Project Management System

Project Title: Project Management System
Submitted By: Keerthika Nagarajan
Superset ID: 5370583

1. INTRODUCTION

The **Project Management System** is a console-based application designed to streamline the management of employees, projects, and tasks within an organization. It simplifies the process of assigning projects and tasks to employees, tracking progress, and organizing work efficiently. Built using Python and MySQL, the system provides a structured and user-friendly way to manage essential project operations.

This application follows a modular design with clear separation of concerns using Object-Oriented Programming (OOP) principles. It includes various components such as entities, data access objects (DAO), utility classes, and custom exceptions for better maintainability and error handling. The integration with MySQL ensures reliable and persistent data storage, making it a practical solution for real-time project management needs.

2. PURPOSE OF THE PROJECT

The primary purpose of the **Project Management System** is to provide an efficient, organized, and user-friendly solution for managing projects, tasks, and employee assignments in a structured manner. In many organizations, project management is done manually, leading to inefficiencies, miscommunication, and difficulty in tracking progress. This system aims to automate these processes and streamline workflow.

It allows managers to create and manage projects, assign tasks to employees, monitor progress, set deadlines, and generate reports. With Python as the core programming language and MySQL as the database, the system ensures reliability, security, and scalability. Exception handling, unit testing, and modular design improve maintainability and software quality.

Key Objectives Include:

- To create a centralized system for managing projects and tasks.
- To automate the process of assigning employees to projects and tasks.
- To track task status, deadlines, and employee performance in real-time.
- To maintain and manage project data efficiently using a MySQL database.
- To implement clean architecture using OOP principles and modular design.
- To include exception handling for better error management.
- To perform unit testing for ensuring system reliability.
- To simplify administrative workload and improve team productivity.

3. SCOPE OF THE PROJECT

The **Project Management System** is designed to cater to small and medium-scale organizations that require a structured platform for managing multiple projects, tasks, and employees. The system ensures seamless coordination between team members and managers by enabling efficient task allocation, progress tracking, and project monitoring.

This system supports various functionalities such as adding new projects, assigning employees to tasks, setting deadlines, deleting records, and viewing comprehensive reports. It also focuses on real-time data management using MySQL and Python-based implementation with proper exception handling, modular design, and testing mechanisms.

4. STRUCTURE OF THE PROJECT

4.1 SQL STRUCTURE (DATABASE SCHEMA)

Database Creation:

CREATE DATABASE project_management_system;

USE project_management_system;

Table Creation:

1. Project Table

- id (INT, Primary Key, Auto Increment)
- project_name (VARCHAR(100), NOT NULL)
- description (TEXT)
- start_date (DATE)
- status (ENUM: 'started', 'dev', 'build', 'test', 'deployed', DEFAULT 'started')

SQL Query:

```
CREATE TABLE Project (
    id INT AUTO_INCREMENT PRIMARY KEY,
    project_name VARCHAR(100) NOT NULL,
    description TEXT,
    start_date DATE,
    status ENUM('started', 'dev', 'build', 'test', 'deployed') DEFAULT 'started'
);
```

2. Employee Table

- id (INT, Primary Key, Auto Increment)
- name (VARCHAR(100), NOT NULL)
- designation (VARCHAR(100))
- gender (CHAR(1))
- salary (DECIMAL(10,2))
- project_id (INT, Foreign Key references Project(id))

SQL Query:

```
CREATE TABLE Employee (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    designation VARCHAR(100),
    gender CHAR(1),
    salary DECIMAL(10,2),
    project_id INT,
    FOREIGN KEY (project_id) REFERENCES Project(id)
);
```

3. Task Table

- task_id (INT, Primary Key, Auto Increment)
- task_name (VARCHAR(100), NOT NULL)
- project_id (INT, Foreign Key references Project(id))
- employee_id (INT, Foreign Key references Employee(id))
- status (ENUM: 'Assigned', 'Started', 'Completed', DEFAULT 'Assigned')
- allocation_date (DATE)
- deadline_date (DATE)

SQL Query:

```
CREATE TABLE Task (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    task_name VARCHAR(100) NOT NULL,
    project_id INT,
```

```
    employee_id INT,
    status ENUM('Assigned', 'Started', 'Completed') DEFAULT 'Assigned',
    allocation_date DATE,
    deadline_date DATE,
    FOREIGN KEY (project_id) REFERENCES Project(id),
    FOREIGN KEY (employee_id) REFERENCES Employee(id)
);
```

4. Expenses Table

- expense_id (INT, Primary Key, Auto Increment)
- project_id (INT, Foreign Key references Project(id))
- employee_id (INT, Foreign Key references Employee(id))
- description (VARCHAR(255))
- amount (DECIMAL(10, 2))
- expense_date (DATE)

SQL Query:

```
CREATE TABLE expenses (
    expense_id INT PRIMARY KEY AUTO_INCREMENT,
    project_id INT,
    employee_id INT,
    description VARCHAR(255),
    amount DECIMAL(10, 2),
    expense_date DATE,
    FOREIGN KEY (project_id) REFERENCES Project(id),
    FOREIGN KEY (employee_id) REFERENCES Employee(id)
);
```

5. User Table

- id (INT, Primary Key, Auto Increment)
- employee_id (INT, Unique, Foreign Key references Employee(id))
- username (VARCHAR(50), NOT NULL, Unique)
- password (VARCHAR(100), NOT NULL)
- role (ENUM: 'admin', 'employee', DEFAULT 'employee')

