**Wk09 Assignment: Hibernate and JPA**

# Problem Statement

• Create a Java Project

• Create Entities and Attributes with Relationships shown in

diagram

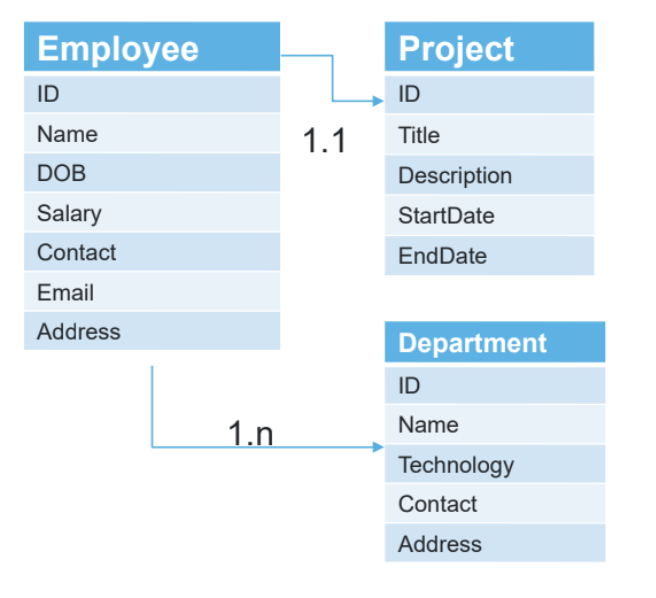
• Create Employee API (which inserts data in all dependent

tables with transaction handling)

• Delete Employee API

• Create Project for Employee

• Update Project and Department for Employee

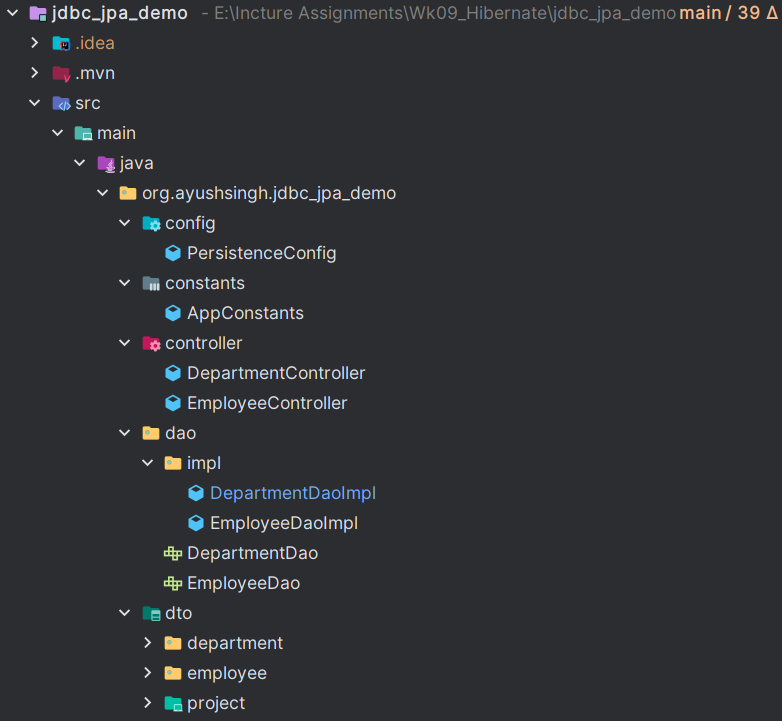


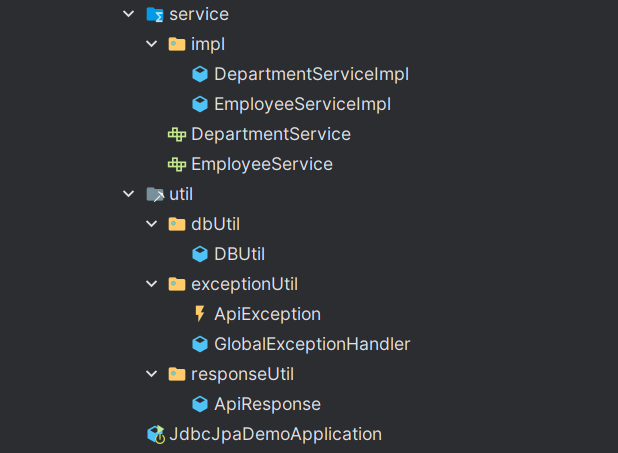
# Project Repository

The complete code can be found here-

<https://github.com/singhayush20/Assignments/tree/main/Wk09_Hibernate>

# Project Structure





## DB Configuration

### PersistenceConfig

package org.ayushsingh.jdbc\_jpa\_demo.config;  
  
