber();
}


@GetMapping(value = "/allActiveElectionParties")
public List<Party> getActiveElectionPartiesController(){
    return service.getActiveElectionParties();
}

@DeleteMapping(value = "/delete/{Id}")
boolean deletePartyByIdDao(@PathVariable Long Id) {
    return service.deletePartyByIdDao(Id);
}


@PutMapping("/{partyId}/votes")
```

```
  public ResponseEntity<String> updatePartyVotes(@PathVariable Long partyId,
@RequestParam Long votes) {
      boolean isUpdated = service.updateVotes(partyId, votes);
      if (isUpdated) {
         return ResponseEntity.ok("Votes updated successfully.");
      } else {
         return ResponseEntity.badRequest().body("Failed to update votes.");
      }
}

@GetMapping("/byConstituencyIdOrName")
  public                                                    List<Party>
getPartiesByConstituencyIdOrNameController(@RequestParam(required = false) Long
constituencyId,
                        @RequestParam(required = false) String constituencyName) {
     return                service.getPartiesByConstituencyIdOrName(constituencyId,
constituencyName);
}

@PutMapping("/resetVotes")
  public ResponseEntity<String> resetVotesByConstituencyId(@RequestParam Long
constituencyId) {
     String                              responseMessage                    =
service.resetAllPartyVotesByConstituencyId(constituencyId);
     return ResponseEntity.ok(responseMessage);
  }
@Autowired
private PartysService service;

@PostMapping
public Party savePartyController(@RequestBody Party party) {
  return service.saveParty(party);
}

@GetMapping
public List<Party> getAllPartyscontroller() {
   return service.getAllPartys();
}
```

```
@GetMapping(value = "/constituency-number/{constituencyId}")
public List<Party> getPartiesByConstituency(@PathVariable Long constituencyId){
    return service.getPartiesByConstituency(constituencyId);
}

@GetMapping(value = "/activeConstituenciePartys")
public List<Party> getAllPartysByActiveConstituenciesNumberController() {
    return service.getAllPartysByActiveConstituenciesNumber();
}


@GetMapping(value = "/allActiveElectionParties")
public List<Party> getActiveElectionPartiesController(){
    return service.getActiveElectionParties();
}

@DeleteMapping(value = "/delete/{Id}")
boolean deletePartyByIdDao(@PathVariable Long Id) {
    return service.deletePartyByIdDao(Id);
}


@PutMapping("/{partyId}/votes")
  public ResponseEntity<String> updatePartyVotes(@PathVariable Long partyId,
@RequestParam Long votes) {
      boolean isUpdated = service.updateVotes(partyId, votes);
      if (isUpdated) {
         return ResponseEntity.ok("Votes updated successfully.");
      } else {
         return ResponseEntity.badRequest().body("Failed to update votes.");
      }
}

@GetMapping("/byConstituencyIdOrName")
```

```
  public List<Party> getPartiesByConstituencyIdOrNameController(@RequestParam(required = false) Long constituencyId,
                        @RequestParam(required = false) String constituencyName) {
      return service.getPartiesByConstituencyIdOrName(constituencyId, constituencyName);
  }

@PutMapping("/resetVotes")
  public ResponseEntity<String> resetVotesByConstituencyId(@RequestParam Long constituencyId) {
      String responseMessage = service.resetAllPartyVotesByConstituencyId(constituencyId);
      return ResponseEntity.ok(responseMessage);
  }


}
```

5.2 Services

a) AdminAuth Service

```
package com.jspider.votingsurvey.services;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.jspider.votingsurvey.dao.dashboard.AdminDao;
import com.jspider.votingsurvey.entity.Admin;

@Service
public class AdminAuthService implements AdminService {

@Autowired
private AdminDao dao;

@Override
public Optional<Admin> getAdminById(Long id) {
    return dao.getAdminByIdDao(id);
}

@Override
public boolean authAdminByIdAndPassword(Long id, String password) {

    Optional<Admin> optional = getAdminById(id);
    if (!(optional.isPresent())) return false;

    Admin admin = optional.get();
    if (id.equals(admin.getId()) && password.equals(admin.getPassword())) return true;

    return false;
}
```

```
}
```

b) Admin Service

```
package com.jspider.votingsurvey.services;

import java.util.Optional;

import com.jspider.votingsurvey.entity.Admin;

public interface AdminService {

/**
    * Fetches a admin by their unique ID.
    *
    * @param id The ID of the admin.
    * @return An Optional containing the admin if found, otherwise empty.
    */
Optional<Admin> getAdminById(Long id);

// Authorization User
public boolean authAdminByIdAndPassword(Long id, String password);
}
```

c) Constituencies Service

```
package com.jspider.votingsurvey.services;

import java.time.LocalDate;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.jspider.votingsurvey.dao.ConstituencysDao;
```

```
import com.jspider.votingsurvey.entity.Constituency;

@Service
public class ConstituencyService implements ConstituencysService {

@Autowired
private ConstituencysDao dao;

@Override
public Constituency saveConstituency(Constituency constituency) {

    if (constituency.getDOLS() == null) {
        constituency.setDOLS(LocalDate.now());
    }

    return dao.saveConstituencyDao(constituency);
}

@Override
public List<Constituency> saveMultipleConstituency(List<Constituency>
constituencyList) {
    return dao.saveMultipleConstituencyDao(constituencyList);
}

@Override
public List<Constituency> getAllConstituency() {
    return dao.getAllConstituencyDao();
}

@Override
public Optional<Constituency> getConstituencyById(Long id) {
    return dao.getConstituencyByIdDao(id);
}