SQL Query:

```
CREATE TABLE User (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employee_id INT UNIQUE,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(100) NOT NULL,
    role ENUM('admin', 'employee') NOT NULL DEFAULT 'employee',
    FOREIGN KEY (employee_id) REFERENCES Employee(id) ON DELETE CASCADE
);
```

Inserting Data:

1. Insert into Project

```
INSERT INTO Project (project_name, description, start_date, status) VALUES
('Quantum AI', 'AI-powered quantum computing research', '2025-01-10', 'started'),
('Autonomous Cars', 'Self-driving car software', '2025-02-15', 'dev'),
('Blockchain Banking', 'Secure banking on blockchain', '2025-03-01', 'build'),
('VR Metaverse', 'Virtual reality social platform', '2025-01-20', 'test'),
('Drone Delivery', 'AI-controlled delivery drones', '2025-02-05', 'deployed');
```

id	project_name	description	start_date	status
1	Quantum AI	AI-powered quantum computing research	2025-01-10	started
2	Autonomous Cars	Self-driving car software	2025-02-15	dev
3	Blockchain Banking	Secure banking on blockchain	2025-03-01	build
4	VR Metaverse	Virtual reality social platform	2025-01-20	test
5	Drone Delivery	AI-controlled delivery drones	2025-02-05	deployed
NULL	NULL	NULL	NULL	NULL

2. Insert into Employee

```
INSERT INTO Employee (name, designation, gender, salary, project_id) VALUES
('Maxine Verstappen', 'Lead Engineer', 'F', 150000, 1),
('Lewis Hamilton', 'UX Designer', 'M', 140000, 2),
('Charlotte Leclerc', 'Data Scientist', 'F', 130000, 3),
('Lana Norris', 'Frontend Dev', 'F', 120000, 4),
('Carlos Sainz', 'Backend Dev', 'M', 125000, 5),
('Georgina Russell', 'DevOps Engineer', 'F', 110000, 1),
('Fernando Alonso', 'Project Manager', 'M', 145000, 2),
```

('Oscar Piastri', 'Junior Developer', 'M', 95000, 3),

('Pierre Gasly', 'QA Tester', 'M', 100000, 4),

('Estelle Ocon', 'Database Admin', 'F', 105000, 5);

id	name	designation	gender	salary	project_id
1	Maxine Verstappen	Lead Engineer	F	150000.00	1
2	Lewis Hamilton	UX Designer	M	140000.00	2
3	Charlotte Leclerc	Data Scientist	F	130000.00	3
4	Lana Norris	Frontend Dev	F	120000.00	4
5	Carlos Sainz	Backend Dev	M	125000.00	2
6	Georgina Russell	DevOps Engineer	F	110000.00	1
7	Fernando Alonso	Project Manager	M	145000.00	2
8	Oscar Piastri	Junior Developer	M	95000.00	3
9	Pierre Gasly	QA Tester	M	100000.00	4
10	Estelle Ocon	Database Admin	F	105000.00	5
NULL	NULL	NULL	NULL	NULL	NULL

3. Insert into Task

INSERT INTO Task (task_name, project_id, employee_id, status, allocation_date, deadline_date) VALUES

('Design AI model', 1, 1, 'Started', '2025-01-15', '2025-03-20'),

('Build car sensors', 2, 2, 'Assigned', '2025-02-20', '2025-04-25'),

('Write smart contracts', 3, 3, 'Started', '2025-03-05', '2025-05-10'),

('Develop VR UI', 4, 4, 'Completed', '2025-01-25', '2025-02-28'),

('Test drone navigation', 5, 5, 'Started', '2025-02-10', '2025-04-15'),

('Optimize database', 5, 10, 'Assigned', '2025-02-12', '2025-03-30'),

('Fix API bugs', 2, 7, 'Started', '2025-02-18', '2025-04-05'),

('Train ML model', 1, 6, 'Assigned', '2025-01-20', '2025-03-25'),

('Test VR physics', 4, 9, 'Started', '2025-01-30', '2025-03-15'),

('Deploy blockchain', 3, 8, 'Assigned', '2025-03-10', '2025-05-01');

task_id	task_name	project_id	employee_id	status	allocation_date	deadline_date
1	Design AI model	1	1	Assigned	2025-01-15	2025-03-20
2	Build car sensors	2	2	Assigned	2025-02-20	2025-04-25
3	Write smart contracts	3	3	Started	2025-03-05	2025-05-10
4	Develop VR UI	4	4	Completed	2025-01-25	2025-02-28
5	Test drone navigation	5	5	Started	2025-02-10	2025-04-15
6	Optimize database	5	10	Assigned	2025-02-12	2025-03-30
7	Fix API bugs	2	7	Started	2025-02-18	2025-04-05
8	Train ML model	1	6	Assigned	2025-01-20	2025-03-25
9	Test VR physics	4	9	Started	2025-01-30	2025-03-15
10	Deploy blockchain	3	8	Assigned	2025-03-10	2025-05-01
11	Design UI Flow	2	3	Assigned	2025-05-02	2025-05-15
NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. Insert into Expenses

INSERT INTO expenses (project_id, employee_id, description, amount, expense_date) VALUES