import com.zaxxer.hikari.HikariDataSource;  
import jakarta.persistence.SharedCacheMode;  
import jakarta.persistence.ValidationMode;  
import jakarta.persistence.spi.ClassTransformer;  
import jakarta.persistence.spi.PersistenceUnitInfo;  
import jakarta.persistence.spi.PersistenceUnitTransactionType;  
import org.hibernate.cfg.AvailableSettings;  
  
import javax.sql.DataSource;  
import java.net.URL;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Properties;  
  
*/\*\*  
 \* Configuration class for defining persistence unit information.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*public class PersistenceConfig implements PersistenceUnitInfo {  
  
 @Override  
 public void addTransformer(ClassTransformer arg0) {  
  
 }  
  
 @Override  
 public boolean excludeUnlistedClasses() {  
 return false;  
 }  
  
 @Override  
 public ClassLoader getClassLoader() {  
 return null;  
 }  
  
 @Override  
 public List<URL> getJarFileUrls() {  
 return null;  
 }  
  
 @Override  
 public DataSource getJtaDataSource() {  
 HikariDataSource dataSource = new HikariDataSource();  
 dataSource.setJdbcUrl("jdbc:mysql://localhost/jpa\_hibernate\_demo");  
 dataSource.setUsername("hbstudent");  
 dataSource.setPassword("hbstudent");  
  
 return dataSource;  
 }  
  
 @Override  
 public List<String> getManagedClassNames() {  
 return Arrays.*asList*("org.ayushsingh.jdbc\_jpa\_demo.entity.Employee",  
 "org.ayushsingh.jdbc\_jpa\_demo.entity.Department",  
 "org.ayushsingh.jdbc\_jpa\_demo.entity.Project");  
 }  
  
 @Override  
 public List<String> getMappingFileNames() {  
 return null;  
 }  
  
 @Override  
 public ClassLoader getNewTempClassLoader() {  
 return null;  
 }  
  
 @Override  
 public DataSource getNonJtaDataSource() {  
 return null;  
 }  
  
 @Override  
 public String getPersistenceProviderClassName() {  
 return "org.hibernate.jpa.HibernatePersistenceProvider";  
 }  
  
 @Override  
 public String getPersistenceUnitName() {  
 return "my-persistence-unit";  
 }  
  
 @Override  
 public URL getPersistenceUnitRootUrl() {  
  
 return null;  
 }  
  
 @Override  
 public String getPersistenceXMLSchemaVersion() {  
  
 return null;  
 }  
  
 @Override  
 public Properties getProperties() {  
 Properties properties = new Properties();  
  
 properties.put(AvailableSettings.*HBM2DDL\_AUTO*, "update"); //for demo purpose, use update/create (don't use in production)  
 properties.put(AvailableSettings.*FORMAT\_SQL*, true);  
 properties.put(AvailableSettings.*SHOW\_SQL*, true);  
  
 // Specify the dialect for your database (e.g., MySQL)  
 properties.put(AvailableSettings.*DIALECT*, "org.hibernate.dialect.MySQLDialect");  
  
 return properties;  
 }  
  
 @Override  
 public SharedCacheMode getSharedCacheMode() {  
  
 return null;  
 }  
  
 @Override  
 public PersistenceUnitTransactionType getTransactionType() {  
  
 return PersistenceUnitTransactionType.*RESOURCE\_LOCAL*;  
 }  
  
 @Override  
 public ValidationMode getValidationMode() {  
  
 return null;  
 }  
  
}

### DBUtil

package org.ayushsingh.jdbc\_jpa\_demo.util.dbUtil;  
  
import jakarta.persistence.EntityManagerFactory;  
import org.ayushsingh.jdbc\_jpa\_demo.config.PersistenceConfig;  
import org.hibernate.jpa.HibernatePersistenceProvider;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
import java.util.HashMap;  
  
  
*/\*\*  
 \* Utility class for managing the database.  
 \* Provides a method {@link #getEntityManagerFactory()} to obtain the {@link EntityManagerFactory} bean for database operations.  
 \* Avoid creating multiple {@link EntityManagerFactory} beans.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@Configuration  
public class DBUtil {  
  
 @Bean  
 public EntityManagerFactory getEntityManagerFactory() {  
 EntityManagerFactory emf = new HibernatePersistenceProvider().createContainerEntityManagerFactory(new PersistenceConfig(), new HashMap<>());  
 return emf;  
 }  
}

