Spring Data Core JPA & Hibernate

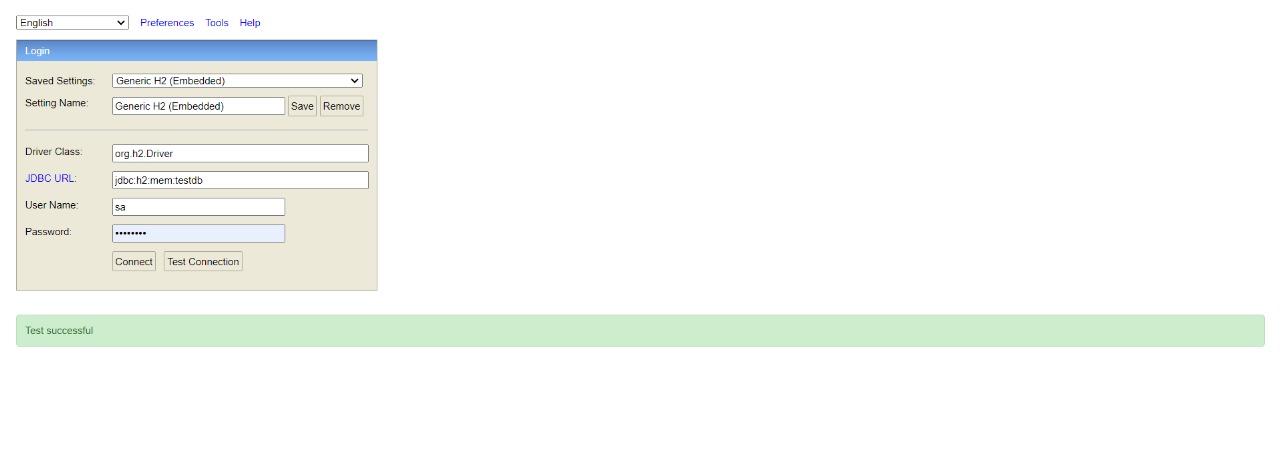
**Exercise 1: Employee Management System - Overview and Setup**