@Override
public Optional<Constituency> getConstituencyByName(String name) {
    return dao.getConstituencyByNameDao(name);
```

```
}

@Override
public List<Constituency> getActiveConstituencies() {
    return dao.getActiveConstituenciesDao();
}

@Override
public List<Constituency> getConstituenciesByState(String state) {
    return dao.getConstituenciesByStateDao(state);
}

@Override
public Constituency updateElectionStatus(Long id, boolean electionActive) {
    Optional<Constituency> optional = getConstituencyById(id);
    if (optional.isPresent()) {
            Constituency constituency = optional.get();
            constituency.setElectionActive(electionActive);
            if (electionActive) constituency.setDOLS(LocalDate.now());
            return dao.saveConstituencyDao(constituency);
    }
    return null;
}

@Override
public List<Constituency> getConstituencyByIdOrName(Long id, String name) {
    return dao.findConstituencyByIdOrNameDao(id, name);
}
}
```

d) Constituencies Service

```
package com.jspider.votingsurvey.services;

import java.util.List;
import java.util.Optional;
```

```
import com.jspider.votingsurvey.entity.Constituency;

public interface ConstituencysService {
/**
    * Saves a constituency entity to the database.
    *
    * @param constituency the constituency entity to be saved
    * @return the saved constituency entity
    */
   Constituency saveConstituency(Constituency constituency);

   /**
    * Saves a multiple constituency entity to the database.
    *
    * @param constituency the constituency entity to be saved
    * @return the saved constituency list entity
    */
   List<Constituency>                saveMultipleConstituency(List<Constituency>
constituencyList);


   /**
    * Retrieves all constituency from the database.
    *
    * @return a list of all constituency
    */
   List<Constituency> getAllConstituency();

   /**
    * Retrieves a constituency by their unique identifier.
    *
    * @param id the ID of the constituency
    * @return an optional containing the constituency if found, otherwise empty
    */
   Optional<Constituency> getConstituencyById(Long id);

   /**
```

```
    * Retrieves a constituency by name.
    *
    * @param name the name of the constituency
    * @return an optional containing the constituency if found, otherwise empty
    */
   Optional<Constituency> getConstituencyByName(String name);

   List<Constituency> getActiveConstituencies();

   List<Constituency> getConstituenciesByState(String state);

 // Update election status for a specific constituency
   Constituency updateElectionStatus(Long id, boolean electionActive);

   List<Constituency> getConstituencyByIdOrName(Long id, String name);



}
```

e) User Service

```
package com.jspider.votingsurvey.services;

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.jspider.votingsurvey.dao.UsersDao;
import com.jspider.votingsurvey.entity.User;

@Service
public class UserService implements UsersService {

@Autowired
```

```
private UsersDao dao;

@Override
public User saveUser(User user) {
    return dao.saveUserDao(user);
}

@Override
public List<User> getAllUsers() {
    return dao.getAllUsers();
}

@Override
public Optional<User> getUserById(int id) {
    return dao.getUserById(id);
}

@Override
public Optional<User> updateUserById(int id, User user) {
    return dao.updateUserById(id, user);
}

@Override
public boolean deleteUserById(int id) {
    return dao.deleteUserById(id);
}

@Override
public Optional<User> getUserByVoterId(Long vId) {
    return dao.getUserByVoterId(vId);
}

@Override
   public Optional<User> getUserByEmail(String email) {
       return dao.getUserByEmail(email);
   }
```

```
// Authorization User
@Override                                                                    // voterId
and password from client side
public boolean loginUserByVoterIdAndPassword(Long voterId, String password) {
    Optional<User> optional = getUserByVoterId(voterId);
    if (!(optional.isPresent())) {
            return false;
    }
    User user = optional.get();

    Long dbVoterId = user.getVoterId(); // From DB
    String dbPassword = user.getPassword(); // From DB

    if (voterId.equals(dbVoterId) && password.equals(dbPassword)) {
            return true;
    }
    return false;

}

@Override
public List<User> getUsersByConstituency(String constituency) {
    return dao.getUsersByConstituencyDao(constituency);
}

@Override
public List<User> updateVotingStatus(String constituency, boolean hasVoted) {
    List<User> userList = dao.getUsersByConstituencyDao(constituency);
    if (!userList.isEmpty()) {
            for (User user : userList) {
                    user.setHasVoted(hasVoted);
            }
            dao.saveAllUserDao(userList);

    }
    return userList;
```

```
}

@Override
public User updateVotingStatusByUserVoterId(Long vId, boolean hasVoted) {
    Optional<User> optional = getUserByVoterId(vId);
    if (!(optional.isPresent())) return null;

    User user = optional.get();
    user.setHasVoted(hasVoted);
    dao.saveUserDao(user);
    return user;
}
@Override
public User saveUser(User user) {
    return dao.saveUserDao(user);
}

@Override
public List<User> getAllUsers() {
    return dao.getAllUsers();
}

@Override
public Optional<User> getUserById(int id) {
    return dao.getUserById(id);
}

@Override
public Optional<User> updateUserById(int id, User user) {
    return dao.updateUserById(id, user);
}

@Override
public boolean deleteUserById(int id) {
    return dao.deleteUserById(id);
}
```

```
@Override
public Optional<User> getUserByVoterId(Long vId) {
    return dao.getUserByVoterId(vId);
}

@Override
   public Optional<User> getUserByEmail(String email) {
      return dao.getUserByEmail(email);
   }


// Authorization User
@Override                                                                    // voterId
and password from client side
public boolean loginUserByVoterIdAndPassword(Long voterId, String password) {
    Optional<User> optional = getUserByVoterId(voterId);
    if (!(optional.isPresent())) {
            return false;
    }
    User user = optional.get();