(1, 1, 'Travel expenses for client meeting', 150.00, '2025-01-10'),

(1, 1, 'Client dinner', 200.00, '2025-01-15'),

(1, 1, 'Conference registration', 120.00, '2025-01-20'),

(1, 2, 'Accommodation for team building event', 300.00, '2025-03-01'),

(1, 2, 'Team transportation cost', 100.00, '2025-03-03'),

(2, 3, 'Software purchase for project', 500.00, '2025-02-15'),

(2, 3, 'Travel expenses for project implementation', 250.00, '2025-02-18'),

(2, 4, 'Client entertainment', 180.00, '2025-02-22'),

(2, 4, 'Consultant fee for project advisory', 1000.00, '2025-02-28'),

(3, 5, 'Travel expenses for development workshop', 220.00, '2025-01-05'),

(3, 5, 'Project-related travel insurance', 50.00, '2025-01-12'),

(4, 6, 'Purchasing hardware for project', 1500.00, '2025-03-10'),

(4, 6, 'Project testing expenses', 300.00, '2025-03-12'),

(5, 7, 'Project research and development', 350.00, '2025-01-25'),

(5, 7, 'Workshop expenses for project training', 400.00, '2025-02-05');

expense_id	project_id	employee_id	description	amount	expense_date
1	1	1	Travel expenses for client meeting	150.00	2025-01-10
2	1	1	Client dinner	200.00	2025-01-15
3	1	1	Conference registration	120.00	2025-01-20
4	1	2	Accommodation for team building event	300.00	2025-03-01
5	1	2	Team transportation cost	100.00	2025-03-03
6	2	3	Software purchase for project	500.00	2025-02-15
7	2	3	Travel expenses for project implementation	250.00	2025-02-18
8	2	4	Client entertainment	180.00	2025-02-22
9	2	4	Consultant fee for project advisory	1000.00	2025-02-28
10	3	5	Travel expenses for development workshop	220.00	2025-01-05
11	3	5	Project-related travel insurance	50.00	2025-01-12
12	4	6	Purchasing hardware for project	1500.00	2025-03-10
13	4	6	Project testing expenses	300.00	2025-03-12
14	5	7	Project research and development	350.00	2025-01-25
15	5	7	Workshop expenses for project training	400.00	2025-02-05
NULL	NULL	NULL	NULL	NULL	NULL

5. Insert into User

INSERT INTO User (username, password, role) VALUES

('admin1', 'admin123', 'admin'),

('admin2', 'securepass', 'admin');

INSERT INTO User (employee_id, username, password) VALUES

(1, 'maxinev', 'quantumai'),

(2, 'lewish', 'autodrive'),

(3, 'charlottel', 'blockchain'),

(4, 'lana_n', 'vrmeta'),

(5, 'carlos_s', 'dronedel'),

(6, 'georginar', 'devopsai'),

(7, 'fernandoa', 'pm2025'),

(8, 'oscarp', 'junior123'),

(9, 'pierreg', 'qatester'),

(10, 'estelleo', 'dbadmin');

id	employee_id	username	password	role
1	NULL	admin1	admin123	admin
2	NULL	admin2	securepass	admin
3	1	maxinev	quantumai	employee
4	2	lewish	autodrive	employee
5	3	charlottel	blockchain	employee
6	4	lana_n	vrmeta	employee
7	5	carlos_s	dronedel	employee
8	6	georginar	devopsai	employee
9	7	fernandoa	pm2025	employee
10	8	oscarp	junior123	employee
11	9	pierreg	qatester	employee
12	10	estelleo	dbadmin	employee
NULL	NULL	NULL	NULL	NULL

4.2. OOP STRUCTURE

1. Entity Module (entity/)

Responsibilities: Represents the data model for Employee, Project, and Task.

employee.py:

```
#entity/employee.py:
class Employee:
   def __init__(self, id=None, name=None, designation=None, gender=None, salary=None,
project_id=None):
       self.__id = id
       self.__name = name
       self.__designation = designation
       self.__gender = gender
       self.__salary = salary
```

```
        self.__project_id = project_id

    def get_id(self): return self.__id
    def get_name(self): return self.__name
    def get_designation(self): return self.__designation
    def get_gender(self): return self.__gender
    def get_salary(self): return self.__salary
    def get_project_id(self): return self.__project_id

    def set_id(self, id): self.__id = id
    def set_name(self, name): self.__name = name
    def set_designation(self, designation): self.__designation = designation
    def set_gender(self, gender): self.__gender = gender
    def set_salary(self, salary): self.__salary = salary
    def set_project_id(self, project_id): self.__project_id = project_id
```

project.py:

```
#entity/project.py:
class Project:
    def __init__(self, id=None, project_name=None, description=None, start_date=None,
status=None):
        self.__id = id
        self.__project_name = project_name
        self.__description = description
        self.__start_date = start_date
        self.__status = status

    def get_id(self): return self.__id
    def get_project_name(self): return self.__project_name
    def get_description(self): return self.__description
    def get_start_date(self): return self.__start_date
    def get_status(self): return self.__status

    def set_id(self, id): self.__id = id
    def set_project_name(self, project_name): self.__project_name = project_name
    def set_description(self, description): self.__description = description
    def set_start_date(self, start_date): self.__start_date = start_date
    def set_status(self, status): self.__status = status
```

