**Case Study 9 – Project Management System**

1. Database Schema -

Created database -

|  |
| --- |
| create database projectmanagementsystem;  use projectmanagementsystem; |

Created table project -

|  |
| --- |
| create table project (  project\_id int primary key auto\_increment,  project\_name varchar(100) not null,  project\_description varchar(200) not null,  start\_date date not null,  status enum('started', 'dev', 'build', 'test', 'deployed') not null  ); |

Created table employee -

|  |
| --- |
| create table employee (  employee\_id int primary key auto\_increment,  employee\_name varchar(100),  employee\_designation varchar(100),  employee\_gender varchar(10),  employee\_salary double,  project\_id int,  hire\_date date,  foreign key (project\_id) references project(project\_id)  ); |

Created table task -

|  |
| --- |
| create table task (  task\_id int primary key auto\_increment,  task\_name varchar(100),  project\_id int,  employee\_id int,  status varchar(20),  allocation\_date date,  deadline date, foreign key (project\_id) references project(project\_id),  foreign key (employee\_id) references employee(employee\_id) ); |

Created extra table expense to show expense report -

|  |
| --- |
| create table expense (  expense\_id int primary key auto\_increment,  employee\_id int,  amount double,  expense\_date date,  foreign key (employee\_id) references employee(employee\_id) ); |

1. Java Schema -

Created 6 Packages -

|  |
| --- |
| 1. entity 2. dao 3. exception 4. util 5. main 6. unitTest (To test the Project) (JUnit) |

Created Classes in the respective Packages -

|  |
| --- |
| entity package - 1. Project.java 2. Employee.java 3. Task.java 4. ExpenseReport.java (Extra class to show the expense report)  dao package - 1. IProjectRepository.java (Interface) 2. ProjectRepositoryImpl.java  exception package -   1. EmployeeNotFoundException.java 2. ProjectNotFoundException.java 3. TaskNotFoundException.java   util package - 1. DBConnUtil.java  2. DBPropertyUtil.java  main package - 1. ProjectApp.java (main class)  unitTest package - 1. ProjectAppTest.java (JUnit) |

3. entity package -

Project.java

|  |
| --- |
| package entity; import java.time.LocalDate;  public class Project {  private int project\_id;  private String project\_name;  private String project\_description;  private LocalDate startDate;  private String status;  //default constructor public Project() {   }  //parameterized constructor public Project(int project\_id, String project\_name, String project\_description, LocalDate startDate, String status) {  this.project\_id = project\_id;  this.project\_name = project\_name;  this.project\_description = project\_description;  this.startDate = startDate;  this.status = status; }  //getters and setter methods public int getProjectId() {  return project\_id; } public void setProjectId(int project\_id) {  this.project\_id = project\_id; }  public String getProjectName() {  return project\_name; } public void setProjectName(String project\_name) {  this.project\_name = project\_name; }  public String getDescription() {  return project\_description; } public void setDescription(String project\_description) {  this.project\_description = project\_description; }  public LocalDate getStartDate(){  return startDate; } public void setStartDate(LocalDate startDate){  this.startDate = startDate; }  public String getStatus() {  return status; } public void setStatus(String status) {  if(status.equalsIgnoreCase("started") || status.equalsIgnoreCase("dev") || status.equalsIgnoreCase("build") ||  status.equalsIgnoreCase("test") || status.equalsIgnoreCase("deployed")) {  this.status = status;  }  else {  throw new IllegalArgumentException("Invalid status value! Must be: started, dev, build, test, deployed");  } }  } |

Employee.java

|  |
| --- |
| package entity;  public class Employee {  private int employee\_id;  private String employee\_name;  private String employee\_designation;  private String employee\_gender;  private double employee\_salary;  private int project\_id;  // Default constructor public Employee() { }  // Parameterized constructor public Employee(int employee\_id, String employee\_name, String employee\_designation, String employee\_gender, double employee\_salary, int project\_id) {  this.employee\_id = employee\_id;  this.employee\_name = employee\_name;  this.employee\_designation = employee\_designation;  this.employee\_gender = employee\_gender;  this.employee\_salary = employee\_salary;  this.project\_id = project\_id; }  // Getter and setter methods public int getEmployeeId() {  return employee\_id; }  public void setEmployeeId(int employee\_id) {  this.employee\_id = employee\_id; } public String getEmployeeName() {  return employee\_name; }  public void setEmployeeName(String employee\_name) {  this.employee\_name = employee\_name; }  public String getEmployeeDesignation() {  return employee\_designation; }  public void setEmployeeDesignation(String employee\_designation) {  this.employee\_designation = employee\_designation; }  public String getEmployeeGender() {  return employee\_gender; }  public void setEmployeeGender(String employee\_gender) {  this.employee\_gender = employee\_gender; }  public double getEmployeeSalary() {  return employee\_salary; }  public void setEmployeeSalary(double employee\_salary) {  this.employee\_salary = employee\_salary; }  public int getProjectId() {  return project\_id; }  public void setProjectId(int project\_id) {  this.project\_id = project\_id; } } |