## Entity

### Employee

package org.ayushsingh.jdbc\_jpa\_demo.entity;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
import java.time.LocalDate;  
  
*/\*\*  
 \* Entity class representing an employee.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
@Entity  
@Table(name = "hb\_demo\_Employee")  
public class Employee {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "ID", nullable = false, unique = true)  
 private Long employeeId;  
  
 @Column(name = "Name", nullable = false)  
 private String name;  
  
 @Column(name = "DOB", nullable = false)  
 private LocalDate dob;  
  
 @Column(name = "Salary", nullable = false)  
 private Long salary;  
  
 @Column(name = "Contact", nullable = false, unique = true, length = 10)  
 private Long contact;  
  
 @Column(name = "Email", nullable = false, unique = true)  
 private String email;  
  
 @Column(name = "Address", nullable = false)  
 private String address;  
  
 @OneToOne(cascade = { CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH  
 }, fetch = FetchType.LAZY)  
 @JoinColumn(name = "Project\_ID", referencedColumnName = "ID")  
 private Project project;  
  
 @ManyToOne(cascade = { CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH }, fetch = FetchType.LAZY)  
 @JoinColumn(name="Department\_ID", referencedColumnName = "ID")  
 private Department department;  
}

### Department

package org.ayushsingh.jdbc\_jpa\_demo.entity;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
import java.util.HashSet;  
import java.util.Objects;  
import java.util.Set;  
  
  
*/\*\*  
 \* Entity class representing a department.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
@Table(name="hb\_demo\_Department")  
@Entity  
public class Department {  
   
  
 @Id  
 @Column(name="ID", nullable = false, unique = true)  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 public Long departmentId;  
  
 @Column(name="Name", nullable = false,unique = true)  
 private String name;  
  
 @Column(name="Technology", nullable = false)  
 private String technology;  
  
 @Column(name="Address", nullable = false)  
 private String address;  
  
 @Column(name="Contact", nullable = false, unique = true, length = 10)  
 private Long contact;  
  
 @OneToMany(mappedBy = "department", cascade = {CascadeType.*MERGE*,CascadeType.*PERSIST*,CascadeType.*DETACH*}, fetch = FetchType.*LAZY*)  
 Set<Employee> employees = new HashSet<>();  
  
 public Long getDepartmentId() {  
 return departmentId;  
 }  
  
 public void setDepartmentId(Long departmentId) {  
 this.departmentId = departmentId;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getTechnology() {  
 return technology;  
 }  
  
 public void setTechnology(String technology) {  
 this.technology = technology;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 public Long getContact() {  
 return contact;  
 }  
  
 public void setContact(Long contact) {  
 this.contact = contact;  
 }  
  
 public Set<Employee> getEmployees() {  
 return employees;  
 }  
  
 public void setEmployees(Set<Employee> employees) {  
 this.employees = employees;  
 }  
  
 @Override  
 public String toString() {  
 return "Department{" +  
 "departmentId=" + departmentId +  
 ", name='" + name + '\'' +  
 ", technology='" + technology + '\'' +  
 ", address='" + address + '\'' +  
 ", contact=" + contact +  
 '}';  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Department that = (Department) o;  
 return Objects.*equals*(departmentId, that.departmentId);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(departmentId);  
 }  
}

### Project

package org.ayushsingh.jdbc\_jpa\_demo.entity;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
import java.time.LocalDate;  
  
  
*/\*\*  
 \* Entity class representing a project.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
@Entity  
@Table(name = "hb\_demo\_Project")  
public class Project {  
  
 @Id  
 @Column(name = "ID", nullable = false, unique = true)  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long projectId;  
  
 @Column(name = "Title", nullable = false, unique = true)  
 private String title;  
  
 @Column(name = "Description", nullable = false)  
 private String description;  
  
 @Column(name = "Start\_Date", nullable = false)  
 private LocalDate startDate;  
  
 @Column(name = "End\_Date", nullable = false)  
 private LocalDate endDate;  
  
 @OneToOne(mappedBy = "project", cascade = { CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH }, fetch = FetchType.LAZY)  
 private Employee employee;  
  
  
}

## Service

### EmployeeService

package org.ayushsingh.jdbc\_jpa\_demo.service;  
  
  
import org.ayushsingh.jdbc\_jpa\_demo.dto.employee.EmployeeCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.employee.EmployeeDetailsDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.project.ProjectCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.project.ProjectDetailsDto;  
  
