# **Core functionalities of Phase 2**

**Backend**

**Source Code for Employee Controller**

package com.innovators.jobreferralportal.controller;  
  
import com.innovators.jobreferralportal.Service.EmployeeService;  
import com.innovators.jobreferralportal.Service.JobService;  
import com.innovators.jobreferralportal.entity.Job;  
import com.innovators.jobreferralportal.entity.ReferredCandidate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.IOException;  
import java.sql.Ref;  
import java.util.List;  
  
  
@RestController  
@RequestMapping("/employee")  
public class EmployeeController {  
  
 @Autowired  
 JobService jobService;  
 @Autowired  
 private EmployeeService employeeService;  
  
 @PostMapping("/referCandidate")  
 public ResponseEntity<String> referCandidate(@RequestPart("resume") MultipartFile resume,  
 @RequestParam("fName") String fName,  
 @RequestParam("lName") String lName,  
 @RequestParam("yearsOfExp") int yearsOfExp,  
 @RequestParam("referredBy") int referredBy){

// now creating referred candidate object  
 try {  
 ReferredCandidate candidate = ReferredCandidate.builder()  
 .fName(fName)  
 .lName(lName)  
 .yearsOfExp(yearsOfExp)  
 .referredBy((long) referredBy)  
 .resume(resume.getBytes())  
 .build();  
  
 employeeService.referCandidate(candidate);  
 return ResponseEntity.*ok*("Candidate referred successfully!");  
 } catch (IOException e) {  
 return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).body("Error referring candidate: " + e.getMessage());  
 }  
  
  
 }  
  
 @GetMapping("/getAllJobs")  
 public ResponseEntity<List<Job>> getAllJobs(){  
 List<Job> opList = employeeService.getAllJobs();  
 return ResponseEntity.*ok*(opList);  
 }

@GetMapping("/getAllReferredCandidates")  
 public ResponseEntity<List<ReferredCandidate>> getAllReferredCandidates(HttpServletRequest request) {  
 HttpSession session = request.getSession();  
 Long employeeId = (Long) session.getAttribute("employeeID");  
  
 if (employeeId == null) {  
 return ResponseEntity.*status*(HttpStatus.*UNAUTHORIZED*).body(null);  
 }  
  
 List<ReferredCandidate> opList = employeeService.getAllReferredCandidatesByEmployeeId(employeeId);  
 return ResponseEntity.*ok*(opList);  
 }  
  
  
 @GetMapping("/search")  
 public List<Job> searchJob(@RequestParam String positionName) {  
 return jobService.searchJob(positionName);  
 }  
 @DeleteMapping("/deleteReferral/{id}")  
 public void deleteReferral(@PathVariable Long id) {  
 employeeService.deleteReferral(id);  
 }  
  
 @GetMapping("/getLeaderBoard")  
 public ResponseEntity<List<List<String>>> leaderBoardList(){  
 List<List<String>> leaderBoardList = hrService.getLeaderBoardList();  
 if (leaderBoardList.isEmpty()){  
 return ResponseEntity.*noContent*().build();  
 }else{  
 return ResponseEntity.*ok*(leaderBoardList);  
 }  
 }  
}

**Description**

* Controller acts as a bridge between the user interface and the model or logic. The employee performs many tasks to refer a candidate for a particular position.
* Firstly, to refer a candidate, they need all the details about the candidate. Here, we are providing details like first name, last name, year of experience, and resume using this end point “/referCandidate”.
* Next, we have “/getAllJobs” where all the jobs will be displayed which are posted by the HR as bulk or individually. By using this feature, the employee can refer a candidate based on their skills, and experience.
* "/getAllReferredCandidates" where all the candidates who have been referred by the employeewill be displayed. By using this feature, the employee can see whom he/she have referred in all time.
* "/search" is used to search the requirement based on the positionname. For an instance, if an employee want to search for a particular job position like Software Engineering, they can type and easily search it and refer a candidate for it.
* “/deleteReferral” is used for deleting the referral that the employee had referred. When there is an situation where the employee had given incorrect details or referred for different position, the employer can delete the referral.
* “/getLeaderboard” is given to the employees to track and display the performance based on the referrals.