Task.java

|  |
| --- |
| package entity;  import java.time.LocalDate;  public class Task {  private int task\_id;  private String task\_name;  private int project\_id;  private int employee\_id;  private String status;  private LocalDate allocationDate;  private LocalDate deadline;  public Task() {}  public Task(int task\_id, String task\_name, int project\_id, int employee\_id, String status, LocalDate allocationDate, LocalDate deadline) {  this.task\_id = task\_id;  this.task\_name = task\_name;  this.project\_id = project\_id;  this.employee\_id = employee\_id;  this.status = status;  this.allocationDate = allocationDate;  this.deadline = deadline; }  public int getTaskId() { return task\_id; } public void setTaskId(int taskId) { this.task\_id = taskId; }  public String getTaskName() { return task\_name; } public void setTaskName(String taskName) { this.task\_name = taskName; }  public int getProjectId() { return project\_id; } public void setProjectId(int projectId) { this.project\_id = projectId; }  public int getEmployeeId() { return employee\_id; } public void setEmployeeId(int employeeId) { this.employee\_id = employeeId; }  public String getStatus() { return status; } public void setStatus(String status) {  if (status.equalsIgnoreCase("assigned") || status.equalsIgnoreCase("started") || status.equalsIgnoreCase("completed")) {  this.status = status;  } else {  throw new IllegalArgumentException("Invalid status. Must be: assigned/started/completed");  } }  public LocalDate getAllocationDate() { return allocationDate; } public void setAllocationDate(LocalDate allocationDate) { this.allocationDate = allocationDate; }  public LocalDate getDeadline() { return deadline; } public void setDeadline(LocalDate deadline) { this.deadline = deadline; }  @Override public String toString() {  return task\_id + " - " + task\_name + " - " + status + " | Start: " + allocationDate + ", Deadline: " + deadline; } } |

ExpenseReport.java

|  |
| --- |
| package entity;    import java.time.LocalDate;    public class ExpenseReport {  private int employeeId;  private double amount;  private LocalDate date;  public ExpenseReport() {  }    public ExpenseReport(int employeeId, double amount, LocalDate date) {  this.employeeId = employeeId;  this.amount = amount;  this.date = date;  }    public int getEmployeeId() {  return employeeId;  }    public void setEmployeeId(int employeeId) {  this.employeeId = employeeId;  }    public double getAmount() {  return amount;  }    public void setAmount(double amount) {  this.amount = amount;  }    public LocalDate getDate() {  return date;  }    public void setDate(LocalDate date) {  this.date = date;  }    @Override  public String toString() {  return "Employee ID: " + employeeId + ", Amount: " + amount + ", Date: " + date;  }  } |

1. dao package -

IProjectRepository.java (Interface)

|  |
| --- |
| package dao;  import entity.Employee;  import entity.ExpenseReport;  import entity.Project;  import entity.Task;  import java.time.LocalDate;  import java.util.List;  import exception.EmployeeNotFoundException;  import exception.ProjectNotFoundException;  import exception.TaskNotFoundException;  public interface IProjectRepository {  boolean createEmployee(Employee emp); boolean createProject(Project pj); boolean createTask(Task tk); boolean assignProjectToEmployee(int project\_id, int employee\_id); boolean assignTaskInProjectToEmployee(int task\_id, int project\_id, int employee\_id); public boolean deleteEmployee(int employee\_id) throws EmployeeNotFoundException; public boolean deleteProject(int project\_id) throws ProjectNotFoundException; public boolean deleteTask(int task\_id) throws TaskNotFoundException; boolean updateProject(Project pj) throws ProjectNotFoundException; List<Task> getAllTasks(int employee\_id, int project\_id); List<Project> getAllProjects(); List<Employee> getAllEmployees(); List<ExpenseReport> generateExpenseReport(LocalDate startDate, LocalDate endDate);  } |