    Long dbVoterId = user.getVoterId(); // From DB
    String dbPassword = user.getPassword(); // From DB

    if (voterId.equals(dbVoterId) && password.equals(dbPassword)) {
            return true;
    }
    return false;

}

@Override
public List<User> getUsersByConstituency(String constituency) {
    return dao.getUsersByConstituencyDao(constituency);
}

@Override
```

```
public List<User> updateVotingStatus(String constituency, boolean hasVoted) {
    List<User> userList = dao.getUsersByConstituencyDao(constituency);
    if (!userList.isEmpty()) {
            for (User user : userList) {
                    user.setHasVoted(hasVoted);
            }
            dao.saveAllUserDao(userList);

    }


@Override
public boolean resetVotesByConstituency(Long constituencyNumber) {
    return dao.resetVotesByConstituencyDao(constituencyNumber);
}
}
```

f) User Service

```
package com.jspider.votingsurvey.services;

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

import com.jspider.votingsurvey.entity.User;

public interface UsersService {

/**
    * Saves a new user in the database.
    *
    * @param user The user to be saved.
    * @return The saved user with a generated ID.
    */
User saveUser(User user);

/**
```

```
    * Retrieves all users from the database.
    *
    * @return A list of all users.
    */
List<User> getAllUsers();

/**
    * Fetches a user by their unique ID.
    *
    * @param id The ID of the user.
    * @return An Optional containing the user if found, otherwise empty.
    */
Optional<User> getUserById(int id);

/**
    * Updates a user by their ID with new details.
    *
    * @param id The ID of the user to update.
    * @param user The updated user details.
    * @return An Optional containing the updated user if the update is successful.
    */
Optional<User> updateUserById(int id, User user);

/**
    * Deletes a user by their ID.
    *
    * @param id The ID of the user to delete.
    * @return true if the deletion was successful, false otherwise.
    */
boolean deleteUserById(int id);

/**
    * Retrieves a user by their Voter ID.
    *
    * @param vId The unique voter ID of the user.
    * @return An Optional containing the user if found, otherwise empty.
    */
```

```
public Optional<User> getUserByVoterId(Long vId);

/**
    * Retrieves a user by their email.
    *
    * @param email The email of the user.
    * @return An Optional containing the user if found, otherwise empty.
    */
   Optional<User> getUserByEmail(String email);

   public List<User> getUsersByConstituency(String constituency);

 // Authorization User
    public boolean loginUserByVoterIdAndPassword(Long voterId, String password);

    public List<User> updateVotingStatus(String constituency, boolean hasVoted);

    public User updateVotingStatusByUserVoterId(Long vId, boolean hasVoted);

    boolean resetVotesByConstituency(Long constituencyNumber);


}
```

g) Party Service-1

```
package com.jspider.votingsurvey.services;

import java.util.List;

import com.jspider.votingsurvey.entity.Party;

public interface PartysService {

/**
```

```
    * Saves a part entity to the database.
    *
    * @param part the part entity to be saved
    * @return the saved part entity
    */
Party saveParty(Party party);

/**
    * Retrieves all party from the database.
    *
    * @return a list of all party
    */
   List<Party> getAllPartys();

   boolean deletePartyByIdDao(Long Id);

   List<Party> getAllPartysByActiveConstituenciesNumber();

   List<Party> getActiveElectionParties();

   List<Party> getPartiesByConstituency(Long constituencyId);

   List<Party> getPartiesByConstituencyIdOrName(Long constituencyId, String
constituencyName);

   boolean updateVotes(Long partyId, Long newVotes);

   String resetAllPartyVotesByConstituencyId(Long constituencyId);


}
```

h) Party Service-whole

```
package com.jspider.votingsurvey.services;

import java.util.List;
```

```
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.jspider.votingsurvey.dao.PartysDao;
import com.jspider.votingsurvey.entity.Constituency;
import com.jspider.votingsurvey.entity.Party;

@Service
public class PartyService implements PartysService {

@Autowired
private PartysDao dao;

@Autowired
   private ConstituencysService constituencysService;

@Override
public Party saveParty(Party party) {
    if (party.getConstituency() != null && party.getConstituency().getId() != 0) {
        Optional<Constituency> constituencyOpt = constituencysService.getConstituencyById(party.getConstituency().getId());

        if (constituencyOpt.isPresent()) {
           party.setConstituency(constituencyOpt.get());
        } else {
           throw new RuntimeException("Constituency not found with ID: " + party.getConstituency().getId());
        }
      }
//       return dao.savePartyDao(party);
    Party savedParty = dao.savePartyDao(party);
    if (!savedParty.getConstituency().isElectionActive()) {
       constituencysService.updateElectionStatus(savedParty.getConstituency().getId(), true);
    }
```

```
    return savedParty;

}
   List<Party> getAllPartys();

   boolean deletePartyByIdDao(Long Id);

   List<Party> getAllPartysByActiveConstituenciesNumber();

   List<Party> getActiveElectionParties();

   List<Party> getPartiesByConstituency(Long constituencyId);

   List<Party> getPartiesByConstituencyIdOrName(Long constituencyId, String
constituencyName);

   boolean updateVotes(Long partyId, Long newVotes);

@Override
public List<Party> getAllPartys() {
    return dao.getAllPartysDao();
}

@Override
public List<Party> getPartiesByConstituency(Long constituencyId) {
    return dao.getPartiesByConstituencyDao(constituencyId);
}

@Override
public boolean deletePartyByIdDao(Long Id) {
    return dao.deletePartyByIdDao(Id);
}