*/\*\*  
 \* Service interface for managing employee-related operations.  
 \* Defines methods for creating, deleting, and updating employees, as well as assigning departments and projects.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*public interface EmployeeService {  
 EmployeeDetailsDto createEmployee(EmployeeCreateDto employee);  
  
 void deleteEmployee(Long employeeId);  
  
 void assignDepartment(Long employeeId, Long departmentId);  
  
 ProjectDetailsDto createProjectForEmployee(Long employeeId, ProjectCreateDto project);  
  
 ProjectDetailsDto updateProjectForEmployee(Long employeeId, ProjectDetailsDto project);  
  
}

### DepartmentService

package org.ayushsingh.jdbc\_jpa\_demo.service;  
  
import org.ayushsingh.jdbc\_jpa\_demo.dto.department.DepartmentCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.department.DepartmentDetailsDto;  
  
  
*/\*\*  
 \* Service interface for managing department-related operations.  
 \* Defines a method for creating departments.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*public interface DepartmentService {  
 DepartmentDetailsDto create(DepartmentCreateDto department);  
}

### DepartmentServiceImpl

package org.ayushsingh.jdbc\_jpa\_demo.service.impl;  
  
  
import lombok.RequiredArgsConstructor;  
import org.ayushsingh.jdbc\_jpa\_demo.dao.DepartmentDao;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.department.DepartmentCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.department.DepartmentDetailsDto;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Department;  
import org.ayushsingh.jdbc\_jpa\_demo.service.DepartmentService;  
import org.modelmapper.ModelMapper;  
import org.springframework.stereotype.Service;  
  
*/\*\*  
 \* Service implementation for managing department-related operations.  
 \* Implements method for creating departments.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@Service  
@RequiredArgsConstructor  
public class DepartmentServiceImpl implements DepartmentService {  
  
 private final DepartmentDao departmentDao;  
 private final ModelMapper modelMapper;  
  
 */\*\*  
 \* Creates a new department.  
 \*  
 \* @param departmentDto The DTO representing the department to be created.  
 \* @return The details as {@link DepartmentDetailsDto} for the created department.  
 \*/* @Override  
 public DepartmentDetailsDto create(DepartmentCreateDto departmentDto) {  
 Department department=this.modelMapper.map(departmentDto,Department.class);  
 department=this.departmentDao.create(department);  
 return this.modelMapper.map(department,DepartmentDetailsDto.class);  
 }  
}

### EmployeeServiceImpl

package org.ayushsingh.jdbc\_jpa\_demo.service.impl;  
  
  
import lombok.RequiredArgsConstructor;  
import org.ayushsingh.jdbc\_jpa\_demo.dao.EmployeeDao;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.employee.EmployeeCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.employee.EmployeeDetailsDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.project.ProjectCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.project.ProjectDetailsDto;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Department;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Employee;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Project;  
import org.ayushsingh.jdbc\_jpa\_demo.service.EmployeeService;  
import org.modelmapper.ModelMapper;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Service;  
  
  
  
*/\*\*  
 \* Service implementation for managing employee-related operations.  
 \* Implements methods for creating, deleting, and updating employees,  
 \* as well as assigning departments and projects.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@Service  
@RequiredArgsConstructor  
public class EmployeeServiceImpl implements EmployeeService {  
  
 private final Logger logger= LoggerFactory.*getLogger*(EmployeeServiceImpl.class);  
 private final EmployeeDao employeeDao;  
 private final ModelMapper modelMapper;  
  
  
 */\*\*  
 \* Creates a new employee.  
 \*  
 \* @param employee The DTO representing the employee to be created.  
 \* @return The details as {@link EmployeeDetailsDto} for the created employee.  
 \*/* @Override  
 public EmployeeDetailsDto createEmployee(EmployeeCreateDto employee) {  
 Employee emp=new Employee();  
 emp.setName(employee.getName());  
 emp.setDob(employee.getDob());  
 emp.setContact(employee.getContact());  
 emp.setEmail(employee.getEmail());  
 emp.setAddress(employee.getAddress());  
 emp.setSalary(employee.getSalary());  
 Project project=this.modelMapper.map(employee.getProject(),Project.class);  
 project.setEmployee(emp);  
 emp.setProject(project);  
 Department department=this.modelMapper.map(employee.getDepartment(),Department.class);  
 emp.setDepartment(department);  
 emp=employeeDao.save(emp);  
 return this.modelMapper.map(emp,EmployeeDetailsDto.class);  
 }  
  