**Source Code for HR Controller**

package com.innovators.jobreferralportal.controller;  
  
  
import com.innovators.jobreferralportal.Service.HRService;  
import com.innovators.jobreferralportal.Service.JobService;  
import com.innovators.jobreferralportal.entity.Employee;  
import com.innovators.jobreferralportal.entity.Job;  
import com.innovators.jobreferralportal.entity.ReferredCandidate;  
import com.innovators.jobreferralportal.repository.EmployeeRepo;  
import com.innovators.jobreferralportal.repository.ReferredCandidateRepo;  
import org.apache.coyote.Response;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.core.io.ByteArrayResource;  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.util.List;  
import java.util.Objects;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/hr")  
public class HRController {  
  
 @Autowired  
 private JobService jobService;  
  
 @Autowired  
 private HRService hrService;  
  
 @Autowired  
 private EmployeeRepo employeeRepo;  
  
 @Autowired  
 private ReferredCandidateRepo referredCandidateRepo;  
  
  
 private final Logger LOGGER =  
 LoggerFactory.*getLogger*(HRController.class);  
  
 @PostMapping("/addJob")  
 public Job addJob(@RequestBody Job jobPosting) {  
 LOGGER.info("Saving Job triggered");  
 return jobService.addJob(jobPosting);  
  
 }  
  
 @PutMapping("/updateJob/{jobId}")  
 public ResponseEntity<String> updateJob(@PathVariable Long jobId, @RequestBody Job updatedJob) {  
 if (jobService.updateJob(jobId, updatedJob)) {  
 return ResponseEntity.*ok*("Job updated successfully");  
 } else {  
 return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).body("Job not found");  
 }  
 }  
  
 @DeleteMapping("/deleteJob/{jobId}")  
 public ResponseEntity<String> deleteJob(@PathVariable Long jobId) {  
 jobService.deleteJob(jobId);  
 return ResponseEntity.*ok*("Job deleted successfully");  
  
 }  
  
 //hr needs to fetch all records  
 @GetMapping("/getAllReferredCandidates")  
 public ResponseEntity<List<ReferredCandidate>> getAllReferredCandidates() {  
 List<ReferredCandidate> candidates = hrService.getAllReferredCandidates();  
 if (candidates.isEmpty()) {  
 return ResponseEntity.*noContent*().build();  
 }  
 return ResponseEntity.*ok*(candidates);  
 }  
@GetMapping("/downloadResume/{id}")  
public ResponseEntity<ByteArrayResource> downloadResume(@PathVariable Long id) {  
 try {  
 ReferredCandidate candidate = referredCandidateRepo.getReferenceById(id);  
 if (candidate != null && candidate.getResume() != null) {  
 ByteArrayResource resource = new ByteArrayResource(candidate.getResume());  
 return ResponseEntity.*ok*()  
 .header(HttpHeaders.*CONTENT\_DISPOSITION*, "attachment;filename=resume\_" + id + ".pdf")  
 .contentType(MediaType.*APPLICATION\_PDF*)  
 .body(resource);  
 }  
 return ResponseEntity.*notFound*().build();  
 } catch (Exception e) {  
 return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).build();  
 }  
}  
  
  
@PutMapping("/updateStatus")  
public ResponseEntity<String> updateCandidateStatus(@RequestParam String statusUpdate, @RequestParam Long id){  
 //Now check if Status is "ACCEPTED" now a method needs to be called  
 // to increase the count for employee, this method should be using  
 //pickup referredBy id(use repo) and use that to increment counter with the employeeID  
 if(statusUpdate.equals("ACCEPTED")){  
 hrService.scoreIncrementor(id);  
 }  
 if( hrService.updateStatus(statusUpdate,id)) {  
 return ResponseEntity.*ok*("Status updated Successfully");  
 }else {  
 return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).body("Failed to update");  
 }  
}  
  