//   @Override
//   public List<Party> getAllPartysByActiveConstituenciesNumber() {
//      List<Constituency>                activeConstituencies                =
constituencysService.getActiveConstituencies();
```

```
//      if (activeConstituencies.isEmpty()) return List.of();
//      Long constituencyId = activeConstituencies.get(0).getId();
//      return getPartiesByConstituency(constituencyId);
//  }

@Override
public List<Party> getAllPartysByActiveConstituenciesNumber() {
   List<Constituency> activeConstituencies =
constituencysService.getActiveConstituencies();

   if (activeConstituencies.isEmpty()) return List.of();

   // Get the most recent active constituency (highest ID)
   Long recentConstituencyId = activeConstituencies.stream()
         .map(Constituency::getId)
         .max(Long::compare)
         .orElse(null);

   return recentConstituencyId != null ?
getPartiesByConstituency(recentConstituencyId) : List.of();
}




@Override
public List<Party> getActiveElectionParties() {
    return dao.getActiveElectionPartiesDao();
}

@Override
public boolean updateVotes(Long partyId, Long newVotes) {
     return dao.updateVotesDao(partyId, newVotes);
}

@Override
```

```
public List<Party> getPartiesByConstituencyIdOrName(Long constituencyId, String constituencyName) {

    if (constituencyId != null) {
        return dao.getPartiesByConstituencyDao(constituencyId);
      } else if (constituencyName != null && !constituencyName.isEmpty()) {
        return dao.getPartiesByConstituencyNameDao(constituencyName);
      }
      return List.of(); // Return empty list if both are null
}

@Override
public String resetAllPartyVotesByConstituencyId(Long constituencyId) {
    int updateStatus = dao.resetAllPartyVotesByConstituencyIdDao(constituencyId);
    return updateStatus >0 ?  "All votes reset successfully." : "Error: No parties found in this constituency.";

