



DAYANANDA SAGAR
UNIVERSITY



SCHOOL OF
ENGINEERING

Dayananda Sagar University

School of Engineering

Devarakaggalahalli, Harohalli, Kanakapura Road, Ramanagara Dt., Bengaluru – 562 112

Department of Computer Science & Engineering

Mini Project Report on

Spring Boot

Job Portal

By

K Kavya -ENG22CS0076

Kruthika B N - ENG22CS0084

Sneha -ENG22CS0173

Tanuja D -ENG22CS0196

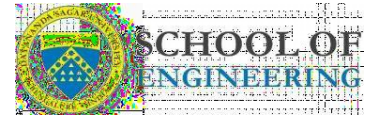
Under the supervision of

Dr.Gokulakrishnan S

Assistant Professor,
Department of Computer Science and Engineering



DAYANANDA SAGAR
UNIVERSITY



Dayananda Sagar University

School of Engineering

Devarakaggalahalli, Harohalli, Kanakapura Road, Ramanagara Dt., Bengaluru – 562 112

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the work titled “ **Job Portal** ” is carried out by **K Kavya(ENG22CS0076), Kruthika B N (ENG22CS0084), Sneha(ENG22CS0173), Tanuja D(ENG22CS0196)**. Bonafide students of Bachelor of Technology in Computer Science and Engineering at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and e, during the year **2024- 2025**.

Dr. Gokulakrishnan S

Assistant Professor, Dept. of CSE,
School of Engineering,
Dayananda Sagar University.

Dr. Girisha G S

Chairperson CSE,
School of Engineering,
Dayananda Sagar University.

Dayananda Sagar University

School of Engineering

Devarakaggalahalli, Harohalli, Kanakapura Road, Ramanagara Dt., Bengaluru – 562 112

Department of Computer Science & Engineering

DECLARATION

We, **K.Kavya(ENG22CS0076), Kruthika.B.N(ENG22CS0084), Sneha(ENG22CS0173), Tanuja D(ENG22CS0196)** are students of the fifth semester B.Tech in Computer Science and Engineering, at School of Engineering, Dayananda Sagar University, hereby declare that the mini-project titled “**Job Portal** ” has been carried out by us and submitted in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering during the academic year 2024-2025.

Student Signature

Name1: K Kavya

USN : ENG22CS0076

Name2: Kruthika B N

USN : ENG22CS0084

Name3: Sneha

USN : ENG22CS0173

Name4: Tanuja D

USN: ENG22CS0196

Place : Bangalore

Date :

ABSTRACT

1 **Set Up the Project:**

- Use Spring Initializr to create the project.
- Add dependencies: Spring Web, Spring Data JPA, Spring Security, etc.
- Configure database connectivity in application.yml or application.properties.

2 **Domain Layer:**

- Define entities like Job, User, and Application.
- Use annotations like @Entity and @Id for mapping to the database.

3 **Repository Layer:**

- Create repository interfaces (e.g., JobRepository, UserRepository) using JpaRepository for database operations.
- Add custom query methods (e.g., findByLocation, findByTitleContaining).

4 **Service Layer:**

- Implement business logic in service classes (e.g., JobService).
- Use services as an abstraction between controllers and repositories.

5 **Controller Layer:**

- Define REST API endpoints using controllers (e.g., JobController).
- Handle HTTP methods like GET, POST, and DELETE for managing jobs.

6 **Authentication & Authorization :**

- Use Spring Security for user authentication and role-based access control.
- Secure endpoints with role permissions.

7 **Testing and Deployment:**

- Test APIs with tools like Postman.
- Package the application into a JAR/WAR and deploy on servers or Doc

Job Portal

INTRODUCTION

A **job portal** is an online platform that connects job seekers with employers, facilitating the recruitment process. It serves as a hub where companies post job vacancies, and candidates can search, apply, and track job opportunities. Building a job portal using **Spring Boot** provides a robust, scalable, and maintainable solution due to Spring Boot's powerful features and ease of integration with modern technologies.

Why Spring Boot for a Job Portal?