 */\*\*  
 \* Deletes an employee by ID.  
 \*  
 \* @param employeeId The ID of the employee to be deleted.  
 \*/* @Override  
 public void deleteEmployee(Long employeeId) {  
 logger.info("deleting employee with id {}",employeeId);  
 employeeDao.delete(employeeId);  
 logger.info("deleted employee with id {}",employeeId);  
 }  
  
  
 */\*\*  
 \* Updates a project for an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param prj The DTO representing the updated project details.  
 \* @return The details as {@link ProjectDetailsDto} for the updated project.  
 \*/* @Override  
 public ProjectDetailsDto updateProjectForEmployee(Long employeeId, ProjectDetailsDto prj) {  
 logger.info("updating project for employee with id {} project: {}",employeeId,prj);  
 Project project=this.modelMapper.map(prj,Project.class);  
 Project updatedProject=employeeDao.updateProjectForEmployee(employeeId,project);  
 logger.info("updated project for employee with id {} project: {}",employeeId,updatedProject);  
 return this.modelMapper.map(updatedProject,ProjectDetailsDto.class);  
 }  
  
  
 */\*\*  
 \* Assigns a department to an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param departmentId The ID of the department to be assigned.  
 \*/* @Override  
 public void assignDepartment(Long employeeId, Long departmentId) {  
 logger.info("assigning department to employee with id {} department: {}",employeeId,departmentId);  
 this.employeeDao.assignDepartment(employeeId,departmentId);  
 logger.info("assigned department to employee with id {} ",employeeId);  
 }  
  
  
 */\*\*  
 \* Creates a new project for an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param newProject The DTO representing the project to be created.  
 \* @return The details as {@link ProjectDetailsDto} of the created project.  
 \*/* @Override  
 public ProjectDetailsDto createProjectForEmployee(Long employeeId, ProjectCreateDto newProject) {  
 logger.info("creating project for employee with id {} project: {}",employeeId,newProject);  
 Project project=this.modelMapper.map(newProject,Project.class);  
 project=employeeDao.assignNewProject(employeeId,project);  
 logger.info("created project for employee with id {} project: {}",employeeId,project);  
 return this.modelMapper.map(project,ProjectDetailsDto.class);  
 }  
  
}

## Persistence Layer

### EmployeeDao

package org.ayushsingh.jdbc\_jpa\_demo.dao;  
  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Employee;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Project;  
  
*/\*\*  
 \* Interface for performing CRUD operations related to employees and their projects.  
 \* Defines methods for saving, deleting, and updating employee and project entities.  
 \* Also includes methods for assigning departments and new projects to employees.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*public interface EmployeeDao {  
 Employee save(Employee employee);  
 void delete(Long employeeId);  
 void assignDepartment(Long employeeId, Long departmentId);  
  
 Project updateProjectForEmployee(Long employeeId, Project project);  
  
  
 Project assignNewProject(Long employeeId, Project project);  
}

### EmployeeDao Implementation

package org.ayushsingh.jdbc\_jpa\_demo.dao.impl;  
  
  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.EntityManagerFactory;  
import lombok.RequiredArgsConstructor;  
import org.ayushsingh.jdbc\_jpa\_demo.dao.EmployeeDao;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Department;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Employee;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Project;  
import org.ayushsingh.jdbc\_jpa\_demo.util.exceptionUtil.ApiException;  
import org.springframework.stereotype.Component;  
  
import java.util.Objects;  
  
  
*/\*\*  
 \* Implementation of the {@link EmployeeDao} interface for performing CRUD operations related to employees.  
 \* Uses JPA for database interaction.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@Component  
@RequiredArgsConstructor  
public class EmployeeDaoImpl implements EmployeeDao {  
 private final EntityManagerFactory entityManagerFactory;  
  
  
 */\*\*  
 \* Saves an employee entity.  
 \*  
 \* @param employee The employee entity to be saved.  
 \* @return The saved {@link Employee} entity.  
 \*/* @Override  
 public Employee save(Employee employee) {  
 EntityManager entityManager=entityManagerFactory.createEntityManager();  
 try{  
 entityManager.getTransaction().begin();  
 entityManager.persist(employee);  
 entityManager.getTransaction().commit();  
 return employee;  
 }  
 finally {  
 if(entityManager.getTransaction().isActive()) {  
 entityManager.getTransaction().rollback();  
 }  
 entityManager.close();  
 }  
 }  
  