//LeaderBoard score needs to accessible for both HR and Employee  
@GetMapping("/getLeaderBoard")  
public ResponseEntity<List<List<String>>> leaderBoardList(){  
 List<List<String>> leaderBoardList = hrService.getLeaderBoardList();  
 if (leaderBoardList.isEmpty()){  
 return ResponseEntity.*noContent*().build();  
 }else{  
 return ResponseEntity.*ok*(leaderBoardList);  
 }  
}  
@GetMapping("/getAllJobs")  
public ResponseEntity<List<Job>> getAllJobs(){  
 List<Job> opList = jobService.getAllJobs();  
 return ResponseEntity.*ok*(opList);  
}

@GetMapping("/search")  
public List<Job> searchJob(@RequestParam String keyword) {  
 return jobService.searchJob(keyword);  
}  
  
  
@GetMapping("/searchCandidates")  
public ResponseEntity<List<ReferredCandidate>> getReferredCandidates(@RequestParam String positionName){  
 List<ReferredCandidate> opList = hrService.getAllReferredCandidatesSearch(positionName);  
 return ResponseEntity.*ok*(opList);  
}

**Description**

* HR adds the jobs so that the employees can refer a candidate. They post the jobs in the referral portal. The employee can access for referring candidates by using "/addJob” HR can fill out details of the job and hit the add button.
* Updating of job is also can be modified by HR. These details can be updated and stored in the database by using "/updateJob”
* When a candidate is been referred to the job position, then that position is no longer needed to be in listings. So the HR can delete the job by using "/deleteJob”
* Even, HR needs to know the referred candidates to continue the further referral process. It can be done by using "/getAllReferredCandidates"
* "/downloadResume/{id}" will help the HR to download the resume which is given by the employee while referring a candidate. There are further more process after this, but resume is one of the most important aspect to decide to accept or reject the candidate.
* "/updateStatus" method is responsible for updating the status of a candidate based on a provided statusUpdate and id.
* "/getLeaderBoard" is the method used in retrieving a leaderboard like representing a ranking of employees based on a referral or scores.
* "/getAllJobs" is the method used to fetch a list of all available jobs to the HR.
* “/search” is used for searching job postings based on a keyword.

**Source code for User Controller**

package com.innovators.jobreferralportal.controller;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import com.innovators.jobreferralportal.entity.Employee;  
import com.innovators.jobreferralportal.repository.EmployeeRepo;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;  
import org.springframework.web.bind.annotation.\*;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import jakarta.servlet.http.HttpSession;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/users")  
public class UserController {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserController.class);  
  
 @Autowired  
 private PasswordEncoder passwordEncoder;  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private EmployeeRepo employeeRepository;  
  
 @PostMapping("/login")  
 public ResponseEntity<String> login(@RequestParam String username, @RequestParam String password, HttpServletRequest request) {  
 try {  
 Authentication authentication = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(username, password));  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 request.getSession().setAttribute("SPRING\_SECURITY\_CONTEXT", SecurityContextHolder.*getContext*());  
  
 Optional<Employee> employeeOpt = employeeRepository.findByUsername(username);  
 if (employeeOpt.isPresent()) {  
 Employee employee = employeeOpt.get();  
 HttpSession session = request.getSession(true);  
 session.setAttribute("employeeID", employee.getEmployeeID());  
 session.setMaxInactiveInterval(30 \* 60);  
 return ResponseEntity.*ok*("Login successful for user: " + employee.getUsername());  
 } else {  
 *logger*.warn("Login failed - user not found: {}", username);  
 return ResponseEntity.*status*(404).body("User not found");  
 }  
 } catch (Exception e) {  
 *logger*.error("Login failed for user: {} - Error: {}", username, e.getMessage());  
 return ResponseEntity.*status*(401).body("Invalid credentials");  
 }  
 }  
  
 @PostMapping("/logout")  
 public ResponseEntity<String> logout(HttpServletRequest request, HttpServletResponse response) {  
 Authentication auth = SecurityContextHolder.*getContext*().getAuthentication();  
 if (auth != null) {  
 *logger*.info("Logging out user: {}", auth.getName());  
 new SecurityContextLogoutHandler().logout(request, response, auth);  
 }  
 *logger*.info("Logout successful");  
 return ResponseEntity.*ok*("Logout successful");  
 }