task.py:

```
#entity/task.py:
class Task:
    def __init__(self, task_id=None, task_name=None, project_id=None, employee_id=None,
```

```
status=None, allocation_date=None, deadline_date=None):
        self.__task_id = task_id
        self.__task_name = task_name
        self.__project_id = project_id
        self.__employee_id = employee_id
        self.__status = status
        self.__allocation_date = allocation_date
        self.__deadline_date = deadline_date

    def get_task_id(self): return self.__task_id
    def get_task_name(self): return self.__task_name
    def get_project_id(self): return self.__project_id
    def get_employee_id(self): return self.__employee_id
    def get_status(self): return self.__status
    def get_allocation_date(self): return self.__allocation_date
    def get_deadline_date(self): return self.__deadline_date

    def set_task_id(self, task_id): self.__task_id = task_id
    def set_task_name(self, task_name): self.__task_name = task_name
    def set_project_id(self, project_id): self.__project_id = project_id
    def set_employee_id(self, employee_id): self.__employee_id = employee_id
    def set_status(self, status): self.__status = status
    def set_allocation_date(self, allocation_date): self.__allocation_date = allocation_date
    def set_deadline_date(self, deadline_date): self.__deadline_date = deadline_date
```

user.py:

```
#entity/user.py:
class User:
    def __init__(self, id=None, employee_id=None, username=None, password=None,
role='employee'):
        self.__id = id
        self.__employee_id = employee_id
        self.__username = username
        self.__password = password
        self.__role = role

    def get_id(self): return self.__id
    def get_employee_id(self): return self.__employee_id
    def get_username(self): return self.__username
    def get_password(self): return self.__password
    def get_role(self): return self.__role

    def set_id(self, id): self.__id = id
    def set_employee_id(self, employee_id): self.__employee_id = employee_id
    def set_username(self, username): self.__username = username
```

```
    def set_password(self, password): self.__password = password
    def set_role(self, role): self.__role = role
```

2. Employee Module(employee/)

Responsibilities: Manages employee tasks, expenses, and project access.

EmployeeModule.py:

```
# employee/EmployeeModule.py
from dao.ProjectRepositoryImpl import ProjectRepositoryImpl
from exception.EmployeeNotFoundException import EmployeeNotFoundException
from exception.ProjectNotFoundException import ProjectNotFoundException

class EmployeeModule:
    def __init__(self, employee_id):
        self.repository = ProjectRepositoryImpl()
        self.employee_id = employee_id
        self.employee_details = self.repository.get_employee_details(employee_id)
        self.project_id = self.employee_details['project_id'] if self.employee_details else None

    def display_menu(self):
        while True:
            print(f"\nEmployee Dashboard - {self.employee_details['name']}")
            print(f"Project: {self.employee_details['project_name'] or 'None'}")
            print("1. View my tasks")
            print("2. Update task status")
            print("3. View project details")
            print("4. View my expenses")
            print("5. Generate expense report")
            print("6. View team members")
            print("7. Logout")

            choice = input("Enter your choice: ")

            try:
                if choice == '1':
                    self.view_my_tasks()
                elif choice == '2':
                    self.update_task_status()
                elif choice == '3':
                    self.view_project_details()
                elif choice == '4':
                    self.view_my_expenses()
                elif choice == '5':
                    self.generate_expense_report()
                elif choice == '6':
```

```
                self.view_team_members()
            elif choice == '7':
                print("Logging out...")
                break
            else:
                print("Invalid choice. Please try again.")
        except Exception as e:
            print(f"Error: {str(e)}")

    def view_my_tasks(self):
        tasks = self.repository.get_all_tasks(self.employee_id, self.project_id)
        if tasks:
            print("\nYour Tasks:")
            print("-" * 80)
            print(f"{'ID':<5}{'Task Name':<30}{'Status':<15}{'Deadline':<12}")
            print("-" * 80)
            for task in tasks:

print(f"{task['task_id']:<5}{task['task_name']:<30}{task['status']:<15}{task['deadline_date']:
<12}")
        else:
            print("No tasks assigned to you.")

    def update_task_status(self):
        self.view_my_tasks()
        task_id = int(input("Enter task ID to update: "))
        new_status = input("Enter new status (Assigned/Started/Completed): ").capitalize()

        if new_status not in ['Assigned', 'Started', 'Completed']:
            print("Invalid status. Please use Assigned, Started, or Completed.")
            return

        try:
            cursor = self.repository.connection.cursor()
            query = "UPDATE Task SET status = %s WHERE task_id = %s AND employee_id =
%s"
            cursor.execute(query, (new_status, task_id, self.employee_id))
            if cursor.rowcount > 0:
                self.repository.connection.commit()
                print("Task status updated successfully!")
            else:
                print("Task not found or you don't have permission to update it.")
        except Exception as e:
            print(f"Error updating task: {str(e)}")

    def view_project_details(self):
```

```
        if not self.project_id:
            print("You're not assigned to any project.")
            return

        try:
            cursor = self.repository.connection.cursor(dictionary=True)
            query = "SELECT * FROM Project WHERE id = %s"
            cursor.execute(query, (self.project_id,))
            project = cursor.fetchone()

            if project:
                print("\nProject Details:")
                print(f"Name: {project['project_name']}")
                print(f"Description: {project['description']}")
                print(f"Start Date: {project['start_date']}")
                print(f"Status: {project['status']}")
            else:
                print("Project not found.")
        except Exception as e:
            print(f"Error retrieving project: {str(e)}")

    def view_my_expenses(self):
        try:
            cursor = self.repository.connection.cursor(dictionary=True)
            query = """
            SELECT expense_id, description, amount,
                DATE_FORMAT(expense_date, '%Y-%m-%d') as expense_date
            FROM expenses
            WHERE employee_id = %s
            ORDER BY expense_date DESC
            """
            cursor.execute(query, (self.employee_id,))
            expenses = cursor.fetchall()

            if expenses:
                print("\nYour Expenses:")
                print("-" * 80)
                print(f"{'ID':<5}{'Date':<17}{'Description':<40}{'Amount':<12}")
                print("-" * 80)
                for exp in expenses:

print(f"{exp['expense_id']:<5}{exp['expense_date']:<17}{exp['description']:<40}{exp['amou
nt']:<12}")
            else:
                print("No expenses found.")
        except Exception as e:
```

```
            print(f"Error retrieving expenses: {str(e)}")

    def generate_expense_report(self):
        start_date = input("Enter start date (YYYY-MM-DD): ")
        end_date = input("Enter end date (YYYY-MM-DD): ")

        try:
            cursor = self.repository.connection.cursor(dictionary=True)
            query = """
            SELECT * FROM expenses
            WHERE employee_id = %s
            AND expense_date BETWEEN %s AND %s
            ORDER BY expense_date
            """
            cursor.execute(query, (self.employee_id, start_date, end_date))
            expenses = cursor.fetchall()

            if expenses:
                total = sum(exp['amount'] for exp in expenses)
                print(f"\nExpense Report for {start_date} to {end_date}")
                print("-" * 80)
                for exp in expenses:
                    print(f"{exp['expense_date']}: {exp['description']} - ${exp['amount']:.2f}")
                print("-" * 80)
                print(f"Total Expenses: ${total:.2f}")
            else:
                print(f"No expenses found between {start_date} and {end_date}")
        except Exception as e:
            print(f"Error generating report: {str(e)}")

    def view_team_members(self):
        if not self.project_id:
            print("You're not assigned to any project.")
            return

        team = self.repository.get_team_members(self.project_id)
        if team:
            print("\nTeam Members:")
            print("-" * 40)
            print(f"{'ID':<5}{'Name':<20}{'Designation':<15}")
            print("-" * 40)
            for member in team:
                print(f"{member['id']:<5}{member['name']:<20}{member['designation']:<15}")
        else:
            print("No team members found or you're not assigned to a project.")
```