ProjectRepositoryImpl.java

|  |
| --- |
| package dao;  import entity.Employee;  import entity.ExpenseReport;  import entity.Task;  import entity.Project;  import exception.EmployeeNotFoundException;  import exception.ProjectNotFoundException;  import exception.TaskNotFoundException;  import util.DBConnUtil;  import java.sql.\*;  import java.time.LocalDate;  import java.util.ArrayList;  import java.util.List;  public class ProjectRepositoryImpl implements IProjectRepository {  private Connection conn; public ProjectRepositoryImpl() {  this.conn = DBConnUtil.getDbConnection();  System.out.println("DB connection in constructor: " + this.conn); }  @Override public boolean createEmployee(Employee emp) {  String sql = "INSERT INTO employee(employee\_name, employee\_designation, employee\_gender, employee\_salary, project\_id) VALUES (?,?,?,?,?)";  try (PreparedStatement stmt = conn.prepareStatement(sql)) {  stmt.setString(1, emp.getEmployeeName());  stmt.setString(2, emp.getEmployeeDesignation());  stmt.setString(3, emp.getEmployeeGender());  stmt.setDouble(4, emp.getEmployeeSalary());   stmt.setInt(5, emp.getProjectId());    return stmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }   return false; }  @Override public boolean createProject(Project pj) {  String sql = "INSERT INTO project(project\_name, project\_description, start\_date, status) VALUES (?, ?, ?, ?)";  try (PreparedStatement stmt = conn.prepareStatement(sql)) {  stmt.setString(1, pj.getProjectName());  stmt.setString(2, pj.getDescription());  stmt.setDate(3, java.sql.Date.valueOf(pj.getStartDate()));  stmt.setString(4, pj.getStatus());   return stmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }  return false; }  @Override public boolean createTask(Task tk) {  String sql = "INSERT INTO task(task\_name, project\_id, employee\_id, status, allocation\_date, deadline) VALUES (?, ?, ?, ?, ?, ?)";  try (PreparedStatement stmt = conn.prepareStatement(sql)) {  stmt.setString(1, tk.getTaskName());  stmt.setInt(2, tk.getProjectId());  stmt.setInt(3, tk.getEmployeeId());  stmt.setString(4, tk.getStatus());  stmt.setDate(5, Date.valueOf(tk.getAllocationDate()));  stmt.setDate(6, Date.valueOf(tk.getDeadline()));   return stmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }   return false; }  @Override public boolean assignProjectToEmployee(int project\_id, int employee\_id) {  String sql = "UPDATE employee SET project\_id = ? WHERE employee\_id = ?";  try (PreparedStatement stmt = conn.prepareStatement(sql)) {  stmt.setInt(1, project\_id);  stmt.setInt(2, employee\_id);   return stmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }   return false; }  @Override public boolean assignTaskInProjectToEmployee(int task\_id, int project\_id, int employee\_id) {  String sql = "UPDATE task SET project\_id = ?, employee\_id = ? WHERE task\_id = ?";  try (PreparedStatement stmt = conn.prepareStatement(sql)) {  stmt.setInt(1, project\_id);  stmt.setInt(2, employee\_id);  stmt.setInt(3, task\_id);   return stmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }   return false; }  @Override public boolean deleteEmployee(int employee\_id) throws EmployeeNotFoundException {  String checkSql = "SELECT employee\_id FROM employee WHERE employee\_id = ?";  String deleteExpensesSql = "DELETE FROM expense WHERE employee\_id = ?";  String deleteTasksSql = "DELETE FROM task WHERE employee\_id = ?";  String deleteEmployeeSql = "DELETE FROM employee WHERE employee\_id = ?";   try (Connection con = DBConnUtil.getDbConnection()) {  // Check if employee exists  try (PreparedStatement checkStmt = con.prepareStatement(checkSql)) {  checkStmt.setInt(1, employee\_id);  ResultSet rs = checkStmt.executeQuery();  if (!rs.next()) {  throw new EmployeeNotFoundException("Employee with ID " + employee\_id + " not found.");  }  }    try (PreparedStatement deleteExpensesStmt = con.prepareStatement(deleteExpensesSql)) {  deleteExpensesStmt.setInt(1, employee\_id);  deleteExpensesStmt.executeUpdate();  }   // Delete associated tasks  try (PreparedStatement deleteTasksStmt = con.prepareStatement(deleteTasksSql)) {  deleteTasksStmt.setInt(1, employee\_id);  deleteTasksStmt.executeUpdate();  }   // Delete the employee  try (PreparedStatement deleteEmployeeStmt = con.prepareStatement(deleteEmployeeSql)) {  deleteEmployeeStmt.setInt(1, employee\_id);  return deleteEmployeeStmt.executeUpdate() > 0;  }   } catch (SQLException e) {  e.printStackTrace();  }   return false; }   @Override public boolean deleteProject(int project\_id) throws ProjectNotFoundException {  String checkSql = "SELECT project\_name FROM project WHERE project\_id = ?";  String deleteSql = "DELETE FROM project WHERE project\_id = ?";   try (PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {  checkStmt.setInt(1, project\_id);  ResultSet rs = checkStmt.executeQuery();  if (!rs.next()) {  throw new ProjectNotFoundException("Project with id " + project\_id + " not found");  }  } catch (SQLException e) {  e.printStackTrace();  }   try (PreparedStatement deleteStmt = conn.prepareStatement(deleteSql)) {  deleteStmt.setInt(1, project\_id);  return deleteStmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }   return false; }   @Override public boolean deleteTask(int task\_id) throws TaskNotFoundException{  String checkSql = "SELECT task\_name FROM task WHERE task\_id = ?";  String deleteSql = "DELETE FROM task WHERE task\_id = ?";   try (PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {  checkStmt.setInt(1, task\_id);  ResultSet rs = checkStmt.executeQuery();  if (!rs.next()) {  throw new TaskNotFoundException("Task with id " + task\_id + " not found");  }  } catch (SQLException e) {  e.printStackTrace();  }   try (PreparedStatement deleteStmt = conn.prepareStatement(deleteSql)) {  deleteStmt.setInt(1, task\_id);  return deleteStmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }   return false; }   @Override public List<Task> getAllTasks(int employee\_id, int project\_id) {  List<Task> tasks = new ArrayList<>();  String sql = "SELECT \* FROM task WHERE employee\_id = ? AND project\_id = ?";   try (Connection conn = DBConnUtil.getDbConnection();  PreparedStatement stmt = conn.prepareStatement(sql)) {   stmt.setInt(1, employee\_id);  stmt.setInt(2, project\_id);   try (ResultSet rs = stmt.executeQuery()) {  while (rs.next()) {  tasks.add(new Task(  rs.getInt("task\_id"),  rs.getString("task\_name"),  rs.getInt("project\_id"),  rs.getInt("employee\_id"),  rs.getString("status"),  rs.getDate("allocation\_date").toLocalDate(),  rs.getDate("deadline").toLocalDate()  ));  }  }  } catch (SQLException e) {  e.printStackTrace();  }   return tasks; }  public List<Project> getAllProjects() {  List<Project> projectList = new ArrayList<>();  String query = "SELECT \* FROM Project";   try (Connection con = DBConnUtil.getDbConnection();  Statement stmt = con.createStatement();  ResultSet rs = stmt.executeQuery(query)) {   while (rs.next()) {  Project p = new Project(  rs.getInt("project\_id"),  rs.getString("project\_name"),  rs.getString("project\_description"),  rs.getDate("start\_date").toLocalDate(),  rs.getString("status")  );  projectList.add(p);  }  } catch (SQLException e) {  e.printStackTrace();  }   return projectList; }  @Override public List<Employee> getAllEmployees() {  List<Employee> employeeList = new ArrayList<>();  String query = "SELECT \* FROM employee";   try (Connection con = DBConnUtil.getDbConnection();  Statement stmt = con.createStatement();  ResultSet rs = stmt.executeQuery(query)) {   while (rs.next()) {  Employee emp = new Employee(  rs.getInt("employeeId"),  rs.getString("employeeName"),  rs.getString("role"),  rs.getString("gender"),  rs.getDouble("salary"),  rs.getInt("projectId")  );  employeeList.add(emp);  }  } catch (SQLException e) {  e.printStackTrace();  }   return employeeList; }  public Project getProjectByNameAndDate(String name, LocalDate date) {  String query = "SELECT \* FROM project WHERE project\_name = ? AND start\_date = ? ORDER BY project\_id DESC LIMIT 1";  try (Connection conn = DBConnUtil.getDbConnection();  PreparedStatement ps = conn.prepareStatement(query)) {  ps.setString(1, name);  ps.setDate(2, Date.valueOf(date));  ResultSet rs = ps.executeQuery();  if (rs.next()) {  Project p = new Project();  p.setProjectId(rs.getInt("project\_id"));  p.setProjectName(rs.getString("project\_name"));  p.setDescription(rs.getString("project\_description"));  p.setStartDate(rs.getDate("start\_date").toLocalDate());  p.setStatus(rs.getString("status"));  return p;  }  } catch (SQLException e) {  e.printStackTrace();  }  return null; }  public Employee getEmployeeByNameAndProject(String name, int projectId) {  String query = "SELECT \* FROM employee WHERE employee\_name = ? AND project\_id = ? ORDER BY employee\_id DESC LIMIT 1";  try (Connection conn = DBConnUtil.getDbConnection();  PreparedStatement ps = conn.prepareStatement(query)) {  ps.setString(1, name);  ps.setInt(2, projectId);  ResultSet rs = ps.executeQuery();  if (rs.next()) {  return new Employee(  rs.getInt("employee\_id"),  rs.getString("employee\_name"),  rs.getString("employee\_designation"),  rs.getString("employee\_gender"),  rs.getFloat("employee\_salary"),  rs.getInt("project\_id")  );  }  } catch (SQLException e) {  e.printStackTrace();  }  return null; }  @Override public boolean updateProject(Project pj) throws ProjectNotFoundException {  String checkSql = "SELECT \* FROM project WHERE project\_id = ?";  String updateSql = "UPDATE project SET project\_name = ?, project\_description = ?, start\_date = ?, status = ? WHERE project\_id = ?";   try (PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {  checkStmt.setInt(1, pj.getProjectId());  ResultSet rs = checkStmt.executeQuery();  if (!rs.next()) {  throw new ProjectNotFoundException("Project with ID " + pj.getProjectId() + " not found.");  }  } catch (SQLException e) {  e.printStackTrace();  }   try (PreparedStatement updateStmt = conn.prepareStatement(updateSql)) {  updateStmt.setString(1, pj.getProjectName());  updateStmt.setString(2, pj.getDescription());  updateStmt.setDate(3, Date.valueOf(pj.getStartDate()));  updateStmt.setString(4, pj.getStatus());  updateStmt.setInt(5, pj.getProjectId());   return updateStmt.executeUpdate() > 0;  } catch (SQLException e) {  e.printStackTrace();  }   return false; }  @Override public List<ExpenseReport> generateExpenseReport(LocalDate start, LocalDate end) {  List<ExpenseReport> reports = new ArrayList<>();  String sql = "SELECT employee\_id, amount, expense\_date FROM expense WHERE expense\_date BETWEEN ? AND ?";   try (Connection conn = DBConnUtil.getDbConnection();  PreparedStatement ps = conn.prepareStatement(sql)) {   ps.setDate(1, java.sql.Date.valueOf(start));  ps.setDate(2, java.sql.Date.valueOf(end));   try (ResultSet rs = ps.executeQuery()) {  while (rs.next()) {  reports.add(new ExpenseReport(  rs.getInt("employee\_id"),  rs.getDouble("amount"),  rs.getDate("expense\_date").toLocalDate()  ));  }  }  } catch (SQLException e) {  e.printStackTrace();  }   return reports; }  } |