@PostMapping("/addUsers")  
public ResponseEntity<Map<String, String>> addUsers(@RequestBody Employee employee) {  
 *logger*.info("Attempting to add user: {}", employee.getUsername());  
 if (employee.getPassword() != null) {  
 String encodedPassword = passwordEncoder.encode(employee.getPassword());  
 employee.setPassword(encodedPassword);  
 }  
 try {  
 employeeRepository.save(employee);  
 *logger*.info("User added successfully: {}", employee.getUsername());  
 Map<String, String> response = new HashMap<>();  
 response.put("message", "User added successfully: " + employee.getUsername());  
 return ResponseEntity.*ok*(response);  
 } catch (Exception e) {  
 *logger*.error("Error adding user: {} - Error: {}", employee.getUsername(), e.getMessage());  
 Map<String, String> response = new HashMap<>();  
 response.put("error", "Error adding user: " + e.getMessage());  
 return ResponseEntity.*status*(500).body(response);  
 }  
}

**Description**

* This user controller will deal with the login and logout which is the authentication function. There will be secure access to this referral portal
* login() will validate the user credentials like checking the username and password with the database.
* logout() will make the employee leave the referral portal. It terminates the active session. This will improve the security and user experience.
* "/addUsers" is the method where it will receive an employee object from the request and encodes the password. It will add the user successfully. First, the employee needs to login into the portal and see the available positions and then, refer a candidate, if it is a success then a pop up specifying “user added successfully”

**Employee Service**

package com.innovators.jobreferralportal.Service;  
  
import com.innovators.jobreferralportal.entity.Job;  
import com.innovators.jobreferralportal.entity.ReferredCandidate;  
import com.innovators.jobreferralportal.repository.JobRepo;  
import com.innovators.jobreferralportal.repository.ReferredCandidateRepo;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class EmployeeServiceImpl implements EmployeeService{  
  
 @Autowired  
 private ReferredCandidateRepo referredCandidateRepo;  
  
 @Autowired  
 private JobRepo jobRepo;

@Override  
 public void referCandidate(ReferredCandidate referredCandidate) {  
 referredCandidateRepo.save(referredCandidate);  
 }  
  
@Override  
 public List<ReferredCandidate> getAllReferredCandidatesByEmployeeId(Long employeeId) {  
 return referredCandidateRepo.findByReferredBy(employeeId);  
   
 }  
 @Override  
 public void deleteReferral(Long id) {  
 referredCandidateRepo.deleteById(id);  
 }  
}

**Description**

* In employee services we have referCandidate() which allows the employee to refer a candidate based on their skills. We are saving the details in the database.
* getAllReferredCandidatesByEmployeeId will give a list with all the candidates who are referred by a particular employee
* deleteREferral is used for deleting the referral by the employee.

**HR Service**

package com.innovators.jobreferralportal.Service;  
  
  
import com.innovators.jobreferralportal.controller.HRController;  
import com.innovators.jobreferralportal.entity.Employee;  
import com.innovators.jobreferralportal.entity.ReferredCandidate;  
import com.innovators.jobreferralportal.repository.EmployeeRepo;  
import com.innovators.jobreferralportal.repository.JobRepo;  
import com.innovators.jobreferralportal.repository.ReferredCandidateRepo;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
//import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class HRServiceImpl implements HRService{  
  
 @Autowired  
 private EmployeeRepo employeeRepo;  
  
 @Autowired  
 private ReferredCandidateRepo referredCandidateRepo;  
  
 private final Logger LOGGER =  
 LoggerFactory.*getLogger*(HRServiceImpl.class);  
  
 @Override  
 public List<ReferredCandidate> getAllReferredCandidates() {  
  
 List<ReferredCandidate> referredCandidateList = referredCandidateRepo.findAll();  
 return referredCandidateList;  
 }  
  