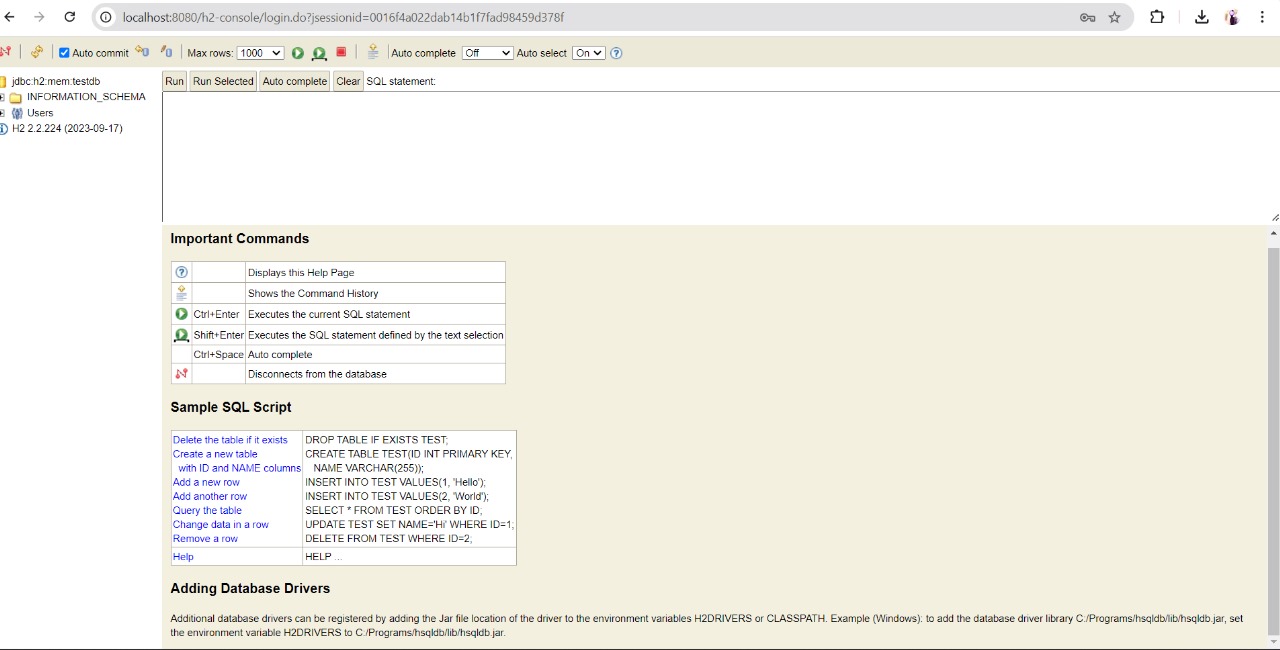
**Resources>Application.properties:**

spring.application.name=EmployeeManagementSystem  
spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=password  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console

**Output:**

Open <http://localhost:8080/h2-console>.

****



**Exercise 2: Employee Management System - Creating Entities**

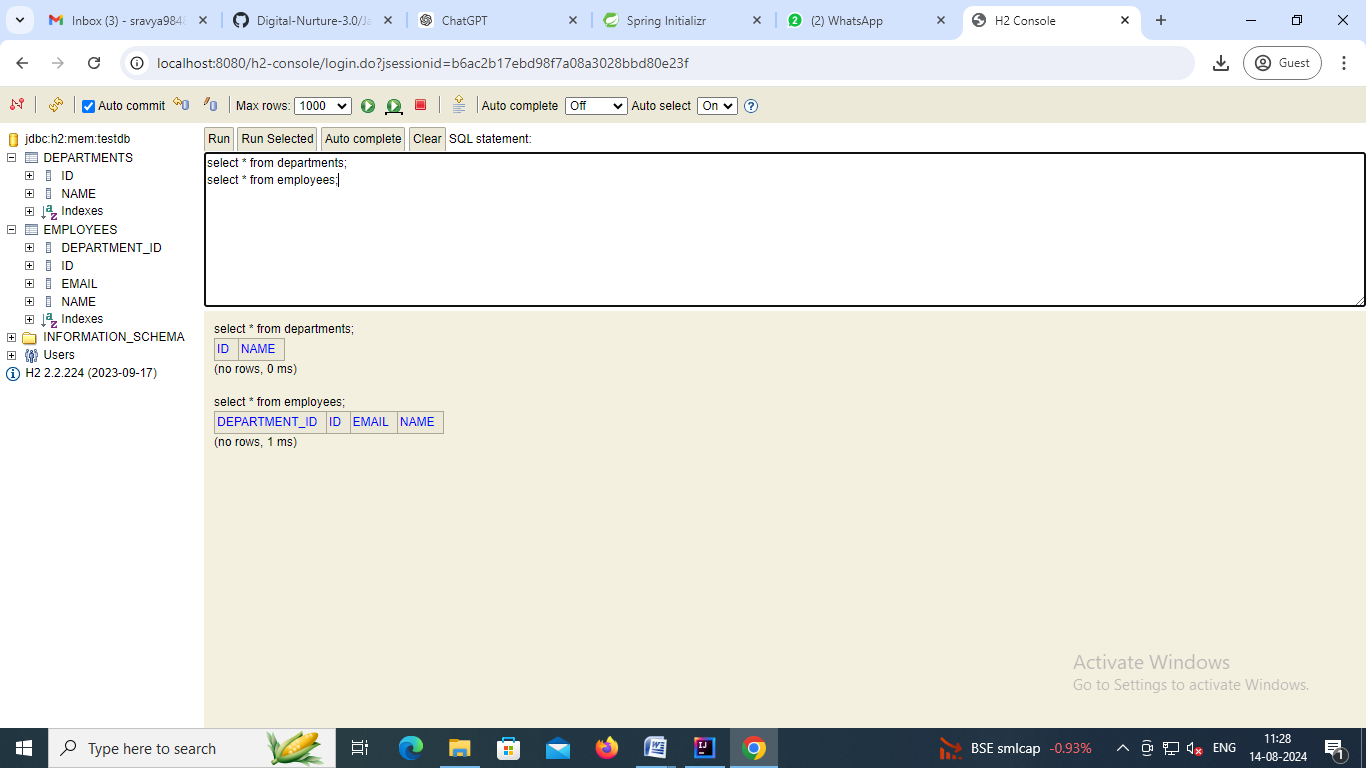
Deparment.java:

package com.example.EmployeeManagementSystem.model;  
  
import jakarta.persistence.\*;  
import java.util.HashSet;  
import java.util.Set;  
  
@Entity  
@Table(name = "departments")  
public class Department {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(name = "name", nullable = false, unique = true)  
 private String name;  
  
 @OneToMany(mappedBy = "department", cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*)  
 private Set<Employee> employees = new HashSet<>();  
  
 *// Constructors, Getters, Setters* public Department() {}  
  
 public Department(String name) {  
 this.name = name;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Set<Employee> getEmployees() {  
 return employees;  
 }  
  
 public void setEmployees(Set<Employee> employees) {  
 this.employees = employees;  
 }  
  
 public void addEmployee(Employee employee) {  
 employees.add(employee);  
 employee.setDepartment(this);  
 }  
  
 public void removeEmployee(Employee employee) {  
 employees.remove(employee);  
 employee.setDepartment(null);  
 }  
}

Employee.java:

package com.example.EmployeeManagementSystem.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "employees")  
public class Employee {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(name = "name", nullable = false)  
 private String name;  
  
 @Column(name = "email", nullable = false, unique = true)  
 private String email;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "department\_id", nullable = false)  
 private Department department;  
  
 *// Constructors, Getters, Setters* public Employee() {}  
  
 public Employee(String name, String email, Department department) {  
 this.name = name;  
 this.email = email;  
 this.department = department;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
}

OUTPUT:



**Exercise 3: Employee Management System - Creating Repositories**

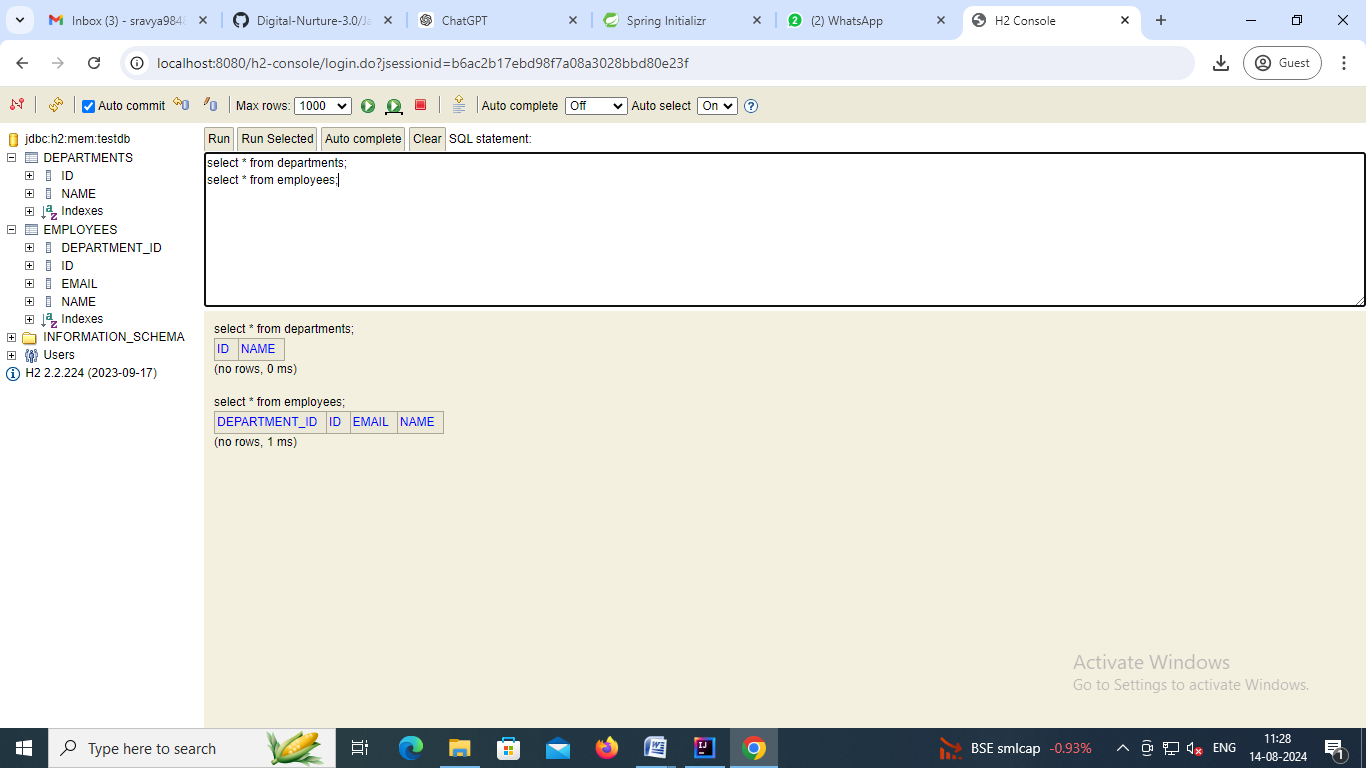
EmployeeRepository.java:

package com.example.EmployeeManagementSystem.repository;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
  
 *// Derived query method to find employees by name* List<Employee> findByName(String name);  
  
 *// Derived query method to find employees by email* Employee findByEmail(String email);  
  
 *// Derived query method to find employees by department id* List<Employee> findByDepartmentId(Long departmentId);  
}

DepartmentRepository.java:

package com.example.EmployeeManagementSystem.repository;  
  
import com.example.EmployeeManagementSystem.entity.Department;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface DepartmentRepository extends JpaRepository<Department, Long> {  
  
 *// Derived query method to find departments by name* Department findByName(String name);  
}

OUTPUT:



**Exercise 4: Employee Management System - Implementing CRUD Operations**

EmployeeController.java:

package com.example.EmployeeManagementSystem.controller;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import com.example.EmployeeManagementSystem.repository.EmployeeRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/employees")  
public class EmployeeController {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 *// Create a new employee* @PostMapping  
 public Employee createEmployee(@RequestBody Employee employee) {  
 return employeeRepository.save(employee);  
 }  
  
 *// Get all employees* @GetMapping  
 public List<Employee> getAllEmployees() {  
 return employeeRepository.findAll();  
 }  
  
 *// Get an employee by ID* @GetMapping("/{id}")  
 public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 return ResponseEntity.*ok*(employee.get());  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Update an employee by ID* @PutMapping("/{id}")  
 public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 Employee existingEmployee = employee.get();  
 existingEmployee.setName(updatedEmployee.getName());  
 existingEmployee.setEmail(updatedEmployee.getEmail());  
 existingEmployee.setDepartment(updatedEmployee.getDepartment());  
 employeeRepository.save(existingEmployee);  
 return ResponseEntity.*ok*(existingEmployee);  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Delete an employee by ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 employeeRepository.delete(employee.get());  
 return ResponseEntity.*noContent*().build();  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
}

DepartmentController.java:

package com.example.EmployeeManagementSystem.controller;  
  
import com.example.EmployeeManagementSystem.entity.Department;  
import com.example.EmployeeManagementSystem.repository.DepartmentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/departments")  
public class DepartmentController {  
  
 @Autowired  
 private DepartmentRepository departmentRepository;  
  