 */\*\*  
 \* Deletes an employee by ID.  
 \*  
 \* @param employeeId The ID of the employee to be deleted.  
 \*/* @Override  
 public void delete(Long employeeId) {  
 EntityManager entityManager=entityManagerFactory.createEntityManager();  
 try{  
 entityManager.getTransaction().begin();  
 Employee employee = entityManager.find(Employee.class, employeeId);  
 entityManager.remove(employee);  
 entityManager.getTransaction().commit();  
 }  
 finally {  
 if(entityManager.getTransaction().isActive()) {  
 entityManager.getTransaction().rollback();  
 }  
 entityManager.close();  
 }  
 }  
  
  
 */\*\*  
 \* Assigns a department to an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param departmentId The ID of the department to be assigned.  
 \*/* @Override  
 public void assignDepartment(Long employeeId, Long departmentId) {  
 EntityManager entityManager=entityManagerFactory.createEntityManager();  
 try{  
 entityManager.getTransaction().begin();  
 Employee employee = entityManager.find(Employee.class, employeeId);  
 if(employee==null){  
 throw new ApiException("Employee not found");  
 }  
 Department department = entityManager.find(Department.class, departmentId);  
 if(department==null){  
 throw new ApiException("Department not found");  
 }  
 employee.setDepartment(department);  
 entityManager.persist(employee);  
 entityManager.getTransaction().commit();  
 }  
 finally {  
 if(entityManager.getTransaction().isActive()) {  
 entityManager.getTransaction().rollback();  
 }  
 entityManager.close();  
 }  
 }  
  
  
 */\*\*  
 \* Updates a project for an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param project The project entity representing the updated project details.  
 \* @return The updated {@link Project} entity.  
 \*/* @Override  
 public Project updateProjectForEmployee(Long employeeId, Project project) {  
 EntityManager entityManager=entityManagerFactory.createEntityManager();  
 try{  
 entityManager.getTransaction().begin();  
 Employee employee = entityManager.find(Employee.class, employeeId);  
 if(employee==null){  
 throw new ApiException("Employee not found");  
 }  
 Project oldProject=employee.getProject();  
 if(oldProject==null){  
 //if no previous project, save new  
 employee.setProject(project);  
 }  
 else{  
 if(!Objects.*equals*(oldProject.getProjectId(), project.getProjectId())){  
 //if old project id and new project id is not same, throw exception  
 throw new ApiException("The project with id "+project.getProjectId()+" does not exist");  
 }  
 oldProject.setTitle(project.getTitle());  
 oldProject.setStartDate(project.getStartDate());  
 oldProject.setEndDate(project.getEndDate());  
 oldProject.setStartDate(project.getStartDate());  
 employee.setProject(oldProject);  
 }  
 entityManager.persist(employee);  
 entityManager.getTransaction().commit();  
 return project;  
 }  
 finally {  
 if(entityManager.getTransaction().isActive()) {  
 entityManager.getTransaction().rollback();  
 }  
 entityManager.close();  
 }  
 }  
  
  
 */\*\*  
 \* Assigns a new project to an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param project The project entity representing the new project to be assigned.  
 \* @return The assigned {@link Project} entity.  
 \*/* @Override  
 public Project assignNewProject(Long employeeId, Project project) {  
 EntityManager entityManager=entityManagerFactory.createEntityManager();  
 try{  
 entityManager.getTransaction().begin();  
 Employee employee = entityManager.find(Employee.class, employeeId);  
 if(employee==null){  
 throw new ApiException("Employee not found");  
 }  
 entityManager.remove(employee.getProject());  
 employee.setProject(project);  
 entityManager.persist(employee);  
 entityManager.getTransaction().commit();  
 return project;  
 }  
 finally {  
 if(entityManager.getTransaction().isActive()) {  
 entityManager.getTransaction().rollback();  
 }  
 entityManager.close();  
 }  
 }  
}

### DepartmentDao

package org.ayushsingh.jdbc\_jpa\_demo.dao;  
  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Department;  
  
*/\*\*  
 \* Interface for performing CRUD operations related to departments.  
 \* Defines a method for creating a new department entity.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*public interface DepartmentDao {  
 Department create(Department department);  
}

### DepartmentDao Implementation

package org.ayushsingh.jdbc\_jpa\_demo.dao.impl;  
  
  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.EntityManagerFactory;  
import lombok.RequiredArgsConstructor;  
import org.ayushsingh.jdbc\_jpa\_demo.dao.DepartmentDao;  
import org.ayushsingh.jdbc\_jpa\_demo.entity.Department;  
import org.springframework.stereotype.Component;  
  
  
*/\*\*  
 \* Implementation of the {@link DepartmentDao} interface for performing CRUD operations related to employees.  
 \* Uses JPA for database interaction.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@Component  
@RequiredArgsConstructor  
public class DepartmentDaoImpl implements DepartmentDao {  
 private final EntityManagerFactory entityManagerFactory;  
  