 @Override  
 public boolean updateStatus(String status, Long id) {  
 Optional<ReferredCandidate> referredCandidate = referredCandidateRepo.findById(id);  
 if(referredCandidate.isPresent()){  
 ReferredCandidate candidate = referredCandidate.get();  
 candidate.setStatus(status);  
 return true;  
 }else{  
 return false;  
 }  
 @Override  
 public void scoreIncrementor(Long id) {  
 //getting referred candidate from repo  
 ReferredCandidate referredCandidate = referredCandidateRepo.getReferenceById(id);  
 Long referredByID = referredCandidate.getReferredBy();  
 Employee employee = employeeRepo.getReferenceById(referredByID);  
 int current\_score = employee.getScore();  
 LOGGER.info("User's current score is: "+String.*valueOf*(current\_score));  
 current\_score++;  
 employee.setScore(current\_score);  
 LOGGER.info("User's score has been updated, current User Score :"+current\_score);  
  
 }  
  
 @Override  
 public List<List<String>> getLeaderBoardList() {  
 List<List<String>> res = new ArrayList<>();  
 //get all employee details with descending order of score  
 List<Employee> employeeList = employeeRepo.findAll();  
 //sort them based on score  
 employeeList.sort(Comparator.*comparingInt*(Employee::getScore).reversed());  
 //create a List of string with only Lname,Fname and score  
 for (Employee e: employeeList){  
 String employeeName = e.getLName() + "," +e.getFName();  
 String score = String.valueOf(e.getScore());  
 String empId = String.*valueOf*(e.getEmployeeID());  
 List<String> employeeDetailList = new ArrayList<>();  
 employeeDetailList.add(empId);  
 employeeDetailList.add(employeeName);  
 employeeDetailList.add(score);  
 res.add(employeeDetailList);  
 employeeDetailList.clear();  
 }  
return res;  
 }

**Description**

* getAllReferredCandidates() is used to fetch all the referred candidates that are referred by the employees. Those data are viewed by the HR and used for further processes like selecting to next rounds depending on the candidates referred and other factors.
* By using updateStatus() the status of the job can be modified. This method is used to update the status of the referred candidate.
* By using scoreIncrementor method, it will increment the score of an employee who referred a candidate, by updating it by 1.
* getLeaderBoardList will give a leaderboard list, by ranking the employees based on their scores in descending order.

**Job Service**

package com.innovators.jobreferralportal.Service;  
  
  
import com.innovators.jobreferralportal.entity.Job;  
import com.innovators.jobreferralportal.repository.JobRepo;  
import org.apache.poi.ss.usermodel.Row;  
import org.apache.poi.ss.usermodel.Sheet;  
import org.apache.poi.ss.usermodel.Workbook;  
import org.apache.poi.ss.usermodel.WorkbookFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.dao.DataAccessException;  
import org.springframework.stereotype.Service;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Iterator;  
import java.util.List;  
  
@Service  
public class JobServiceImpl implements JobService {  
  
 @Autowired  
 private JobRepo jobRepo;  
  
 @Override  
 public Job addJob(Job jobListing) {  
 return jobRepo.save(jobListing);  
 }  
  
 @Override  
 public boolean updateJob(Long id, Job updatedJob) {  
 //get Job based on id  
 Job jobForUpdate = jobRepo.getReferenceById(id);  
  
 jobForUpdate.setPositionName(updatedJob.getPositionName());  
 jobForUpdate.setJobDescription(updatedJob.getJobDescription());  
 jobForUpdate.setDepartmentName(updatedJob.getDepartmentName());  
 jobForUpdate.setNumberOfOpenPositions(updatedJob.getNumberOfOpenPositions());  
  
 try{  
 jobRepo.save(jobForUpdate);  
 return true;  
 }catch(DataAccessException e){  
 return false;  
 }  
  
 @Override  
 public void deleteJob(Long id) {  
 jobRepo.deleteById(id);  
 Job jobCheck = jobRepo.getReferenceById(id);  
  