Spring Boot is an ideal framework for developing a job portal because it offers:

- **Rapid Development:** Provides built-in configurations, reducing boilerplate code.
- **RESTful APIs:** Easy creation of APIs for managing jobs, users, and applications.
- **Database Integration:** Seamless integration with databases using Spring Data JPA.
- **Security:** Built-in Spring Security for user authentication and role-based access control.
- **Scalability:** Suitable for both small-scale and enterprise-level applications.

Features of the Job Portal:

- **User Management:** Allows user registration for different roles (e.g., job seekers, recruiters, admin).
- **Job Listings:** Enables recruiters to post job opportunities and candidates to browse available jobs.
- **Search and Filters:** Supports job search by keywords, location, and other criteria.
- **Applications:** Allows job seekers to apply for jobs and recruiters to track applications.
- **Authentication and Authorization:** Ensures secure access to resources based on user roles.

Objective:

To design and develop a modular, feature-rich job portal using Spring Boot that simplifies the recruitment process and ensures an efficient user experience for both employers and job seeker

Project Description

The **Job Portal** is a web-based application designed to connect job seekers with employers, streamlining the hiring process. This system provides an interactive platform for users to browse job opportunities, apply for positions, and manage recruitment activities. Built using **Spring Boot**, the project ensures scalability, security, and ease of development.

Purpose

The primary purpose of the job portal is to:

- Simplify the recruitment process for employers and job seekers.
- Provide a centralized platform for managing job postings and applications.
- Offer a seamless and secure user experience through role-based access control and intuitive interfaces.

For Job Seekers:

- User Registration and Login: Create and manage personal profiles.
- Job Search: Browse and filter job opportunities based on criteria like location, job title, and company.
- Job Applications: Apply for jobs and track application status.

For Employers:

- Post Job Openings: Add and manage job listings.
- Application Management: View and filter candidate applications.
- User Dashboard: Monitor job postings and interact with applicants.

Admin Features:

- User Management: Manage job seekers and employer accounts.
- Job Approval: Moderate and approve job listings.

Technology Stack

1. Backend:

- Spring Boot for application logic and API development.
- Spring Data JPA for database operations.
- Spring Security for authentication and authorization.

2. Frontend:

- Thymeleaf or React/Angular for user interfaces (optional).

3. Database:

- MySQL/PostgreSQL for data storage.

4. Tools:

- Maven for build automation.
- Postman for API testing.
- Docker (optional) for containerized deployment.

Workflow

- User Authentication: Users register and log in based on their roles (Job Seeker, Employer, Admin).
- Job Posting: Employers create and manage job listings.
- Job Search: Job seekers browse available positions and apply.
- Application Tracking: Employers and job seekers can track the status of applications.