  
  
 */\*\*  
 \* Creates a new department in the database.  
 \*  
 \* @param department The department object to be created.  
 \* @return The created department object.  
 \*/* @Override  
 public Department create(Department department) {  
  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 try{  
 entityManager.getTransaction().begin();  
 entityManager.persist(department);  
 entityManager.getTransaction().commit();  
 return department;  
 }  
  
 finally {  
 if(entityManager.getTransaction().isActive()){  
 entityManager.getTransaction().rollback();  
 }  
 entityManager.close();  
 }  
 }  
}

## Controller

### DepartmentController

package org.ayushsingh.jdbc\_jpa\_demo.controller;  
  
import lombok.RequiredArgsConstructor;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.department.DepartmentCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.department.DepartmentDetailsDto;  
import org.ayushsingh.jdbc\_jpa\_demo.service.DepartmentService;  
  
import org.ayushsingh.jdbc\_jpa\_demo.util.responseUtil.ApiResponse;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
*/\*\*  
 \* Controller class for managing department-related operations.  
 \* REST APIs for department related operations are defined here.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@RestController  
@RequestMapping("/api/v1/department")  
@RequiredArgsConstructor  
public class DepartmentController {  
 private final DepartmentService departmentService;  
  
  
 */\*\*  
 \* Creates a new department.  
 \*  
 \* @param department The DTO representing the department to be created.  
 \* @return {@link ResponseEntity} containing {@link ApiResponse} with details of the created department.  
 \*/* @PostMapping(value = "/new")  
 public ResponseEntity<ApiResponse<DepartmentDetailsDto>> createDepartment(@RequestBody DepartmentCreateDto department) {  
 DepartmentDetailsDto departmentDetailsDto=departmentService.create(department);  
 return new ResponseEntity<>(new ApiResponse<>(departmentDetailsDto), HttpStatus.*CREATED*);  
 }  
}

### EmployeeController

package org.ayushsingh.jdbc\_jpa\_demo.controller;  
  
import lombok.RequiredArgsConstructor;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.employee.EmployeeCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.employee.EmployeeDetailsDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.project.ProjectCreateDto;  
import org.ayushsingh.jdbc\_jpa\_demo.dto.project.ProjectDetailsDto;  
import org.ayushsingh.jdbc\_jpa\_demo.service.EmployeeService;  
import org.ayushsingh.jdbc\_jpa\_demo.util.responseUtil.ApiResponse;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
*/\*\*  
 \* Controller class for managing employee-related operations.  
 \* REST APIs for employee related operations are defined here.  
 \*  
 \* @author Ayush Singh  
 \* @version 1.0  
 \* @since 2024-04-12  
 \*/*@RestController  
@RequestMapping("/api/v1/employee")  
@RequiredArgsConstructor  
public class EmployeeController {  
  
 private final EmployeeService employeeService;  
  
  
 */\*\*  
 \* Creates a new employee.  
 \*  
 \* @param employee The DTO representing the employee to be created.  
 \* @return {@link ResponseEntity} containing {@link ApiResponse} with details of the created employee.  
 \*/* @PostMapping("/new")  
 public ResponseEntity<ApiResponse<EmployeeDetailsDto>> createEmployee(@RequestBody EmployeeCreateDto employee) {  
 EmployeeDetailsDto createdEmployee = employeeService.createEmployee(employee);  
 return new ResponseEntity<>(new ApiResponse<>(createdEmployee), HttpStatus.*CREATED*);  
 }  
  
  
 */\*\*  
 \* Deletes an employee by ID.  
 \*  
 \* @param employeeId The ID of the employee to be deleted.  
 \* @return {@link ResponseEntity} containing {@link ApiResponse} indicating the success of the operation.  
 \*/* @DeleteMapping("/{employeeId}")  
 public ResponseEntity<ApiResponse<String>> deleteEmployee(@PathVariable Long employeeId) {  
 employeeService.deleteEmployee(employeeId);  
 return new ResponseEntity<>(new ApiResponse<>("Employee deleted successfully"), HttpStatus.*OK*);  
 }  
  