3. DAO Module (dao/)

Responsibilities: Interface and implementation for DB operations.

IProjectRepository.py:

```
#dao/IProjectRepository.py:
from abc import ABC, abstractmethod
from entity.employee import Employee
from entity.project import Project
from entity.task import Task

class IProjectRepository(ABC):
    @abstractmethod
    def create_employee(self, emp: Employee) -> bool: pass
    @abstractmethod
    def create_project(self, pj: Project) -> bool: pass
    @abstractmethod
    def create_task(self, task: Task) -> bool: pass
    @abstractmethod
    def assign_project_to_employee(self, project_id: int, employee_id: int) -> bool: pass
    @abstractmethod
    def assign_task_in_project_to_employee(self, task_id: int, project_id: int, employee_id:
int) -> bool: pass
    @abstractmethod
    def delete_employee(self, employee_id: int) -> bool: pass
    @abstractmethod
    def delete_project(self, project_id: int) -> bool: pass
    @abstractmethod
    def get_all_tasks(self, emp_id: int, project_id: int) -> list: pass
```

ProjectRepositoryImpl.py:

```
#dao/ProjectRepositoryImpl.py:
import mysql.connector
from dao.IProjectRepository import IProjectRepository
from entity.employee import Employee
from entity.project import Project
from entity.task import Task
from exception.EmployeeNotFoundException import EmployeeNotFoundException
from exception.ProjectNotFoundException import ProjectNotFoundException
from util.DBConnUtil import DBConnUtil
from util.DBPropertyUtil import DBPropertyUtil

class ProjectRepositoryImpl(IProjectRepository):
    def __init__(self):
        self.connection_string = DBPropertyUtil.get_connection_string("db.properties")
```

```
        self.connection = DBConnUtil.get_connection(self.connection_string)

    def __del__(self):
        if self.connection and self.connection.is_connected():
            self.connection.close()

    def create_employee(self, emp: Employee) -> bool:
        try:
            cursor = self.connection.cursor()
            query = """
            INSERT INTO Employee (name, designation, gender, salary, project_id)
            VALUES (%s, %s, %s, %s, %s)
            """
            values = (emp.get_name(), emp.get_designation(), emp.get_gender(),
emp.get_salary(), emp.get_project_id())
            cursor.execute(query, values)
            self.connection.commit()
            return True
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return False

    def create_project(self, pj: Project) -> bool:
        try:
            cursor = self.connection.cursor()
            query = """
            INSERT INTO Project (project_name, description, start_date, status)
            VALUES (%s, %s, %s, %s)
            """
            values = (pj.get_project_name(), pj.get_description(), pj.get_start_date(),
pj.get_status())
            cursor.execute(query, values)
            self.connection.commit()
            return True
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return False

    def create_task(self, task: Task) -> bool:
        try:
            cursor = self.connection.cursor()
            query = """
            INSERT INTO Task (task_name, project_id, employee_id, status, allocation_date,
deadline_date)
            VALUES (%s, %s, %s, %s, %s, %s)
            """
```

```
            values = (task.get_task_name(), task.get_project_id(), task.get_employee_id(),
task.get_status(),
                      task.get_allocation_date(), task.get_deadline_date())
            cursor.execute(query, values)
            self.connection.commit()
            return True
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return False

    def assign_project_to_employee(self, project_id: int, employee_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT id FROM Project WHERE id = %s", (project_id,))
            if not cursor.fetchone():
                raise ProjectNotFoundException(f"Project with ID {project_id} not found")

            cursor.execute("SELECT id FROM Employee WHERE id = %s", (employee_id,))
            if not cursor.fetchone():
                raise EmployeeNotFoundException(f"Employee with ID {employee_id} not
found")

            query = "UPDATE Employee SET project_id = %s WHERE id = %s"
            cursor.execute(query, (project_id, employee_id))
            self.connection.commit()
            return True
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return False

    def assign_task_in_project_to_employee(self, task_id: int, project_id: int, employee_id:
int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT id FROM Project WHERE id = %s", (project_id,))
            if not cursor.fetchone():
                raise ProjectNotFoundException(f"Project with ID {project_id} not found")

            cursor.execute("SELECT id FROM Employee WHERE id = %s", (employee_id,))
            if not cursor.fetchone():
                raise EmployeeNotFoundException(f"Employee with ID {employee_id} not
found")

            cursor.execute("SELECT task_id FROM Task WHERE task_id = %s", (task_id,))
            if not cursor.fetchone():
                raise Exception("Task not found")
```

```
            query = "UPDATE Task SET employee_id = %s WHERE task_id = %s AND
project_id = %s"