5. exception package -

EmployeeNotFoundException.java

|  |
| --- |
| package exception;    public class EmployeeNotFoundException extends Exception {  public EmployeeNotFoundException(String message) {  super(message);  }  } |

ProjectNotFoundException.java

|  |
| --- |
| package exception;    public class ProjectNotFoundException extends Exception {  public ProjectNotFoundException(String message) {  super(message);  }  } |

TaskNotFoundException.java

|  |
| --- |
| package exception;    public class TaskNotFoundException extends Exception {  public TaskNotFoundException(String message) {  super(message);  }  } |

6. util package -

db.properties File (Not in util package, it is in Project folder)

|  |
| --- |
| user=root  password=Sanika@12345  protocol=jdbc:mysql:  system=localhost  database=projectmanagementsystem  port=3306 |

DBConnUtil.java

|  |
| --- |
| package util;    import java.io.IOException;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;    public class DBConnUtil {    private static final String fileName = "db.properties";    public static Connection getDbConnection() {  Connection con = null;    try {  Class.forName("com.mysql.cj.jdbc.Driver");  String connString = DBPropertyUtil.getConnectionString(fileName);  System.out.println("Connection String: " + connString);    con = DriverManager.getConnection(connString);  System.out.println("DB Connection Established");    } catch (ClassNotFoundException e) {  System.out.println("JDBC Driver not found");  e.printStackTrace();    } catch (IOException e) {  System.out.println("Failed to read DB properties file");  e.printStackTrace();    } catch (SQLException e) {  System.out.println("Error while connecting to the database");  e.printStackTrace();  }    return con;  }  } |