 }

@Override  
public List<Job> searchJob(String positionName) {  
 return jobRepo.findAllByPositionNameContaining(positionName);  
}  
  
private List<Job> parseExcelFile(MultipartFile file) throws IOException {  
 List<Job> jobList = new ArrayList<>();  
 Workbook workbook = WorkbookFactory.*create*(file.getInputStream());  
 Sheet sheet = workbook.getSheetAt(0); // Make sure that the data is in the first sheet  
 Iterator<Row> rows = sheet.iterator();  
  
 // Skip the header row  
 if (rows.hasNext()) rows.next();  
  
 while (rows.hasNext()) {  
 Row currentRow = rows.next();  
 Job job = new Job();  
  
 job.setPositionName(currentRow.getCell(0).getStringCellValue());  
 job.setJobDescription(currentRow.getCell(1).getStringCellValue());  
 job.setDepartmentName(currentRow.getCell(2).getStringCellValue());  
 job.setNumberOfOpenPositions(currentRow.getCell(3).getStringCellValue());  
  
 jobList.add(job);  
 }  
  
 workbook.close();  
 return jobList;

}

}

**Description**

* addJob() is used to insert new jobs in the referral portal ensuring that relevant details like position name, description, and location. It is important to maintain up to date listings making the employees view and apply for open positions.
* updateJob() is used to modify the details of an existing job posting in the referral portal such as updating the description or application deadline. It makes sure that outdated information is corrected
* deleteJob() is a function that removes specific job postings from the referral portal’s database, ensuring that irrelevant or filled positions are no longer visible to users.
* searchJob() is the method to search for jobs with position names that contains a sspecific keyword.
* parseExcelFile () will read the job details from an excel file and creates a list of job from the data.

**FrontEnd**

**Source code for employee dashboard**

import { Component, OnInit } from '@angular/core';  
import { EmployeeService } from '../../Services/services/employee.service';  
import { Router } from '@angular/router';  
import { CommonModule } from '@angular/common';  
@Component({  
 selector: 'app-employee-dashboard',  
 standalone: *true*,  
 imports: [CommonModule],  
 templateUrl: './employee-dashboard.component.html',  
 styleUrls: ['./employee-dashboard.component.css']  
})  
export class EmployeeDashboardComponent implements OnInit {  
 jobs: any[] = [];  
  
 constructor(private employeeService: EmployeeService, private router: Router) {}  
  
 ngOnInit(): void {  
 this.loadJobs();  
 }  
  
 loadJobs(): void {  
 this.employeeService.getAllJobs().subscribe({  
 next: (data) => {  
 this.jobs = data;  
   
 },  
 error: (error) => {  
 console.error('Error fetching jobs:', error);  
 },  
 complete: () => {  
 console.info('Job fetching complete');  
 }  
 });  
 }  
  
 referCandidate(jobId: number): void {  
 this.router.navigate(['/refer', jobId]);  
 }  
searchJobs(): void {  
 if (this.searchQuery) {  
 this.jobService.searchJobs(this.searchQuery).subscribe((data: any[]) => {  
 this.jobs = data;  
 });  
 } else {  
 this.loadJobs();  
 }  
 }  
  
}

**Description**

* loadJobs() methods will interact with the employeeService to get all the jobs which re in the backend.
* referCandidate() is used to navigate to the refer route with the selected jobId as a parameter.
* searchJobs() is used to search of the jobs based on position name

**Source Code for HR dashboard**

import { Component, OnInit } from '@angular/core';  
import { HrService } from '../../Services/services/hr.service';  
import { CommonModule } from '@angular/common';  
import { FormsModule } from '@angular/forms';   
import { JobService } from '../../Services/services/job.service';  
@Component({  
 selector: 'app-hr-dashboard',  
 templateUrl: './hr-dashboard.component.html',  
 styleUrls: ['./hr-dashboard.component.css'],  
 standalone: *true*,  
 imports: [CommonModule, FormsModule]  
})  
export class HrDashboardComponent implements OnInit {  
 jobs: any[] = [];  
 leaderBoard: any[] = [];  
 showJobModal = *false*;  
 selectedJob: any = *null*;  
 newJob = { jobId: *0*, positionName: '', jobDescription: '' ,departmentName:'',numberOfOpenPositions:*0* };  
 alertMessage: string = '';  
 showConfirmationModal = *false*;  
 jobToDeleteId: number | null = *null*;  
 constructor(private hrService: HrService , private jobService: JobService) {}  
  