            cursor.execute(query, (employee_id, task_id, project_id))
            self.connection.commit()
            return True
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return False

    def delete_employee(self, employee_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT id FROM Employee WHERE id = %s", (employee_id,))
            if not cursor.fetchone():
                raise EmployeeNotFoundException(f"Employee with ID {employee_id} not
found")

            cursor.execute("UPDATE Task SET employee_id = NULL WHERE employee_id =
%s", (employee_id,))
            cursor.execute("DELETE FROM Employee WHERE id = %s", (employee_id,))
            self.connection.commit()
            return True
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return False

    def delete_project(self, project_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT id FROM Project WHERE id = %s", (project_id,))
            if not cursor.fetchone():
                raise ProjectNotFoundException(f"Project with ID {project_id} not found")

            cursor.execute("DELETE FROM Task WHERE project_id = %s", (project_id,))
            cursor.execute("UPDATE Employee SET project_id = NULL WHERE project_id =
%s", (project_id,))
            cursor.execute("DELETE FROM Project WHERE id = %s", (project_id,))
            self.connection.commit()
            return True
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return False

    def get_all_tasks(self, emp_id: int, project_id: int) -> list:
        try:
```

```
            cursor = self.connection.cursor(dictionary=True)
            cursor.execute("SELECT id FROM Employee WHERE id = %s", (emp_id,))
            if not cursor.fetchone():
                raise EmployeeNotFoundException(f"Employee with ID {emp_id} not found")

            cursor.execute("SELECT id FROM Project WHERE id = %s", (project_id,))
            if not cursor.fetchone():
                raise ProjectNotFoundException(f"Project with ID {project_id} not found")

            query = """
            SELECT t.task_id, t.task_name, t.status,
                DATE_FORMAT(t.allocation_date, '%Y-%m-%d') as allocation_date,
                DATE_FORMAT(t.deadline_date, '%Y-%m-%d') as deadline_date
            FROM Task t
            WHERE t.employee_id = %s AND t.project_id = %s
            """
            cursor.execute(query, (emp_id, project_id))
            tasks = cursor.fetchall()
            return tasks
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return []

    def get_expenses_by_date_range(self, start_date, end_date):
        cursor = self.connection.cursor(dictionary=True)
        query = """
        SELECT expense_id, project_id, employee_id, description, amount,
            DATE_FORMAT(expense_date, '%Y-%m-%d') as expense_date
        FROM expenses
        WHERE expense_date BETWEEN %s AND %s
        """
        cursor.execute(query, (start_date, end_date))
        expenses = cursor.fetchall()
        cursor.close()
        return expenses
    def authenticate_user(self, username, password):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM User WHERE username = %s AND password = %s"
            cursor.execute(query, (username, password))
            user = cursor.fetchone()
            return user
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return None
```

```
    def get_employee_details(self, employee_id):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = """
            SELECT e.*, p.project_name
            FROM Employee e
            LEFT JOIN Project p ON e.project_id = p.id
            WHERE e.id = %s
            """
            cursor.execute(query, (employee_id,))
            return cursor.fetchone()
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return None

    def get_team_members(self, project_id):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = """
            SELECT e.id, e.name, e.designation
            FROM Employee e
            WHERE e.project_id = %s
            """
            cursor.execute(query, (project_id,))
            return cursor.fetchall()
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return []

    def authenticate_admin(self, username, password):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM User WHERE username = %s AND password = %s AND
role = 'admin'"
            cursor.execute(query, (username, password))
            return cursor.fetchone()
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return None

    def authenticate_employee(self, username, password):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM User WHERE username = %s AND password = %s AND
role = 'employee'"
            cursor.execute(query, (username, password))
```

```
            return cursor.fetchone()
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            return None

    def authenticate_user(self, username, password):
        try:
            hashed_pw = hash_password(password)
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM User WHERE username = %s AND password = %s"
            cursor.execute(query, (username, hashed_pw))
            return cursor.fetchone()
        except mysql.connector.Error as err:
            logger.error(f"Authentication error: {err}")
            return None
```