}


}
```

CHAPTER - 6

TESTING

TESTING

The testing phase of the Voting Survey Application is crucial to ensure the reliability, security, and functionality of the system. It begins with **unit testing**, where individual components such as user login, survey creation, and vote submission are tested in isolation to verify they work as expected. Next, **integration testing** is conducted to ensure that these components interact correctly — for example, confirming that user-submitted votes are properly stored and reflected in the results. **System testing** follows, where the entire application is tested end-to-end in a real-world scenario to identify any functional or performance issues. **Usability testing** is also performed to evaluate the user interface and overall user experience, ensuring that the application is intuitive and accessible. Additionally, **security testing** checks for vulnerabilities such as unauthorized access or data manipulation. Finally, **acceptance testing** is carried out to confirm that the application meets all functional requirements and is ready for deployment. Thorough documentation and bug tracking are maintained throughout the process to ensure that issues are resolved before the application goes live.

6.1 Methodology of Testing used in the Project

Testing is a crucial phase to ensure that the Voter Survey Application is reliable, user-friendly, and performs well under different conditions. Multiple levels of testing were conducted:

Unit Testing

- Test individual events like message broadcasting and user connection notifications.
- Tools: Jest (for JavaScript/React apps) or JUnit (for Java backend).

Integration Testing

- Test real-time message flow between multiple users.
- Example: After submitting a survey, it was verified that the response is saved correctly in the database and results are updated in real-time.

Performance Testing

- Simulate multiple users to test the application scalability.
- Observed that the system maintained performance without significant delays.

Functional Testing

Verified all features:

- User registration and login
- Survey participation
- Admin survey creation and management

- Result visualization

Security Testing

- Tested for vulnerabilities like:
  a) Unauthorized access
  b) SQL Injection / NoSQL Injection
  c) Cross-site scripting (XSS)
- Implemented input validation and authentication token checks.

6.2 Testing Scenario

Test Case	Input	Expected Output	Status
Open an application	"Open Notepad"	Notepad launches	Passed
Check system time	"Say time?"	System announces the correct time.	Passed
Perform a Google search	"Search Python tutorials"	Browser opens with search results	Passed
Send a whatsapp (optional feature)	"Send a whatsapp message"	Whatsapp message sent successfully	Passed
Offline command test	Disconnect Internet, say "Open Calculator"	Calculator opens without errors	Passed
AI search	"who won world cup"	System answers successfully.	Passed

Testing plays a vital role in the software development life cycle of the Voter Survey Application. It ensures that the system behaves as expected, meets the user requirements, and performs reliably under different conditions. The goal of testing in this project was not just to identify and fix bugs, but also to validate the overall functionality, security, usability, and compatibility of the application.

The testing process began with unit testing, where individual components such as login validation, survey form submissions, and data-fetching functions were tested in isolation. For frontend components, unit tests were written to ensure that UI elements rendered properly and handled inputs as expected. On the backend, key functions like user authentication, data storage, and API responses were tested using tools such as Jest or JUnit, depending on the tech stack.

Following unit testing, integration testing was conducted to verify that various modules worked together seamlessly. This included verifying end-to-end scenarios like user signup → survey participation → data storage → result visualization. This phase was crucial in identifying interface-related issues between the frontend, backend, and database systems.

Functional testing was carried out to ensure that the application's core features met their functional requirements. Scenarios like creating a survey, responding to a survey, viewing results, and accessing the admin panel were manually tested. Test cases were created to simulate real-world usage, and the outcomes were compared against expected results.

In addition, performance testing was done to check how the application behaves under load. Multiple users were simulated to submit survey responses simultaneously, and the system’s response time, server load, and data handling were monitored. The results confirmed that the application maintained consistent performance without any crashes or significant slowdowns.

CHAPTER - 7

IMPLEMENTATION

IMPLEMENTATION

The implementation phase of the Voting Survey Application involves converting the planned design and functionalities into a working software product. This begins with setting up the frontend interface using web technologies such as HTML, CSS, and JavaScript (or frameworks like React) to ensure a responsive and user-friendly experience for both users and administrators. Simultaneously, the backend development is carried out using technologies such as Java, Node.js, or Python to handle server-side logic, including user authentication, survey management, and vote processing.