 *// Create a new department* @PostMapping  
 public Department createDepartment(@RequestBody Department department) {  
 return departmentRepository.save(department);  
 }  
  
 *// Get all departments* @GetMapping  
 public List<Department> getAllDepartments() {  
 return departmentRepository.findAll();  
 }  
  
 *// Get a department by ID* @GetMapping("/{id}")  
 public ResponseEntity<Department> getDepartmentById(@PathVariable Long id) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 return ResponseEntity.*ok*(department.get());  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Update a department by ID* @PutMapping("/{id}")  
 public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department updatedDepartment) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 Department existingDepartment = department.get();  
 existingDepartment.setName(updatedDepartment.getName());  
 departmentRepository.save(existingDepartment);  
 return ResponseEntity.*ok*(existingDepartment);  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Delete a department by ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 departmentRepository.delete(department.get());  
 return ResponseEntity.*noContent*().build();  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
}

OUTPUT:



**Exercise 5: Employee Management System - Defining Query Methods**

Employee.java:

package com.example.EmployeeManagementSystem.entity;  
  
import jakarta.persistence.\*;  
import lombok.Data;  
  
@Data  
@Entity  
@Table(name = "employees")  
@NamedQueries({  
 @NamedQuery(  
 name = "Employee.findByDepartmentNameNamed",  
 query = "SELECT e FROM Employee e WHERE e.department.name = :departmentName"  
 ),  
 @NamedQuery(  
 name = "Employee.findByEmailNamed",  
 query = "SELECT e FROM Employee e WHERE e.email = :email"  
 )  
})  
public class Employee {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String name;  
  
 private String email;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "department\_id", nullable = false)  
 private Department department;  
}

EmployeeController.java:

package com.example.EmployeeManagementSystem.controller;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import com.example.EmployeeManagementSystem.repository.EmployeeRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/employees")  
public class EmployeeController {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 *// Create a new employee* @PostMapping  
 public Employee createEmployee(@RequestBody Employee employee) {  
 return employeeRepository.save(employee);  
 }  
  
 *// Get all employees* @GetMapping  
 public List<Employee> getAllEmployees() {  
 return employeeRepository.findAll();  
 }  
  
 *// Get an employee by ID* @GetMapping("/{id}")  
 public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 return ResponseEntity.*ok*(employee.get());  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Update an employee by ID* @PutMapping("/{id}")  
 public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 Employee existingEmployee = employee.get();  
 existingEmployee.setName(updatedEmployee.getName());  
 existingEmployee.setEmail(updatedEmployee.getEmail());  
 existingEmployee.setDepartment(updatedEmployee.getDepartment());  
 employeeRepository.save(existingEmployee);  
 return ResponseEntity.*ok*(existingEmployee);  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Delete an employee by ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 employeeRepository.delete(employee.get());  
 return ResponseEntity.*noContent*().build();  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Additional endpoints to use custom queries  
  
 // Find employees by name containing a specific substring* @GetMapping("/search/by-name")  
 public List<Employee> findByName(@RequestParam String name) {  
 return employeeRepository.findByNameContainingIgnoreCase(name);  
 }  
  
 *// Find employees by department name using method naming convention* @GetMapping("/search/by-department")  
 public List<Employee> findByDepartmentName(@RequestParam String departmentName) {  
 return employeeRepository.findByDepartmentName(departmentName);  
 }  
  
 *// Find employees by department name using custom JPQL query* @GetMapping("/search/by-department-jpql")  
 public List<Employee> findByDepartmentNameJPQL(@RequestParam String departmentName) {  
 return employeeRepository.findByDepartmentNameJPQL(departmentName);  
 }  
  
 *// Find employees by name using native SQL query* @GetMapping("/search/by-name-native")  
 public List<Employee> findByNameNativeSQL(@RequestParam String name) {  
 return employeeRepository.findByNameNativeSQL(name);  
 }  
}