4. Exception Module (exception/)

Responsibilities: Custom exception classes

EmployeeNotFoundException.py:

```
#exception/EmployeeNotFoundException.py:
class EmployeeNotFoundException(Exception):
    def __init__(self, message="Employee not found"):
        self.message = message
        super().__init__(self.message)
```

ProjectNotFoundException.py:

```
#exception/ProjectNotFoundException.py:
class ProjectNotFoundException(Exception):
    def __init__(self, message="Project not found"):
        self.message = message
        super().__init__(self.message)
```

5. Utility Module (util/)

Responsibilities: Database connection and properties utility

DBConnUtil.py:

```
#util/DBConnUtil.py:
import mysql.connector
from mysql.connector import Error
```

```
class DBConnUtil:
    @staticmethod
    def get_connection(connection_string):
        try:
            parts = connection_string.split('://')[1].split('@')
            user_pass = parts[0].split(':')
            host_port_db = parts[1].split('/')
            host_port = host_port_db[0].split(':')

            username = user_pass[0]
            password = user_pass[1] if len(user_pass) > 1 else ''
            host = host_port[0]
            port = host_port[1] if len(host_port) > 1 else '3306'
            database = host_port_db[1]

            connection = mysql.connector.connect(
                host=host,
                user=username,
                password=password,
                database=database,
                port=port
            )

            if connection.is_connected():
                print("Connected to MySQL database")
                return connection
        except Error as e:
            print(f"Error while connecting to MySQL: {e}")
            return None
```

DBPropertyUtil.py:

```
#util/DBPropertyUtil.py:
class DBPropertyUtil:
    @staticmethod
    def get_connection_string(property_file):
        try:
            with open(property_file, 'r') as file:
                properties = {}
                for line in file:
                    if '=' in line:
                        key, value = line.strip().split('=', 1)
                        properties[key.strip()] = value.strip()

                hostname = properties.get('hostname', 'localhost')
                dbname = properties.get('dbname', 'project_management_system')
```

```
            username = properties.get('username', 'root')
            password = properties.get('password', '')
            port = properties.get('port', '3306')

            return
f"mysql+pymysql://{username}:{password}@{hostname}:{port}/{dbname}"
        except FileNotFoundError:
            raise Exception("Property file not found")
        except Exception as e:
            raise Exception(f"Error reading property file: {str(e)}")
```

6. Main Application (main/)

Responsibilities: Menu-driven app to interact with the system.