A database system like MySQL, MongoDB, or Firebase is integrated to securely store user data, survey questions, and responses. API endpoints are developed to allow smooth communication between frontend and backend components. Once the core modules—user registration, survey participation, admin control panel, and results display—are implemented, they are integrated and tested together. Version control tools like Git are used throughout to manage code updates. After successful internal testing and debugging, the application is deployed to a hosting platform for real-world use, marking the transition from development to a functional and accessible product.

Implementation Phase – Key Points

Technology Stack Selection

- Frontend: HTML, CSS, JavaScript, or frameworks like React/Angular
- Backend: Java, Python (Flask/Django), or Node.js
- Database: MySQL, PostgreSQL, MongoDB, or Firebase

Frontend Development

- Responsive user interface for user and admin panels
- Navigation between login, dashboard, survey participation, and result views

Backend Development

- API creation for handling data transfer between frontend and backend
- Logic for authentication, vote submission, and result processing

Database Integration

- Design and implement schemas for users, surveys, options, and responses
- Ensure secure and efficient data storage and retrieval

Security Implementation

- Password hashing and secure login systems
- Role-based access control (user vs admin)
- Input validation to prevent SQL injection and XSS

Admin Panel

- Create, edit, delete surveys
- View participation statistics and manage visibility

Voting Logic

- One-user-one-vote implementation
- Time-based survey activation/deactivation

Real-Time Features (if needed)

- Use of Web Sockets or polling for live result updates

Deployment

- Hosting using platforms like Firebase, Heroku, Vercel, or AWS
- Set up production environment and database connections

Version Control and CI/CD

- Use Git and GitHub for version tracking
- Optional integration of CI/CD for automated testing and deployment

Error Handling and Logging

- Build mechanisms for catching and displaying errors
- Log events for debugging and monitoring purposes

Post-Implementation Testing

- Conduct system, integration, and user acceptance testing before release

To-Do List Management:

This feature allows users to create and manage their to-do lists directly through the virtual assistant. It can store tasks, set deadlines, and notify the user about upcoming tasks.

- **Purpose**: To help users organize their daily tasks and keep track of important activities.
- **Implementation**:
    - The assistant will listen for commands like “Add task to my to-do list” or “What’s on my to-do list today?”
    - It will store tasks in a local database or a simple text file, where each task will be associated with a due date.
    - The assistant can remind the user of pending tasks or upcoming deadlines.

Once all modules are fully developed, tested, and integrated, the assistant is packaged for deployment. This typically involves creating an executable file for Windows, Linux, or macOS users, making it easy to distribute and install. The system is then ready for end-users to install on their computers and begin using.

CHAPTER - 8

SNAPSHOTS

SNAPSHOTS

The following section contains visual representations of various functionalities and workflows implemented in the system. These snapshots provide a clearer understanding of how the system operates, from user interaction to backend processes.

8.1 Home Page Voting

Fig. 8.1 Home Page Voting

8.2 Voter Login

The Voter Login Page is the entry point for registered users (voters) to access the voting system. It ensures secure authentication before allowing access to surveys or polls. The page is designed with a clean, user-friendly interface that is responsive across devices.

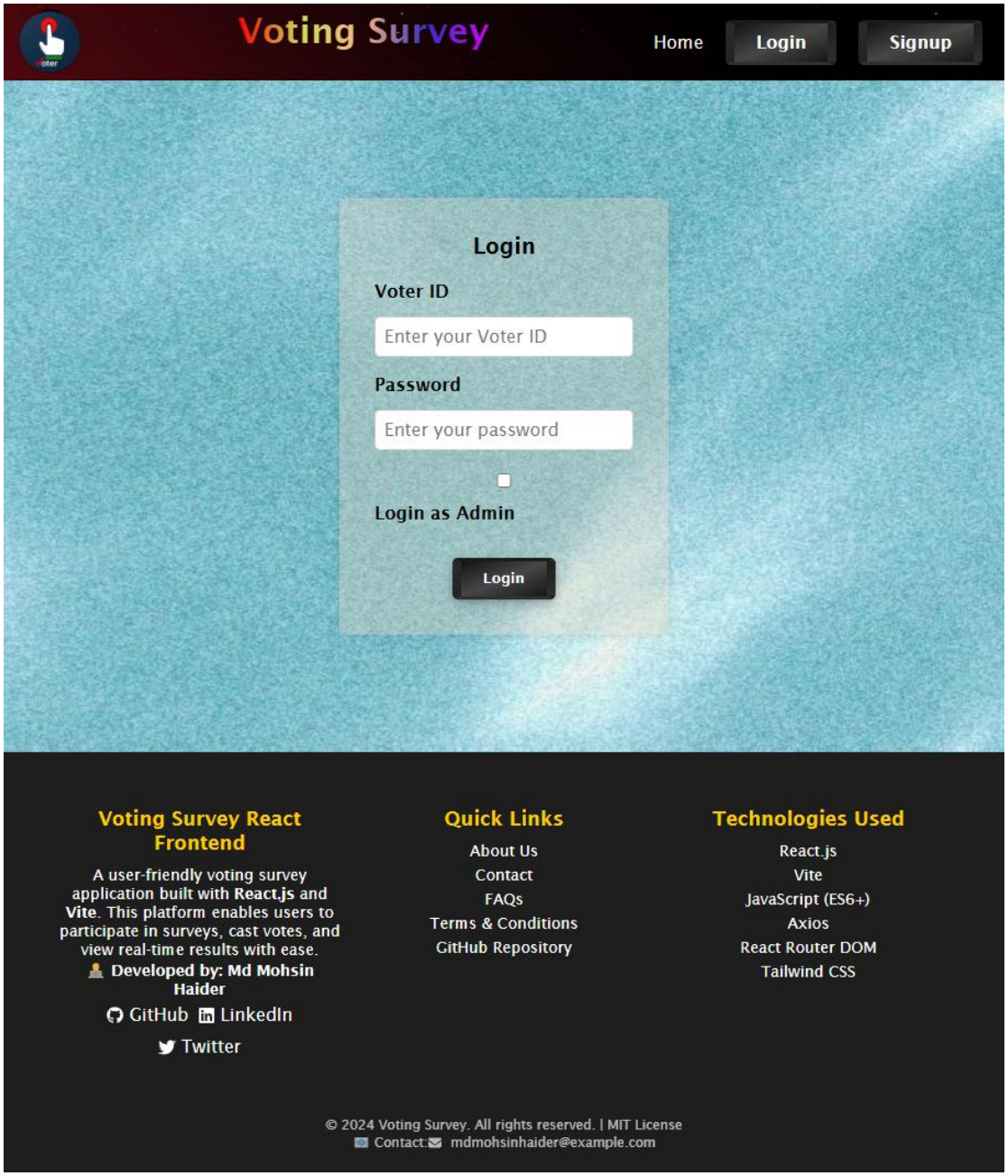

Fig. 8.2 Voter login

8.3 Voter Register

The Voter Registration Page allows new users to create an account to participate in surveys or polls. It collects necessary user details and ensures that each voter is uniquely registered and verified before being granted access to vote. The design is clean, intuitive, and responsive.

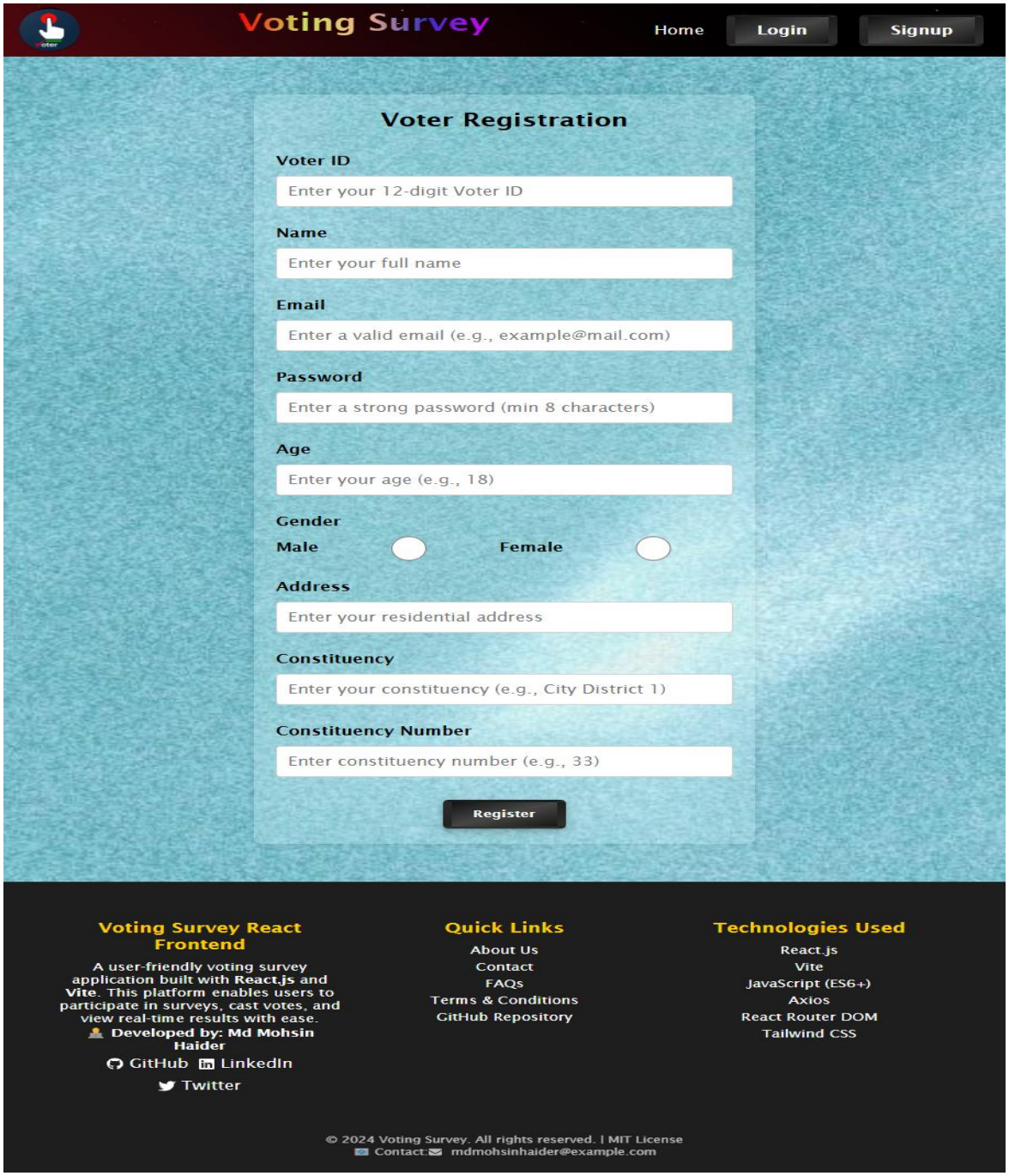

Fig. 8.3 Voting Register

CHAPTER - 9

CONCLUSION

CONCLUSION

In conclusion, the Voting Survey Application represents a comprehensive solution that marries modern web development techniques with practical voting and survey needs. The project demonstrates how technology can be effectively used to modernize traditional voting and surveying methods. The system is scalable, secure, and user-friendly, making it suitable for educational institutions, local elections, and organizational polls With future enhancements like advanced analytics, mobile app integration, or blockchain-based validation, the Voting Survey System can be further improved to support more complex and large-scale electoral processes.

the Voting Survey Application serves as a secure, efficient, and user-friendly platform that simplifies the process of conducting surveys and collecting votes digitally. By integrating essential features like user authentication, real-time vote tracking, admin control panels, and result visualization, the application ensures both accessibility and transparency. Its modular structure, built with modern web technologies and backed by a robust backend and database system, allows for scalability and easy maintenance. The successful implementation and testing of this project demonstrate the practical application of software development principles and problem-solving skills.

This project not only addresses the challenges of traditional voting systems but also lays a strong foundation for future enhancements such as live analytics, mobile support, and advanced security layers. Overall, the Voting Survey Application is a valuable tool for educational institutions, organizations, or any group requiring efficient and reliable survey-based decision-making.