DBPropertyUtil.java

|  |
| --- |
| package util;    import java.io.FileInputStream;  import java.io.IOException;  import java.util.Properties;    public class DBPropertyUtil {  public static String getConnectionString(String fileName) throws IOException {  String connStr = null;  Properties props = new Properties();  FileInputStream fis = new FileInputStream(fileName);  props.load(fis);    String user = props.getProperty("user");  String password = props.getProperty("password");  String protocol = props.getProperty("protocol");  String system = props.getProperty("system");  String database = props.getProperty("database");  String port = props.getProperty("port");    connStr = protocol + "//" + system + ":" + port + "/" + database +  "?user=" + user + "&password=" + password;    return connStr;  }  } |

7. main package -

ProjectApp.java

|  |
| --- |
| package main;  import dao.ProjectRepositoryImpl; import entity.Employee; import entity.ExpenseReport; import entity.Project; import entity.Task; import exception.EmployeeNotFoundException; import exception.ProjectNotFoundException; import exception.TaskNotFoundException;  import java.time.LocalDate; import java.util.List; import java.util.Scanner;  public class ProjectApp {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  ProjectRepositoryImpl repo = new ProjectRepositoryImpl();   while (true) {  System.out.println("\n \*\*\*\*\*PROJECT MANAGEMENT SYSTEM\*\*\*\*\*");  System.out.println("1. Add Employee");  System.out.println("2. Add Project");  System.out.println("3. Add Task");  System.out.println("4. Assign Project to Employee");  System.out.println("5. Assign Task within a Project to an Employee");  System.out.println("6. Delete Employee");  System.out.println("7. Delete Task");  System.out.println("8. List all projects assigned with tasks to an employee");  System.out.println("9. Update a Project");  System.out.println("10. View All Projects");  System.out.println("11. Generate Expense Report");  System.out.println("12. Exit");  System.out.print("Enter your choice: ");  int choice = scanner.nextInt();   try {  switch (choice) {  case 1 -> addEmployee(scanner, repo);  case 2 -> addProject(scanner, repo);  case 3 -> addTask(scanner, repo);  case 4 -> assignProjectToEmployee(scanner, repo);  case 5 -> assignTaskToEmployee(scanner, repo);  case 6 -> deleteEmployee(scanner, repo);  case 7 -> deleteTask(scanner, repo);  case 8 -> listEmployeeProjects(scanner, repo);  case 9 -> updateProject(scanner, repo);  case 10 -> viewAllProjects(repo);  case 11 -> generateExpenseReport(scanner, repo);  case 12 -> {  System.out.println("Exiting...");  scanner.close();  return;  }  default -> System.out.println("Invalid choice! Try again...");  }  } catch (EmployeeNotFoundException | TaskNotFoundException | ProjectNotFoundException e) {  System.out.println("Error: " + e.getMessage());  }  } }  private static void addEmployee(Scanner scanner, ProjectRepositoryImpl repo) {  System.out.print("Enter Employee Name: ");  scanner.nextLine();  String name = scanner.nextLine();  System.out.print("Enter Designation: ");  String designation = scanner.nextLine();  System.out.print("Enter Gender: ");  String gender = scanner.nextLine();  System.out.print("Enter Salary: ");  double salary = scanner.nextDouble();  System.out.print("Enter ID of Project Assigned: ");  int projectId = scanner.nextInt();   Employee emp = new Employee(0, name, designation, gender, salary, projectId);  System.out.println(repo.createEmployee(emp) ? "Employee added successfully!" : "Failed to add employee."); }  private static void addProject(Scanner scanner, ProjectRepositoryImpl repo) {  System.out.print("Enter Project Name: ");  scanner.nextLine();  String projectName = scanner.nextLine();  System.out.print("Enter Project Description: ");  String description = scanner.nextLine();  System.out.print("Enter Start Date (YYYY-MM-DD): ");  LocalDate startDate = LocalDate.parse(scanner.nextLine());  System.out.print("Enter Status (started/dev/build/test/deployed): ");  String status = scanner.nextLine();   Project project = new Project(0, projectName, description, startDate, status);  System.out.println(repo.createProject(project) ? "Project added successfully!" : "Failed to add project."); }  