DepartmentController.java:

package com.example.EmployeeManagementSystem.controller;  
  
import com.example.EmployeeManagementSystem.entity.Department;  
import com.example.EmployeeManagementSystem.repository.DepartmentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/departments")  
public class DepartmentController {  
  
 @Autowired  
 private DepartmentRepository departmentRepository;  
  
 *// Create a new department* @PostMapping  
 public Department createDepartment(@RequestBody Department department) {  
 return departmentRepository.save(department);  
 }  
  
 *// Get all departments* @GetMapping  
 public List<Department> getAllDepartments() {  
 return departmentRepository.findAll();  
 }  
  
 *// Get a department by ID* @GetMapping("/{id}")  
 public ResponseEntity<Department> getDepartmentById(@PathVariable Long id) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 return ResponseEntity.*ok*(department.get());  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Update a department by ID* @PutMapping("/{id}")  
 public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department updatedDepartment) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 Department existingDepartment = department.get();  
 existingDepartment.setName(updatedDepartment.getName());  
 departmentRepository.save(existingDepartment);  
 return ResponseEntity.*ok*(existingDepartment);  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Delete a department by ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 departmentRepository.delete(department.get());  
 return ResponseEntity.*noContent*().build();  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
}

EmployeeRepository:

package com.example.EmployeeManagementSystem.repository;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
  
 *// Find employees by name containing a specific substring (case-insensitive)* List<Employee> findByNameContainingIgnoreCase(String name);  
  
 *// Find employees by department name using method naming convention* List<Employee> findByDepartmentName(String departmentName);  
  
 *// Find employees whose email starts with a specific prefix* List<Employee> findByEmailStartingWith(String prefix);  
  
 *// Custom JPQL query to find employees by department name* @Query("SELECT e FROM Employee e WHERE e.department.name = :departmentName")  
 List<Employee> findByDepartmentNameJPQL(@Param("departmentName") String departmentName);  
  
 *// Custom native SQL query to find employees by name* @Query(value = "SELECT \* FROM employees WHERE name LIKE %:name%", nativeQuery = true)  
 List<Employee> findByNameNativeSQL(@Param("name") String name);  
  
 *// Named Queries* List<Employee> findByDepartmentNameNamed(@Param("departmentName") String departmentName);  
  
 List<Employee> findByEmailNamed(@Param("email") String email);  
}

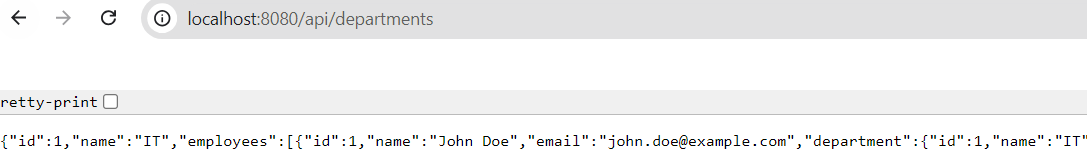
Department.java:

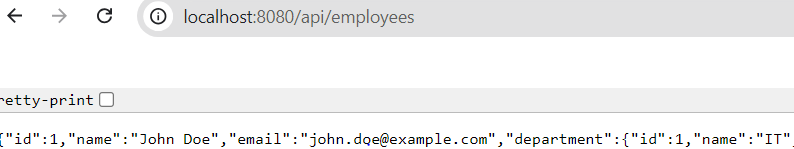
package com.example.EmployeeManagementSystem.entity;  
  
import jakarta.persistence.\*;  
import lombok.Data;  
import java.util.List;  
  
@Data  
@Entity  
@Table(name = "departments")  
public class Department {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String name;  
  
 @OneToMany(mappedBy = "department", cascade = CascadeType.*ALL*, orphanRemoval = true)  
 private List<Employee> employees;  
}

DepartmentRepository.java:

package com.example.EmployeeManagementSystem.repository;  
  
import com.example.EmployeeManagementSystem.entity.Department;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface DepartmentRepository extends JpaRepository<Department, Long> {  
  
 *// Add custom queries if needed*}

OUTPUT:





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

EmployeeReposistory.java:

package com.example.EmployeeManagementSystem.repository;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
 Page<Employee> findByNameContaining(String name, Pageable pageable);  
}

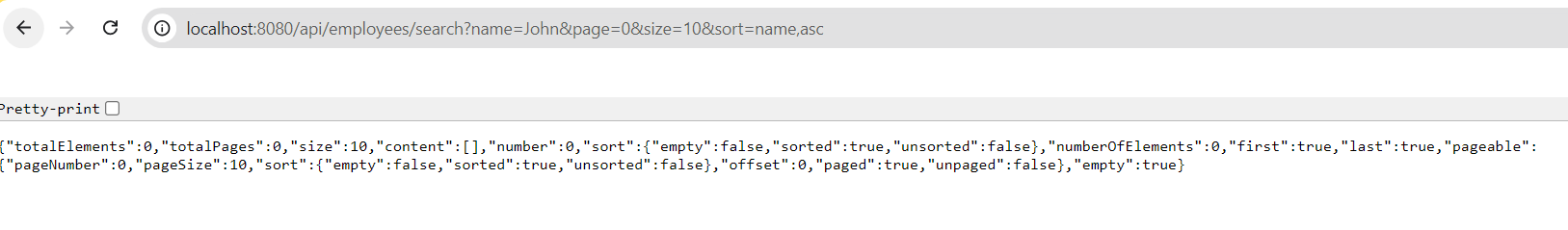
EmployeeService.java:

package com.example.EmployeeManagementSystem.service;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import com.example.EmployeeManagementSystem.repository.EmployeeRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.stereotype.Service;  
  
@Service  
public class EmployeeService {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 public Page<Employee> searchEmployees(String name, Pageable pageable) {  
 return employeeRepository.findByNameContaining(name, pageable);  
 }  
}

EmployeeController.java:

package com.example.EmployeeManagementSystem.controller;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import com.example.EmployeeManagementSystem.service.EmployeeService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.domain.Sort;  
import org.springframework.web.bind.annotation.\*;  
  
@RestController  
@RequestMapping("/api/employees")  
public class EmployeeController {  
  
 @Autowired  
 private EmployeeService employeeService; *// Proper injection* @GetMapping("/search")  
 public Page<Employee> searchEmployees(  
 @RequestParam(value = "name", defaultValue = "") String name,  
 @RequestParam(value = "page", defaultValue = "0") int page,  
 @RequestParam(value = "size", defaultValue = "10") int size,  
 @RequestParam(value = "sort", defaultValue = "id,asc") String sort) {  
  
 String[] sortParams = sort.split(",");  
 Pageable pageable = PageRequest.*of*(page, size, Sort.*by*(Sort.Direction.*fromString*(sortParams[1]), sortParams[0]));  
  
 *// Instance method call, correct usage* return employeeService.searchEmployees(name, pageable);  
 }  
}

OUTPUT:



**Exercise 7: Employee Management System - Enabling Entity Auditing**

SpringSecurityAuduiting.java:

package com.example.EmployeeManagementSystem.security;  
  
import org.springframework.data.domain.AuditorAware;  
import org.springframework.security.core.context.SecurityContextHolder;  
import java.util.Optional;  
  
public class SpringSecurityAuditorAware implements AuditorAware<String> {  
 @Override  
 public Optional<String> getCurrentAuditor() {  
 *// Return the current authenticated user. You might need to adjust this to fit your security setup.* return Optional.*ofNullable*(SecurityContextHolder.*getContext*().getAuthentication())  
 .map(auth -> auth.getName());  
 }  
}

**Auditing Config.java:**

package com.example.EmployeeManagementSystem.config;  
  
import com.example.EmployeeManagementSystem.security.SpringSecurityAuditorAware;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;  
import org.springframework.data.domain.AuditorAware;  
  
@Configuration  
@EnableJpaAuditing(auditorAwareRef = "auditorProvider")  
public class AuditingConfig {  
  