Moreover, the application promotes transparency, reduces manual effort, and minimizes the risk of biased or manipulated results—challenges often seen in traditional voting systems. From design to deployment, the entire development cycle reinforced the importance of planning, testing, version control, and user-centric design. Looking ahead, the application can be extended with features such as mobile app integration, multi-language support, advanced analytics, blockchain-based voting for enhanced security, and real-time dashboards for live surveys. Overall, this project not only demonstrates the technical skills gained during its development but also provides a meaningful solution that can be adapted to real-world use cases across educational institutions, corporate environments, and social organizations.

By meticulously addressing user authentication, secure data handling, real-time vote tracking, and flexible administrative controls, the project not only streamlines the voting process but also underscores the importance of robust security and data integrity. The thoughtful implementation of features such as responsive UI design, modular component development, and multi-layered error handling provides users with an intuitive and efficient experience, while the admin panel empowers administrators with the tools to create, manage, and monitor surveys effortlessly. Rigorous testing—spanning unit, integration, system, usability, and security assessments—ensured that every element of the application met high standards of functionality and performance. This project not only highlights the successful application of advanced programming and database management principles but also sets the stage for future enhancements like live analytics, enhanced mobile responsiveness, and additional security measures such as two-factor authentication. Ultimately, the Voting Survey Application stands as a testament to innovative problem-solving and forward-thinking design in the digital voting landscape, providing a reliable and scalable platform for a variety of organizations.

During the project, various challenges were encountered, such as optimizing speech recognition accuracy, handling system compatibility issues across different operating systems, and minimizing delays between input and execution. Careful testing and debugging helped to resolve these challenges, resulting in a stable and efficient final product. Additionally, the implementation of features like error handling, command fallback mechanisms, and simple user guidance systems made the assistant more robust in real-world use.

The entire development cycle reinforced the importance of planning, testing, version control, and user-centric design. Looking ahead, the application can be extended with features such as mobile app integration, multi-language support, advanced analytics, blockchain-based voting for enhanced security, and real-time dashboards for live surveys.

Overall, the Voting Survey Application stands as a secure, efficient, and user-friendly platform that full fills its intended purpose and paves the way for real-world deployment.

CHAPTER - 10

FUTURE SCOPE

FUTURE SCOPE

The feature scope of the Voting Survey Application outlines the core functionalities included in the current version of the system. It supports user registration and login, allowing voters to securely access the platform and participate in active surveys. The application includes a role-based access system, where regular users can cast votes and view their participation history, while administrators can create, manage, and delete surveys. Admins can also define options, set deadlines, and monitor real-time voting statistics. The system ensures one vote per user per survey, maintaining the integrity of each poll. A dashboard interface is available for both users and admins, designed to be intuitive and user-friendly. In addition, the application incorporates data validation and error handling to improve usability and reliability. Basic data encryption and secure authentication methods (like password hashing) have been implemented to protect sensitive information.

The Voting Survey System has a strong foundation and can be further enhanced with advanced features and wider applicability. Some potential future improvements include:

1. **Mobile Application Integration**

- Developing a mobile app version for Android/iOS to increase accessibility and user engagement.

2. **Biometric Authentication**

- Integrating fingerprint or facial recognition to ensure even higher security during login and voting.

3. **Blockchain-Based Voting**

- Implementing blockchain technology for transparent, tamper-proof vote recording and result verification.

4. **Real-Time Analytics Dashboard**

- Providing administrators with visual insights, charts, and statistics for better decision-making.

5. **Multi-Language Support**

- Expanding the system to support regional languages, making it accessible to a broader user base.

6. **Offline Voting Support**

- Enabling offline voting through local sync options for remote or low-internet areas.

7. **Email and SMS Notifications**

- Sending real-time alerts and confirmations to users about voting schedules, results, or updates.

8. **Scalable to National-Level Elections**

- With proper security and infrastructure, the system can be scaled for municipal, state, or national-level elections.

The Voting Survey Application has significant potential for future enhancements and broader adoption. In its current form, it provides a reliable digital platform for conducting surveys and collecting votes; however, future updates can make it more robust, intelligent, and scalable. One promising direction is the integration of mobile application support, enabling users to participate in surveys from smartphones, thus increasing accessibility and engagement. Multi-language support can further extend its reach to users of different linguistic backgrounds. For enhanced security and transparency, blockchain technology could be incorporated to create an immutable record of votes.