private static void addTask(Scanner scanner, ProjectRepositoryImpl repo) {  System.out.print("Enter Task Name: ");  scanner.nextLine();  String taskName = scanner.nextLine();  System.out.print("Enter Project ID: ");  int projectId = scanner.nextInt();  System.out.print("Enter Employee ID: ");  int employeeId = scanner.nextInt();  scanner.nextLine(); // flush  System.out.print("Enter Task Status (Assigned/Started/Completed): ");  String status = scanner.nextLine();  System.out.print("Enter Allocation Date (YYYY-MM-DD): ");  LocalDate allocationDate = LocalDate.parse(scanner.nextLine());  System.out.print("Enter Deadline Date (YYYY-MM-DD): ");  LocalDate deadline = LocalDate.parse(scanner.nextLine());   Task task = new Task(0, taskName, projectId, employeeId, status, allocationDate, deadline);  System.out.println(repo.createTask(task) ? "Task added successfully!" : "Failed to add task."); }  private static void assignProjectToEmployee(Scanner scanner, ProjectRepositoryImpl repo) {  System.out.print("Enter Project ID: ");  int projectId = scanner.nextInt();  System.out.print("Enter Employee ID: ");  int employeeId = scanner.nextInt();  System.out.println(repo.assignProjectToEmployee(projectId, employeeId)  ? "Project assigned to employee successfully!"  : "Failed to assign project."); }  private static void assignTaskToEmployee(Scanner scanner, ProjectRepositoryImpl repo) {  System.out.print("Enter Task ID: ");  int taskId = scanner.nextInt();  System.out.print("Enter Project ID: ");  int projectId = scanner.nextInt();  System.out.print("Enter Employee ID: ");  int employeeId = scanner.nextInt();  System.out.println(repo.assignTaskInProjectToEmployee(taskId, projectId, employeeId)  ? "Task assigned successfully!" : "Failed to assign task."); }  private static void deleteEmployee(Scanner scanner, ProjectRepositoryImpl repo) throws EmployeeNotFoundException {  System.out.print("Enter Employee ID to delete: ");  int empId = scanner.nextInt();  System.out.println(repo.deleteEmployee(empId) ? "Employee deleted successfully." : "Failed to delete employee."); }  private static void deleteTask(Scanner scanner, ProjectRepositoryImpl repo) throws TaskNotFoundException {  System.out.print("Enter Task ID to delete: ");  int taskId = scanner.nextInt();  System.out.println(repo.deleteTask(taskId) ? "Task deleted successfully." : "Failed to delete task."); }  private static void listEmployeeProjects(Scanner scanner, ProjectRepositoryImpl repo) {  System.out.print("Enter Employee ID: ");  int empId = scanner.nextInt();  System.out.print("Enter Project ID: ");  int projectId = scanner.nextInt();  System.out.println("Tasks Assigned to Employee in the Project:");  repo.getAllTasks(empId, projectId).forEach(System.out::println); }  private static void updateProject(Scanner scanner, ProjectRepositoryImpl repo) throws ProjectNotFoundException {  System.out.print("Enter Project ID to update: ");  int projectId = scanner.nextInt();  scanner.nextLine(); // consume newline  System.out.print("Enter New Project Name: ");  String name = scanner.nextLine();  System.out.print("Enter New Description: ");  String desc = scanner.nextLine();  System.out.print("Enter New Start Date (YYYY-MM-DD): ");  LocalDate start = LocalDate.parse(scanner.nextLine());  System.out.print("Enter New Status: ");  String status = scanner.nextLine();   Project p = new Project(projectId, name, desc, start, status);  System.out.println(repo.updateProject(p) ? "Project updated." : "Failed to update project."); }  private static void viewAllProjects(ProjectRepositoryImpl repo) {  List<Project> projects = repo.getAllProjects();  System.out.println("All Projects:");  projects.forEach(p -> System.out.println(p.getProjectId() + " - " + p.getProjectName())); }  private static void generateExpenseReport(Scanner scanner, ProjectRepositoryImpl repo) {  System.out.print("Enter Start Date (YYYY-MM-DD): ");  scanner.nextLine();  LocalDate startDate = LocalDate.parse(scanner.nextLine());  System.out.print("Enter End Date (YYYY-MM-DD): ");  LocalDate endDate = LocalDate.parse(scanner.nextLine());  List<ExpenseReport> total = repo.generateExpenseReport(startDate, endDate);  System.out.println("Total Expenses from " + startDate + " to " + endDate + ": " + total); }  } |