 @Bean  
 public AuditorAware<String> auditorProvider() {  
 return new SpringSecurityAuditorAware();  
 }  
}

**Auditiorawareimpl.java:**

package com.example.EmployeeManagementSystem.config;  
  
import org.springframework.data.domain.AuditorAware;  
import org.springframework.stereotype.Component;  
  
import java.util.Optional;  
  
@Component  
public class AuditorAwareImpl implements AuditorAware<String> {  
  
 @Override  
 public Optional<String> getCurrentAuditor() {  
 *// Return the username of the currently logged-in user or a default user* return Optional.*of*("system");  
 }  
}

**Exercise 9: Employee Management System - Customizing Data Source Configuration**

**DepartmentDTO**

package com.example.EmployeeManagementSystem.dto;  
  
public class DepartmentDTO {  
 private String name;  
  
 public DepartmentDTO(String name) {  
 this.name = name;  
 }  
  
 *// Getters and Setters*}

**EmployeeDTO:**

package com.example.EmployeeManagementSystem.dto;  
  
import com.example.EmployeeManagementSystem.entity.Department;  
  
public class EmployeeDTO {  
 private String firstName;  
 private String lastName;  
 private String email;  
 private Department department;  
  
 public EmployeeDTO(String firstName, String lastName, String email, Department department) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.email = email;  
 this.department = department;  
 }  
  
 *// Getters and Setters*}

**Exercise 10: Employee Management System - Hibernate-Specific Features**

**Employee.java:**

package com.example.EmployeeManagementSystem.entity;  
  
import javax.persistence.\*;  
import org.hibernate.annotations.Cache;  
import org.hibernate.annotations.CacheConcurrencyStrategy;  
import org.hibernate.annotations.Formula;  
import org.hibernate.annotations.Type;  
  
@Entity  
@Cache(usage = CacheConcurrencyStrategy.*READ\_WRITE*)  
public class Employee {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String name;  
  
 private String position;  
  
 @Type(type = "org.hibernate.type.StringClobType")  
 private String largeTextField;  
  
 @Formula("salary \* 0.1")  
 private Double tax;  
  
 @ManyToOne  
 @JoinColumn(name = "department\_id")  
 private Department department;  
  
 *// Getters and Setters*}

EmployeeService.java:

package com.example.EmployeeManagementSystem.service;  
  
import com.example.EmployeeManagementSystem.entity.Employee;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import org.hibernate.Session;  
import java.util.List;  
  
@Service  
public class EmployeeService {  
  
 @PersistenceContext  
 private EntityManager entityManager;  
  
 @Transactional  
 public void batchInsertEmployees(List<Employee> employees) {  
 Session session = entityManager.unwrap(Session.class);  
 int batchSize = 50;  
 int i = 0;  
  
 for (Employee employee : employees) {  
 session.save(employee);  
 if (++i % batchSize == 0) {  
 *// Flush and clear the session* session.flush();  
 session.clear();  
 }  
 }  
 *// Handle the remaining entities* session.flush();  
 session.clear();  
 }  
}

EmployeeDTO.java:

package com.example.EmployeeManagementSystem.dto;  
import com.example.EmployeeManagementSystem.entity.Department;  
public class EmployeeDTO {  
 private String firstName;  
 private String lastName;  
 private String email;  
 private Department department;  
  
 public EmployeeDTO(String firstName, String lastName, String email, Department department) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.email = email;  
 this.department = department;  
 }  
  
 *// Getters and Setters*}

Department.java:

package com.example.EmployeeManagementSystem.entity;  
  
import java.util.\*;  
import javax.persistence.\*;  
import org.hibernate.annotations.Cache;  
import org.hibernate.annotations.CacheConcurrencyStrategy;  
  
@Entity  
@Cache(usage = CacheConcurrencyStrategy.*READ\_WRITE*)  
public class Department {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String name;  
  
 @OneToMany(mappedBy = "department")  
 private List<Employee> employees;  
  
 *// Getters and Setters*}