**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

**Business Scenario:**

Add pagination and sorting capabilities to your employee search functionality.

**Instructions:**

1. **Pagination:**
   * Implement pagination for the employee list using **Page** and **Pageable**.
2. **Sorting:**
   * Add sorting functionality to your queries.
   * Combine pagination and sorting in your search endpoint.

**EmployeeController.java🡪**

package com.code.employee.controllers;

import java.util.List;

import java.util.Optional;

import org.springframework.data.domain.Pageable;

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

//import org.springframework.http.ResponseEntity;

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 com.code.employee.entity.Department;

import com.code.employee.entity.Employee;

import com.code.employee.entity.EmployeeDTO;

import com.code.employee.repository.DepartmentRepository;

import com.code.employee.repository.EmployeeRepository;

@RestController

    @RequestMapping("/api/employees")

    public class EmployeeController {

        @Autowired

        private EmployeeRepository employeeRepository;

        @Autowired

        private DepartmentRepository  departmentRepository;

        // Create a new Employee

        @PostMapping(value="add")

        public Employee createEmployee(@RequestBody EmployeeDTO employeeDTO) {

            //System.out.println("name"+employeeDTO.getName());

            //System.out.println("name"+employeeDTO.getDeptid());

            //get the details of the department by id

            Department department= departmentRepository.findById(employeeDTO.getDeptid()).get();

            //create a new object of employee

            Employee employee=new Employee();

            employee.setDepartment(department);

            employee.setEmail(employeeDTO.getEmail());

            employee.setName(employeeDTO.getName());

//          employee.setSalary(employeeDTO.getSalary());

            //save the object

            return employeeRepository.save(employee);

        }

        // Get all Employees

        @GetMapping(value="/")

        public List<Employee> getAllEmployees() {

            return employeeRepository.findAll();

        }

        // Get a single Employee by ID

        @GetMapping(value="{id}")

        public Employee getEmployeeById(@PathVariable int id) {

            Employee employee = employeeRepository.findById(id).get();

            return employee;

        }

        // Update an Employee

        @PutMapping(value="/edit/{id}")

        //@path variable used to get the variable passed from url

        //http://localhost:8185/api/employee/1

        //@PathVariable int id returns 1

        public Employee updateEmployee(@PathVariable int id, @RequestBody EmployeeDTO employeeDet) {

            //returns the object of Optional class

            Optional<Employee> optionalEmployee = employeeRepository.findById(id);

            //checking any object is present or not

            if (optionalEmployee.isPresent()) {

                //setting the new value

                Employee employee = optionalEmployee.get();

                employee.setName(employeeDet.getName());

                employee.setEmail(employeeDet.getEmail());

                Department department= departmentRepository.findById(employeeDet.getDeptid()).get();

                employee.setDepartment(department);

                //saving the object

                Employee updatedEmployee = employeeRepository.save(employee);

                //returning the object

                return updatedEmployee;

            } else {

                //returning null if not object found

                return null;

            }

        }

        // Delete an Employee

        @DeleteMapping(value="delete/{id}")

        public String deleteEmployee(@PathVariable int id) {

            Optional<Employee> optionalEmployee = employeeRepository.findById(id);

            if (optionalEmployee.isPresent()) {

                employeeRepository.delete(optionalEmployee.get());

                return "Employee with ID "+id+" is deleted Successfully";

            } else {

                return "Employee with ID "+id+" is not found";

            }

        }

        //named query

        @GetMapping(value="email/{email}")

        public Employee findByEmail(@PathVariable String email)

        {

            return employeeRepository.findByEmailNamed(email);

        }

        @GetMapping(value="department/{id}")

        public Employee findByDepartmentId(@PathVariable int id)

        {

            return employeeRepository.findByDepartmentIdNamed(id);

        }

        // Get all employees with pagination and sorting

        @GetMapping("page")

        public Page<Employee> getAllEmployees(

                @RequestParam(defaultValue = "0") int page,

                @RequestParam(defaultValue = "10") int size,

                @RequestParam(defaultValue = "id") String sortBy){

            Pageable pageable = PageRequest.of(page, size, Sort.by(sortBy));

            return employeeRepository.findAll(pageable);

        }

     // Get all employees by department with pagination and sorting

        @GetMapping("department/{departmentId}")

        public Page<Employee> getAllEmployees(

                @PathVariable int departmentId,

                @RequestParam(defaultValue = "0") int page,

                @RequestParam(defaultValue = "10") int size,

                @RequestParam(defaultValue = "id") String sortBy){

            Pageable pageable = PageRequest.of(page,  size, Sort.by(sortBy));

            return employeeRepository.findByDepartmentId(departmentId, pageable);

        }

    }

**EmployeeRepository.java 🡪**

package com.code.employee.repository;

import java.util.List;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Page;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import com.code.employee.entity.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer>{

    // Using the named query defined in Employee entity

    @Query(name = "Employee.findByEmail")

    Employee findByEmailNamed(@Param("email") String email);

    // Using the named query defined in Employee entity

    @Query(name = "Employee.findByEmail")

    Employee findByDepartmentIdNamed(@Param("email") int departmentId);

    // Derived query methods

    List<Employee> findByName(String name);

    List<Employee> findByDepartmentId(int DepartmentId);

    List<Employee> findByEmail(String email);

    //Find all employees with pagination and sorting

    Page<Employee> findAll(Pageable pageable);

    //Find employees by department with pagination and sorting

    Page<Employee> findByDepartmentId(int departmentId, Pageable pageable);

}