8. unitTest package (JUnit) -

ProjectAppTest.java

|  |
| --- |
| package unitTest;    import dao.ProjectRepositoryImpl;  import entity.Employee;  import entity.Project;  import entity.Task;  import exception.EmployeeNotFoundException;  import exception.ProjectNotFoundException;  import exception.TaskNotFoundException;    import org.junit.jupiter.api.\*;    import java.time.LocalDate;  import java.util.List;    import static org.junit.jupiter.api.Assertions.\*;    @TestMethodOrder(MethodOrderer.OrderAnnotation.class)  public class ProjectAppTest {    private static ProjectRepositoryImpl repo;    @BeforeAll  static void initRepo() {  repo = new ProjectRepositoryImpl();  }    private int createProjectAndReturnId() {  Project project = new Project();  project.setProjectName("Test Project");  project.setDescription("For testing");  project.setStartDate(LocalDate.now());  project.setStatus("started");    boolean success = repo.createProject(project);  assertTrue(success, "Project creation should succeed");    Project inserted = repo.getProjectByNameAndDate("Test Project", LocalDate.now());  assertNotNull(inserted, "Inserted project should be retrievable");  return inserted.getProjectId();  }    private int createEmployeeAndReturnId(int projectId) {  Employee emp = new Employee(0, "Sanika", "Developer", "Female", 500000, projectId);  boolean created = repo.createEmployee(emp);  assertTrue(created, "Employee creation should succeed");    Employee inserted = repo.getEmployeeByNameAndProject("Sanika", projectId);  assertNotNull(inserted, "Inserted employee should be retrievable");  return inserted.getEmployeeId();  }    @Test  @Order(1)  public void testCreateEmployeeSuccess() {  int projectId = createProjectAndReturnId();  Employee emp = new Employee(0, "Sanika", "Tester", "Female", 400000, projectId);  boolean result = repo.createEmployee(emp);  assertTrue(result, "Employee created successfully");  }    @Test  @Order(2)  public void testCreateTaskSuccess() {  int projectId = createProjectAndReturnId();  int employeeId = createEmployeeAndReturnId(projectId);    Task task = new Task(0, "Unit Test Task", projectId, employeeId, "assigned", LocalDate.now(), LocalDate.now().plusDays(7));  boolean result = repo.createTask(task);  assertTrue(result, "Task created successfully");  }    @Test  @Order(3)  public void testSearchTasksForEmployeeInProject() {  int projectId = createProjectAndReturnId();  int employeeId = createEmployeeAndReturnId(projectId);    Task task = new Task(0, "Search Task", projectId, employeeId, "started", LocalDate.now(), LocalDate.now().plusDays(5));  boolean created = repo.createTask(task);  assertTrue(created, "Task should be created for searching");    List<Task> tasks = repo.getAllTasks(employeeId, projectId);  assertNotNull(tasks, "Task list should not be null");  assertFalse(tasks.isEmpty(), "Task list should not be empty");  }    @Test  @Order(4)  public void testEmployeeNotFoundExceptionThrown() {  int invalidEmpId = 9999;  assertThrows(EmployeeNotFoundException.class, () -> {  repo.deleteEmployee(invalidEmpId);  });  }    @Test  @Order(5)  public void testProjectNotFoundExceptionThrown() {  int invalidProjectId = -1;  assertThrows(ProjectNotFoundException.class, () -> {  repo.deleteProject(invalidProjectId);  });  }    @Test  @Order(6)  public void testTaskNotFoundExceptionThrown() {  int invalidTaskId = 9999;  assertThrows(TaskNotFoundException.class, () -> {  repo.deleteTask(invalidTaskId);  });  }  } |

9. Output -

JUnit Test -

|  |
| --- |
|  |

Add Employee

|  |
| --- |
|  |

|  |
| --- |
|  |

Add Project

|  |
| --- |
|  |

|  |
| --- |
|  |

Add Task

|  |
| --- |
|  |

|  |
| --- |
|  |

Assign Project to Employee

|  |
| --- |
|  |

|  |
| --- |
|  |

Assign Task within a project to employee

|  |
| --- |
|  |

|  |
| --- |
|  |

Delete Employee

|  |
| --- |
|  |

Delete Task

|  |
| --- |
|  |

List all projects assigned with tasks to an employee

|  |
| --- |
|  |

Update a Project

|  |
| --- |
|  |

View All Projects

|  |
| --- |
|  |

Generate Expense Report

|  |
| --- |
|  |