Additionally, implementing real-time data visualization dashboards would offer live insights for administrators and voters alike. The application can also benefit from AI-powered analytics to detect voting patterns, generate reports, and prevent suspicious activities. For large-scale use, such as in educational institutions or organizations, features like bulk user import, offline voting modes, and notification systems (email/SMS) could be valuable. As the application evolves, compliance with legal and privacy regulations will also be important, ensuring it remains a trustworthy and adaptable solution for modern digital voting needs.

CHAPTER - 11

REFERENCES

REFERENCES

- **Web Development Technologies**
  - *Mozilla Developer Network (MDN)* – Used for understanding HTML, CSS, and JavaScript syntax and best practices.
    Link: https://developer.mozilla.org

- **Backend Frameworks & Documentation**
  - *Node.js / Express.js Documentation* – For implementing server-side routing and API handling.
    Link: https://expressjs.com
  - *Python Flask/Django Documentation* – If used, for setting up backend routes and database connections.
    Link: https://flask.palletsprojects.com / https://docs.djangoproject.com

- **Database Design & Querying**
  - *W3Schools SQL Tutorial* – For understanding SQL queries and relational database structure.
    Link: https://www.w3schools.com/sql
  - *MongoDB Documentation* – If NoSQL was used for storing user and voting data.
    Link: https://www.mongodb.com/docs/

- **Authentication & Security**
  - *bcrypt.js GitHub Repository* – Used for password hashing and secure storage.
    Link: https://github.com/kelektiv/node.bcrypt.js
  - *JWT (JSON Web Tokens) Documentation* – For implementing secure token-based authentication.
    Link: https://jwt.io

- **UI/UX Design Resources**
  - *Bootstrap Documentation* – For responsive and modern frontend design.
    Link: https://getbootstrap.com
  - *Figma / Canva* – For designing wireframes and mockups.
    Link: https://www.figma.com / https://www.canva.com

- **Project Development Tools**
  - *GitHub* – Used for version control and collaborative development.
    Link: https://github.com
  - *Postman* – For API testing during backend development.
    Link: https://www.postman.com

- **Online Learning Platforms (Optional)**

- *Geeks for Geeks, Coursera, YouTube Tutorials* – For understanding full-stack application flow and concepts.
  Links:
  - https://www.geeksforgeeks.org
  - https://www.coursera.org
  - https://www.youtube.com

- **Cybersecurity and Best Practices**
  - *OWASP Top 10* – Industry-standard resource on common web application security vulnerabilities and how to prevent them.
    Link: https://owasp.org/www-project-top-ten/

- **REST API Standards**
  - *RESTful API Design Guidelines* – Best practices followed while designing the API structure in the project.
    Link: https://restfulapi.net/

- **HTML, CSS, and JS Deep Dive**
  - *freeCodeCamp* – Interactive tutorials and documentation for frontend development and DOM manipulation.
    Link: https://www.freecodecamp.org/

- **Version Control**
  - *Git Official Documentation* – Used to understand version control and collaborative development during the project lifecycle.
    Link: https://git-scm.com/doc

- **Deployment Resources**
  - *Heroku* – Platform used for deploying the project's backend or full stack.
    Link: https://devcenter.heroku.com
  - *Netlify / Vercel* – For frontend deployment if applicable.
    Links: https://www.netlify.com, https://vercel.com

- **Database Tutorials**
  - *SQLZoo / DB Fiddle / SQLite Documentation* – For understanding and testing SQL queries.
    Links:
    - https://sqlzoo.net
    - https://www.db-fiddle.com
    - https://www.sqlite.org/docs.html

CHAPTER - 12

APPENDICES

APPENDICES

The appendices section includes supplementary materials that support the main content of the Voting Survey Application project report. It contains source code snippets of key modules such as user authentication, survey creation, and vote submission to provide deeper technical insight into the application's backend logic. Screenshots of the user interface, including the registration page, login page, dashboard, and admin panel, are provided to visually demonstrate the design and functionality. Additionally, database schema diagrams are included to illustrate the structure of data storage and relationships between entities such as users, surveys, and responses.

Appendix A: Source Code Snippets

- Key functions for user registration, login authentication, and session management
- Survey creation and vote submission logic
- API integration examples between frontend and backend

Appendix B: UI Screenshots

- Voter Registration Page
- Login Page
- Voter Dashboard
- Admin Panel
- Survey Creation and Results Page

Appendix C: Database Schema

- Entity-Relationship (ER) diagram
- Table structure and relationships (Users, Surveys, Votes, Options)

Appendix D: Sample Test Cases

- Functional test cases for login, registration, voting, and admin actions
- Validation and boundary test scenarios
- Result screenshots from testing

Appendix E: Tools and Technologies Used

- Frontend: HTML, CSS, JavaScript (or React)
- Backend: Java / Python / Node.js (depending on your stack)
- Database: MySQL / MongoDB
- Others: GitHub (version control), Firebase/Heroku (deployment), Postman (API testing)

Appendix F: External Libraries/Frameworks

- List of libraries (e.g., Bootstrap, Express.js, bcrypt, JWT, etc.)
- Purpose and version of each library used in the project