Social Impact of the Job Portal

Social Impact

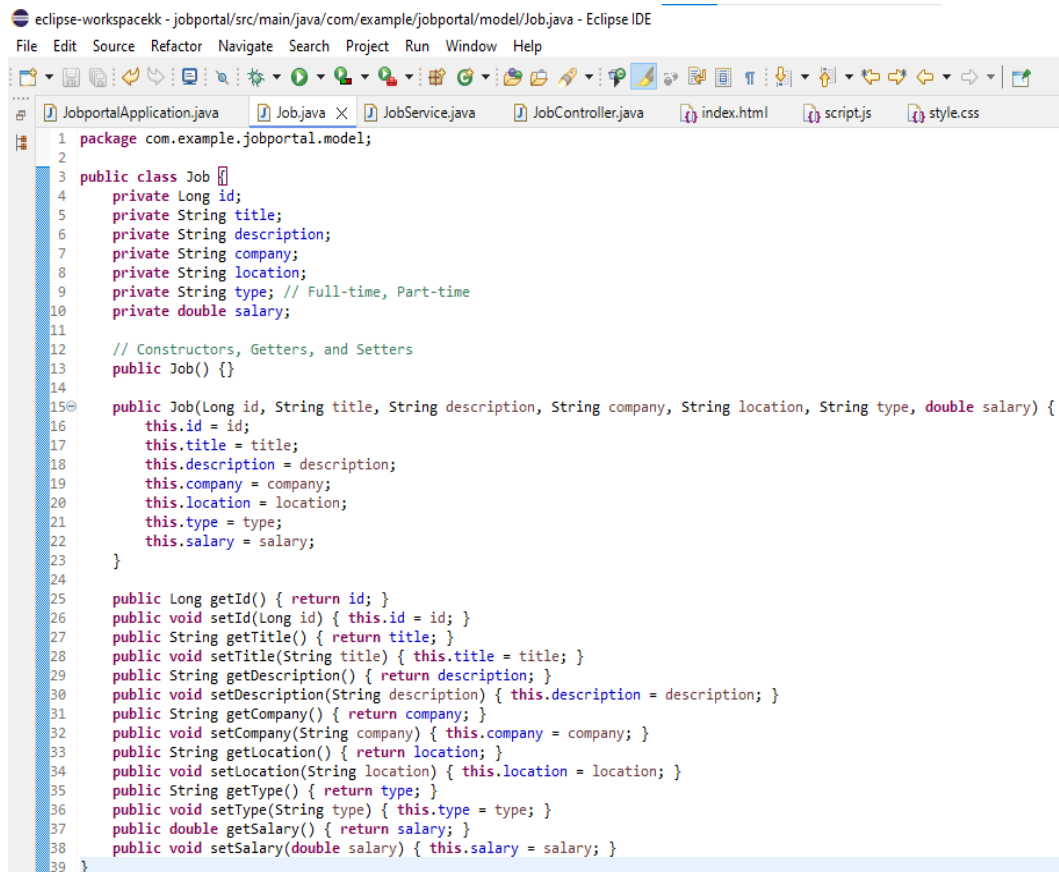
The job portal can create a significant social impact by bridging the gap between job seekers and employers, fostering economic growth and reducing unemployment. By providing an accessible platform for individuals to find opportunities that match their skills, it empowers job seekers to advance their careers and improve their quality of life. Employers, in turn, can connect with a diverse pool of candidates, promoting inclusivity and diversity in the workplace. The portal's ability to reach underserved communities, such as rural areas or economically disadvantaged groups, ensures equal access to employment opportunities, driving social equity. Additionally, features like remote job listings support modern work trends and enable individuals to overcome geographical barriers, contributing to a more connected and inclusive job market. Ultimately, the job portal acts as a catalyst for personal and professional development, fostering a more equitable and dynamic workforce. Unintended Biases. The job portal has the potential to bring transformative social benefits by addressing critical challenges in the employment ecosystem. Unemployment and underemployment are persistent global issues, often exacerbated by a lack of accessible resources and information about available opportunities. This portal democratizes access to jobs, ensuring that individuals from all socioeconomic backgrounds, including marginalized and disadvantaged communities, can compete in the job market on an equal footing.

Working Description

Backend Code (Spring Boot)

- The project utilizes Spring Boot, a popular Java-based framework, for its backend. This framework provides a robust foundation for building RESTful web services and efficient application logic.
- Spring Boot simplifies the development process by automating configuration and offering ready-to-use dependencies. This allows us to focus on building core features and functionality rather than spending time on low-level setup.

Job Class:



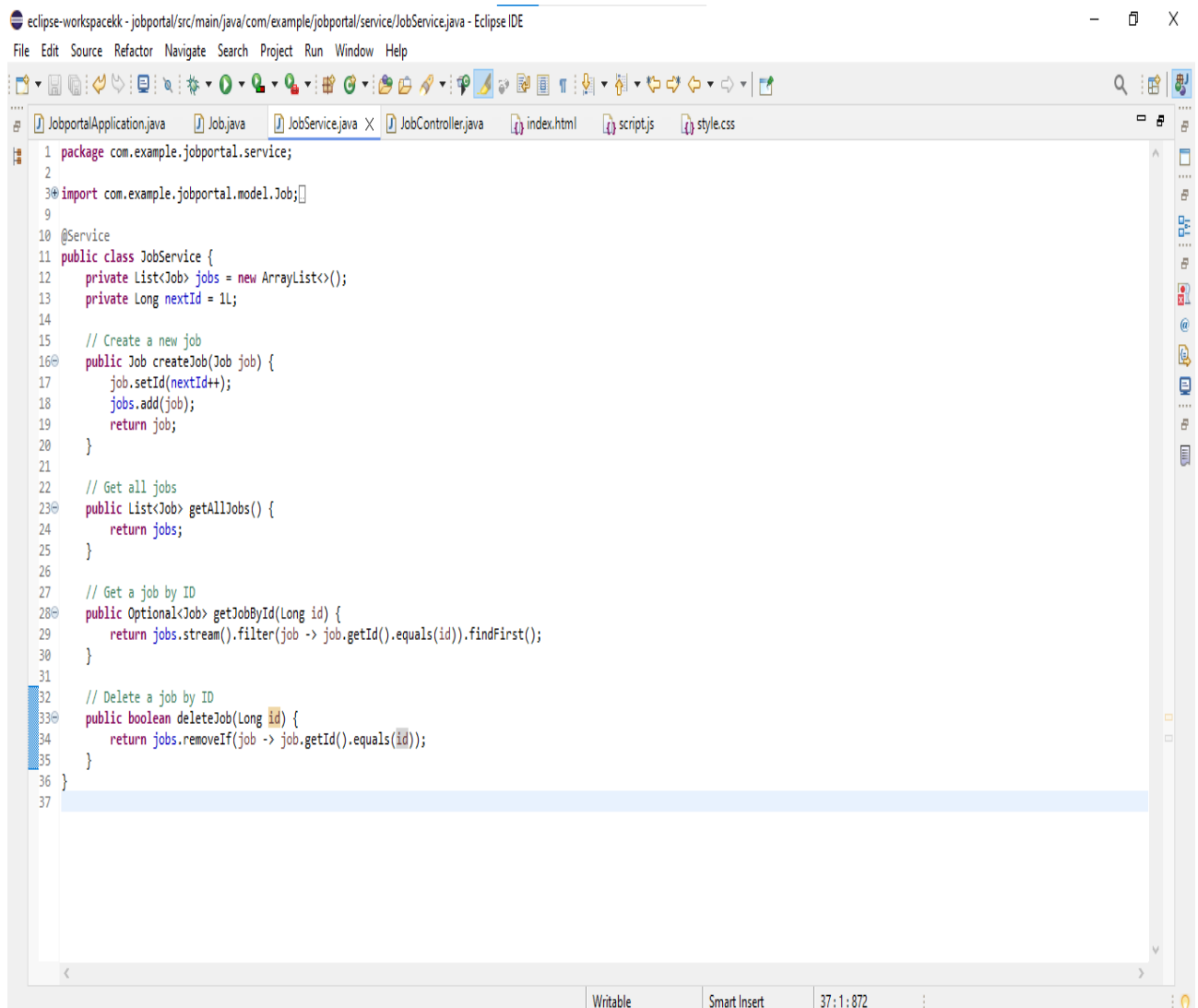
```
1 package com.example.jobportal.model;
2
3 public class Job {
4     private Long id;
5     private String title;
6     private String description;
7     private String company;
8     private String location;
9     private String type; // Full-time, Part-time
10    private double salary;
11
12    // Constructors, Getters, and Setters
13    public Job() {}
14
15    public Job(Long id, String title, String description, String company, String location, String type, double salary) {
16        this.id = id;
17        this.title = title;
18        this.description = description;
19        this.company = company;
20        this.location = location;
21        this.type = type;
22        this.salary = salary;
23    }
24
25    public Long getId() { return id; }
26    public void setId(Long id) { this.id = id; }
27    public String getTitle() { return title; }
28    public void setTitle(String title) { this.title = title; }
29    public String getDescription() { return description; }
30    public void setDescription(String description) { this.description = description; }
31    public String getCompany() { return company; }
32    public void setCompany(String company) { this.company = company; }
33    public String getLocation() { return location; }
34    public void setLocation(String location) { this.location = location; }
35    public String getType() { return type; }
36    public void setType(String type) { this.type = type; }
37    public double getSalary() { return salary; }
38    public void setSalary(double salary) { this.salary = salary; }
39 }
```