 ngOnInit(): void {  
 this.loadJobs();  
 }  
  
 loadJobs(): void {  
 this.hrService.getAllJobs().subscribe((data) => {  
 this.jobs = data;  
 });  
 }  
  
  
 editJob(job: any): void {  
 this.selectedJob = job;  
 this.newJob = { ...job };  
 this.showJobModal = *true*;  
 }  
 openAddJobModal() {  
 this.showJobModal = *true*;   
 this.resetNewJob();  
 }  
 resetNewJob():void{  
 this.newJob = {  
 jobId: *0*,  
 positionName: '',  
 jobDescription: '',  
 departmentName: '',  
 numberOfOpenPositions: *0* }  
 }  
  
 saveJob(): void {  
 if (this.selectedJob) {  
 this.jobService.updateJob(this.selectedJob.jobId, this.newJob).subscribe(  
 (response) => {  
 console.log('Backend response:', response);  
 const index = this.jobs.findIndex(job => job.jobId === this.selectedJob.jobId);  
 if (index !== -*1*) {  
 this.jobs[index] = { ...this.newJob };  
 }  
 this.closeJobModal();  
 },  
 (error) => {  
 console.error('Error updating job in backend:', error);  
 }  
 );  
 } else {  
 this.jobService.addJob(this.newJob).subscribe(  
 (response) => {  
 console.log('Backend response:', response);   
 this.jobs.push(response);  
 this.closeJobModal();  
 },  
 (error) => {  
 console.error('Error adding job:', error);  
 }  
 );  
 }  
 }  
   
  
   
  
 deleteJob(jobId: number): void {  
 this.jobService.deleteJob(jobId).subscribe(() => {  
 this.jobToDeleteId = jobId;  
 this.showConfirmationModal = *true*;  
 });  
 }  
 confirmDelete() {  
 if (this.jobToDeleteId) {  
 this.jobService.deleteJob(this.jobToDeleteId).subscribe(() => {  
 this.loadJobs();   
 this.closeConfirmationModal();   
 });  
 }  
 }  
  
 cancelDelete() {  
 this.closeConfirmationModal();   
 }  
  
 closeConfirmationModal() {  
 this.showConfirmationModal = *false*;  
 this.jobToDeleteId = *null*;   
 }  
  
 closeJobModal(): void {  
 this.showJobModal = *false*;  
 }  
}

searchJobs(): void {  
 if (this.searchQuery) {  
 this.jobService.hrSearchJobs(this.searchQuery).subscribe((data: any[]) => {  
 this.jobs = data;  
 });  
 } else {  
 this.loadJobs();  
 }  
}

**Description**

* loadJobs() is used to fetch all the jobs and also it updates the jobs with the fetched data.
* editJob() will edit the particular job and open the job modal. It will copy the job details into newJob for editing.
* openAddJobModal() will open the job modal and it will reset the new job form.
* resetNewJob() will reset the new Job object to default values.
* saveJob() uses jobService to update the job or to add a new job.
* deleteJob() will delete the job and show us the confirmation modal.
* confrimDelete() means that it needs confirmation if the user confirms then the job is deleted by using the job service.
* cancelDelete() will close the confirmation modal without deleting the job.
* closeConfirmationModal() will reset the confirmation
* closeJobModal() will hide the job modal
* searchJobs() is used to search for jobs by the position name

**Source Code for Login Component**

import { Component, OnInit } from '@angular/core';  
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms'; // Import ReactiveFormsModule  
import { HttpClient } from '@angular/common/http';   
import { Router } from '@angular/router';  
import { AuthService } from '../../Services/services/auth.service';  
import { SessionService } from '../../Services/services/session.service';  
import { CommonModule } from '@angular/common';  
  