  
 */\*\*  
 \* Assigns a department to an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param departmentId The ID of the department to be assigned.  
 \* @return {@link ResponseEntity} containing {@link ApiResponse} indicating the success of the operation.  
 \*/* @PutMapping("/{employeeId}/department/{departmentId}")  
 public ResponseEntity<ApiResponse<String>> assignDepartment(@PathVariable Long employeeId,  
 @PathVariable Long departmentId) {  
 employeeService.assignDepartment(employeeId, departmentId);  
 return new ResponseEntity<>(new ApiResponse<>("Department assigned successfully"), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Creates a new project for an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param project The DTO representing the project to be created.  
 \* @return {@link ResponseEntity} containing {@link ApiResponse} indicating the success of the operation.  
 \*/* @PostMapping("/project/new/{employeeId}")  
 public ResponseEntity<ProjectDetailsDto> createProjectForEmployee(@PathVariable Long employeeId,  
 @RequestBody ProjectCreateDto project) {  
 ProjectDetailsDto createdProject = employeeService.createProjectForEmployee(employeeId, project);  
 return new ResponseEntity<>(createdProject, HttpStatus.*CREATED*);  
 }  
  
 */\*\*  
 \* Updates a project for an employee.  
 \*  
 \* @param employeeId The ID of the employee.  
 \* @param project The DTO representing the updated project details.  
 \* @return {@link ResponseEntity} containing {@link ApiResponse} indicating the success of the operation.  
 \*/* @PutMapping("/project/{employeeId}")  
 public ResponseEntity<ProjectDetailsDto> updateProjectForEmployee(@PathVariable Long employeeId, @RequestBody ProjectDetailsDto project) {  
 ProjectDetailsDto updatedProject = employeeService.updateProjectForEmployee(employeeId, project);  
 return new ResponseEntity<>(updatedProject, HttpStatus.*OK*);  
 }  
}

# API Requests and Response

## Department

### Create Department

POST: <http://localhost:8080/api/v1/department/new>

**Request Body:**

{

    "name":"Department 1",

    "technology": "Technology 1",

    "address": "Address 1",

    "contact": 8765678965

}

**Response Body:**

{

    "data": {

        "departmentId": 16,

        "name": "Department 1",

        "technology": "Technology 1",

        "address": "Address 1",

        "contact": 8765678965

    },

    "message": "Success",

    "code": "2000"

}

## Employee

### Create New Employee

POST: <http://localhost:8080/api/v1/employee/new>

**Request Body:**

{

    "name": "Ayush Singh",

    "dob": "2002-04-20",

    "salary": 1000000,

    "contact": 8764327865,

    "email": "ayushsingh20april@gmail.com",

    "address": "Agra, UP",

    "project": {

        "title": "Project 1",

        "description": "Des 1",

        "startDate": "2020-02-26",

        "endDate": "2020-03-27"

    },

    "department":{

        "name": "Department 2",

        "technology": "Java",

        "address": "Address 2",

        "contact": 9878986654

    }

}

**Response Body:**

{

    "employeeId": 1,

    "name": "Ayush Singh",

    "dob": "2002-04-20",

    "salary": 1000000,

    "contact": 8764327865,

    "email": "ayushsingh20april@gmail.com",

    "address": "Agra, UP",

    "project": {

        "projectId": "1",

        "title": "Project 1",

        "description": "Des 1",

        "startDate": "2020-02-26",

        "endDate": "2020-03-27"

    },

    "department": {

        "departmentId": 1,

        "name": "Department 2",

        "technology": "Java",

        "address": "Address 2",

        "contact": 9878986654

    }

}

### Delete employee

DELETE: <http://localhost:8080/api/v1/employee/1>

**Response Body:**

{

    "data": "Employee deleted successfully",

    "message": "Success",

    "code": "2000"

}

### Update department

PUT: <http://localhost:8080/api/v1/employee/1/department/1>

**Response Body:**

{

    "data": "Department assigned successfully",

    "message": "Success",

    "code": "2000"

}

### Create new project for employee

POST: <http://localhost:8080/api/v1/employee/project/new/1>

**Request Body:**

{

        "title": "Project 2",

        "description": "Des 2",

        "startDate": "2021-05-17",

        "endDate": "2022-04-26"

    }

**Response Body:**

{

    "projectId": "2",

    "title": "Project 2",

    "description": "Des 2",

    "startDate": "2021-05-17",

    "endDate": "2022-04-26"

}

### Update project

PUT: <http://localhost:8080/api/v1/employee/project/1>

**Request Body:**

{

    "projectId": "2",

    "title": "Project 204",

    "description": "Des 204",

    "startDate": "2022-05-17",

    "endDate": "2023-04-26"

}

**Response Body:**

{

    "projectId": "2",

    "title": "Project 204",

    "description": "Des 204",

    "startDate": "2022-05-17",

    "endDate": "2023-04-26"

}