JobController Class.

```
package com.example.jobportal.controller;
import com.example.jobportal.model.Job;
import com.example.jobportal.service.JobService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/jobs")
public class JobController {
    @Autowired
    private JobService jobService;
    // Get all jobs
    @GetMapping
    public List<Job> getAllJobs() {
        return jobService.getAllJobs();
    }
    @PostMapping
    public ResponseEntity<Job> createJob(@RequestBody Job job) {
        Job createdJob = jobService.createJob(job);
        return new ResponseEntity<>(createdJob, HttpStatus.CREATED);
    }
    @GetMapping("/{id}")
    public ResponseEntity<Job> getJobById(@PathVariable Long id) {
        return jobService.getJobById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteJob(@PathVariable Long id) {
        boolean deleted = jobService.deleteJob(id);
        if (deleted) {
            return ResponseEntity.noContent().build();
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

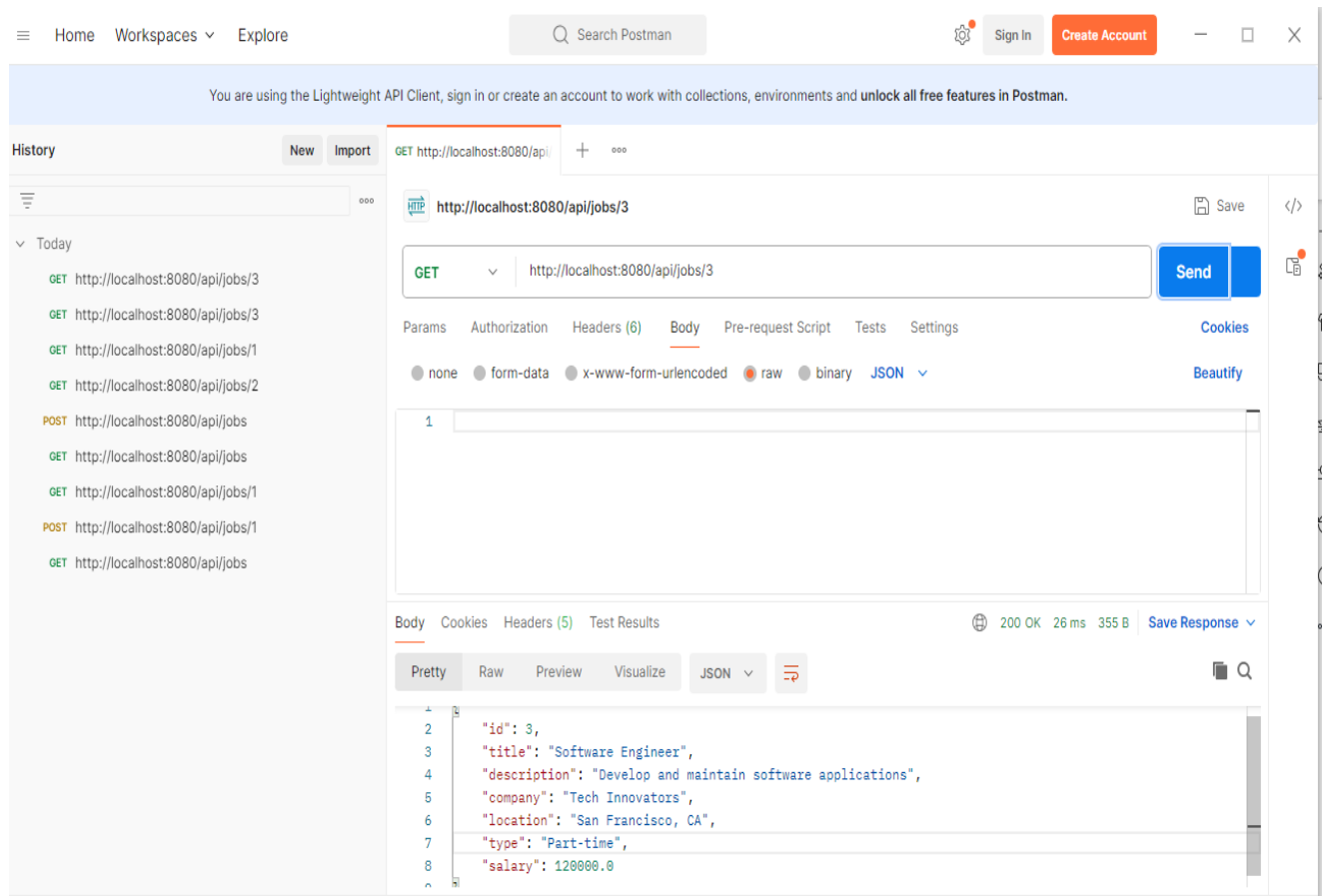
JobService

The screenshot shows the Eclipse IDE interface with the 'JobService.java' file open. The file is located at 'eclipse-workspace\src\main\java\com\example\jobportal\service\JobService.java'. The code defines a Spring Service class for managing jobs. It includes a package declaration, an import for the Job model, and a @Service annotation. The class contains a private List of jobs and a private Long for the next ID. It implements methods for creating, getting all, getting by ID, and deleting jobs. The IDE's toolbar and project explorer are visible at the top and right respectively. The status bar at the bottom shows 'Writable', 'Smart Insert', and the cursor position '37: 1: 872'.

Purpose of JobService

The JobService class is responsible for handling the business logic related to jobs. This includes fetching data from the database, performing validations, and updating the data

Building REST APIs for Job Management



To build a REST API for Job Management using Spring Boot, the process involves several steps. First, you create a Spring Boot project with dependencies such as Spring Web, Spring Data JPA, and a database (like H2 or MySQL). Next, you define the Job entity, which will represent the job object in your system, with fields like id, title, description, status, and timestamps. You then create a JobRepository interface that extends JpaRepository to handle CRUD operations on the Job entity. In the service layer, you implement business logic to create, update, delete, and fetch jobs. The service layer interacts with the JobRepository to perform database operations. Finally, you build a REST