@Component({  
 selector: 'app-login',  
 standalone: *true*,   
 imports: [ReactiveFormsModule,CommonModule],   
 templateUrl: './login.component.html',  
 styleUrls:['./login.component.css']  
})  
export class LoginComponent implements OnInit {  
 loginForm: FormGroup;  
 errorMessage: string | null = *null*;  
  
 constructor(  
 private fb: FormBuilder,  
 private authService: AuthService,   
 private router: Router,  
 private sessionService: SessionService  
 ) {  
 this.loginForm = this.fb.group({  
 username: ['', Validators.required],  
 password: ['', Validators.required]  
 });  
 }  
  
 ngOnInit(): void {}  
  
 onSubmit() {  
 if (this.loginForm.valid) {  
 const { username, password } = this.loginForm.value;  
 this.authService.login(username, password).subscribe({  
 next: response => {  
 const employeeType = this.sessionService.getSessionData('employeeType');  
 if (employeeType === 'HR') {  
 this.router.navigate(['/hr-dashboard']);   
 } else {  
 this.router.navigate(['/employee-dashboard']);  
 }  
 },  
 error: error => {  
 console.error(error);  
 this.errorMessage = 'Invalid login credentials';   
 }  
 });  
 }  
 }  
}

**Description**

* onSubmit() checks if the form is valid or not whether both the fields username and password is filled or not.
* The entered usernameandpassword are retrieved from the form.
* The login**()** method from AuthService sends the username and password to the backend.
* Retrieves the employeeType from the session data.
* If the user is HR then redirect to /hr-dashboard.
* If the user is not HR then redirect to **/**employee**-**dashboard.

**Source Code for refer candidate**

import { Component, OnInit } from '@angular/core';  
import { ActivatedRoute, Router } from '@angular/router';  
import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';  
import { EmployeeService } from '../../Services/services/employee.service';  
import { CommonModule } from '@angular/common';  
  
@Component({  
 selector: 'app-refer-candidate',  
 standalone: *true*,  
 imports: [ReactiveFormsModule,CommonModule],  
 templateUrl: './refer-candidate.component.html',  
 styleUrls: ['./refer-candidate.component.css']  
})  
export class ReferCandidateComponent implements OnInit {  
 jobId!: number;  
 referForm: FormGroup;  
 resumeFile: File | null = *null*;  
  
 constructor(  
 private route: ActivatedRoute,  
 private fb: FormBuilder,  
 private employeeService: EmployeeService,  
 private router: Router  
 ) {  
 this.referForm = this.fb.group({  
 fName: ['', Validators.required],  
 lName: ['', Validators.required],  
 yearsOfExp: ['', Validators.required]  
 });  
 }  
  
 ngOnInit(): void {  
 this.jobId = Number(this.route.snapshot.paramMap.get('jobId'));  
 }  
  
 onFileChange(event: any): void {  
 this.resumeFile = event.target.files[*0*];  
 }  
  
 onSubmit(): void {  
 if (this.referForm.valid && this.resumeFile) {  
 const { fName, lName, yearsOfExp } = this.referForm.value;  
 const referredBy = parseInt(sessionStorage.getItem('employeeID') || '0', *10*);  
 this.employeeService.referCandidate(this.jobId, fName, lName, yearsOfExp, referredBy, this.resumeFile)  
 .subscribe({  
 next: (response) => {  
 if (response.status === *200*) {  
 alert(response.body); // Show the response message  
 this.router.navigate(['/employee-dashboard']);  
 } else {  
 console.error('Unexpected status code:', response.status);  
 }  
 },  
 error: (error) => {  
 console.error('Error referring candidate:', error);  
 }  
 });  
 }  
}  
}

**Description**

* onSubmit() methods will make sure that the form is valid and a resume file has been uploaded.
* It will retrieve the first name, last name, and years of experience from the form.
* It gets the current employee’s ID from the session storage to track who made the referral.
* It calls the referCandidate**()** method in EmployeeService with the form values and resume file.