controller with endpoints for handling HTTP requests. These endpoints allow clients to perform operations like creating a new job, retrieving all jobs, fetching a job by its ID, updating job details, and deleting a job. By utilizing Spring Boot's auto-configuration and dependency injection, the application becomes easy to maintain and scale while interacting with a relational database through Spring Data JPA. This REST API can be extended with additional features like authentication, validation, and pagination for a more robust and production-ready solution.

Integrating the Backend with the Fetch API

Fetch API

The Fetch API provides a modern way for JavaScript code to interact with the backend REST APIs, allowing us to retrieve, create, update, and delete job data seamlessly.

Asynchronous Requests

The Fetch API handles requests asynchronously, ensuring the frontend remains responsive while data is being retrieved or processed.

JSON Data Handling

We'll work with JSON data format for communication between the backend and frontend, providing a lightweight and efficient way to exchange information.

localhost:8080/index.html

Job Portal

Add a New Job

Job Title

Description

Company

Location

Full-time

Salary

Add Job

All Jobs

"Marketing Manager at Creative Media (Los Angeles, CA)
Oversee marketing campaigns and lead generation
Type: Full-time, Salary: \$60000

Delete

Software Developer at TechCorp (Remote)
Build and maintain software applications
Type: Full-time, Salary: \$75000

Delete

Software Engineer at Tech Innovators (San Francisco, CA)
Develop and maintain software applications
Type: Part-time, Salary: \$120000

Delete

Conclusion

In conclusion, building a job portal using Spring Boot offers a powerful and efficient way to manage job listings and provide a seamless user experience. By leveraging Spring Boot's robust features like dependency injection, automatic configuration, and integration with Spring Data JPA for database management, we can easily build a scalable, maintainable, and secure backend for job-related operations. The RESTful APIs provide a standardized way to interact with job data, enabling actions like job creation, updates, retrieval, and deletion. With the flexibility to integrate additional features like authentication, pagination, and filtering, a Spring Boot-based job portal can handle large volumes of users and job listings while maintaining performance and security. This approach lays a solid foundation for building more advanced features, such as job search, notifications, or even integration with third-party services, making it a versatile and reliable solution for job management in a modern web application.