MainModule.py:

```
#main/MainModule.py
from dao.ProjectRepositoryImpl import ProjectRepositoryImpl
from entity.employee import Employee
from entity.project import Project
from entity.task import Task
from exception.EmployeeNotFoundException import EmployeeNotFoundException
from exception.ProjectNotFoundException import ProjectNotFoundException
from employee.EmployeeModule import EmployeeModule

class MainModule:
    def __init__(self):
        self.repository = ProjectRepositoryImpl()
        self.current_user = None

    def display_main_menu(self):
        while True:
            print("\nPROJECT MANAGEMENT SYSTEM LOGIN")
            print("1. Admin Login")
            print("2. Employee Login")
            print("3. Exit")

            choice = input("Enter your choice (1-3): ")

            if choice == '1':
                self.admin_login()
            elif choice == '2':
                self.employee_login()
            elif choice == '3':
                print("Exiting system...")
                break
```

```
        else:
            print("Invalid choice. Please enter 1, 2, or 3.")

    def admin_login(self):
        print("\nADMIN LOGIN")
        username = input("Username: ")
        password = input("Password: ")

        user = self.repository.authenticate_admin(username, password)
        if user:
            self.current_user = user
            print(f"\nWelcome Admin {username}!")
            self.display_admin_menu()
        else:
            print("Invalid admin credentials.")

    def employee_login(self):
        print("\nEMPLOYEE LOGIN")
        username = input("Username: ")
        password = input("Password: ")

        user = self.repository.authenticate_employee(username, password)
        if user:
            self.current_user = user
            employee = self.repository.get_employee_details(user['employee_id'])
            if employee:
                print(f"\nWelcome {employee['name']}!")
                employee_module = EmployeeModule(user['employee_id'])
                employee_module.display_menu()
            else:
                print("Employee record not found. Please contact admin.")
        else:
            print("Invalid employee credentials.")

    def display_admin_menu(self):
        while True:
            print("\nAdmin Dashboard")
            print("1. Add Employee")
            print("2. Add Project")
            print("3. Add Task")
            print("4. Assign project to employee")
            print("5. Assign task within a project to employee")
            print("6. Delete Employee")
            print("7. Delete Project")
            print("8. List all tasks assigned to an employee in a project")
            print("9. Show all employees")
```

```
        print("10. Show all projects")
        print("11. Show all tasks")
        print("12. Generate Expense Report")
        print("13. Logout")

        choice = input("Enter your choice: ")

        try:
            if choice == '1':
                self.add_employee()
            elif choice == '2':
                self.add_project()
            elif choice == '3':
                self.add_task()
            elif choice == '4':
                self.assign_project_to_employee()
            elif choice == '5':
                self.assign_task_in_project_to_employee()
            elif choice == '6':
                self.delete_employee()
            elif choice == '7':
                self.delete_project()
            elif choice == '8':
                self.list_tasks_for_employee_in_project()
            elif choice == '9':
                self.show_all_employees()
            elif choice == '10':
                self.show_all_projects()
            elif choice == '11':
                self.show_all_tasks()
            elif choice == "12":
                self.generate_task_report_by_date()
            elif choice == '13':
                print("Logging out...")
                self.current_user = None
                break
            else:
                print("Invalid choice. Please try again.")
        except Exception as e:
            print(f"Error: {str(e)}")

    def add_employee(self):
        print("\nAdd New Employee")
        name = input("Enter employee name: ")
        designation = input("Enter designation: ")
        gender = input("Enter gender (M/F/O): ")
```

```
        salary = float(input("Enter salary: "))
        project_id = input("Enter project ID (leave empty if none): ")

        emp = Employee(
            name=name,
            designation=designation,
            gender=gender,
            salary=salary,
            project_id=int(project_id) if project_id else None
        )

        if self.repository.create_employee(emp):
            print("Employee created successfully!")
        else:
            print("Failed to create employee.")

    def add_project(self):
        print("\nAdd New Project")
        project_name = input("Enter project name: ")
        description = input("Enter description: ")
        start_date = input("Enter start date (YYYY-MM-DD): ")
        status = input("Enter status (started/dev/build/test/deployed): ")

        pj = Project(
            project_name=project_name,
            description=description,
            start_date=start_date,
            status=status
        )

        if self.repository.create_project(pj):
            print("Project created successfully!")
        else:
            print("Failed to create project.")

    def add_task(self):
        print("\nAdd New Task")
        task_name = input("Enter task name: ")
        project_id = int(input("Enter project ID: "))
        employee_id = input("Enter employee ID (leave empty if none): ")
        status = input("Enter status (Assigned/Started/Completed): ")
        allocation_date = input("Enter allocation date (YYYY-MM-DD): ")
        deadline_date = input("Enter deadline date (YYYY-MM-DD): ")

        task = Task(
            task_name=task_name,
```

```
            project_id=project_id,
            employee_id=int(employee_id) if employee_id else None,
            status=status,
            allocation_date=allocation_date,
            deadline_date=deadline_date
        )

        if self.repository.create_task(task):
            print("Task created successfully!")
        else:
            print("Failed to create task.")

    def assign_project_to_employee(self):
        print("\nAssign Project to Employee")
        project_id = int(input("Enter project ID: "))
        employee_id = int(input("Enter employee ID: "))

        try:
            if self.repository.assign_project_to_employee(project_id, employee_id):
                print("Project assigned successfully!")
        except EmployeeNotFoundException as e:
            print(f"Error: {str(e)}")
        except ProjectNotFoundException as e:
            print(f"Error: {str(e)}")
        except Exception as e:
            print(f"Error: {str(e)}")

    def assign_task_in_project_to_employee(self):
        print("\nAssign Task to Employee in Project")
        task_id = int(input("Enter task ID: "))
        project_id = int(input("Enter project ID: "))
        employee_id = int(input("Enter employee ID: "))

        try:
            if self.repository.assign_task_in_project_to_employee(task_id, project_id,
employee_id):
                print("Task assigned successfully!")
        except EmployeeNotFoundException as e:
            print(f"Error: {str(e)}")
        except ProjectNotFoundException as e:
            print(f"Error: {str(e)}")
        except Exception as e:
            print(f"Error: {str(e)}")

    def delete_employee(self):
        print("\nDelete Employee")
```

```
        employee_id = int(input("Enter employee ID to delete: "))

        try:
            if self.repository.delete_employee(employee_id):
                print("Employee deleted successfully!")
        except EmployeeNotFoundException as e:
            print(f"Error: {str(e)}")
        except Exception as e:
            print(f"Error: {str(e)}")

    def delete_project(self):
        print("\nDelete Project")
        project_id = int(input("Enter project ID to delete: "))

        try:
            if self.repository.delete_project(project_id):
                print("Project deleted successfully!")
        except ProjectNotFoundException as e:
            print(f"Error: {str(e)}")
        except Exception as e:
            print(f"Error: {str(e)}")

    def list_tasks_for_employee_in_project(self):
        print("\nList Tasks for Employee in Project")
        employee_id = int(input("Enter employee ID: "))
        project_id = int(input("Enter project ID: "))

        try:
            tasks = self.repository.get_all_tasks(employee_id, project_id)
            if tasks:
                print("\nTasks assigned to employee in project:")
                for task in tasks:
                    print(f"Task ID: {task['task_id']}, Name: {task['task_name']}, Status:
{task['status']}")
                    print(f"Allocation Date: {task['allocation_date']}, Deadline:
{task['deadline_date']}")
                    print("-" * 40)
            else:
                print("No tasks found for this employee in the specified project.")
        except EmployeeNotFoundException as e:
            print(f"Error: {str(e)}")
        except ProjectNotFoundException as e:
            print(f"Error: {str(e)}")
        except Exception as e:
            print(f"Error: {str(e)}")
```

```
    def show_all_employees(self):
        try:
            cursor = self.repository.connection.cursor(dictionary=True)
            query = "SELECT * FROM Employee ORDER BY name"
            cursor.execute(query)
            employees = cursor.fetchall()

            if employees:
                print("\nAll Employees:")
                print("-" * 80)

print(f"{'ID':<5}{'Name':<20}{'Designation':<20}{'Gender':<8}{'Salary':<14}{'Project
ID':<11}")
                print("-" * 80)
                for emp in employees:
                    print(

f"{emp['id']:<5}{emp['name']:<20}{emp['designation']:<20}{emp['gender']:<8}{emp['salary'
]:<14}{emp['project_id'] or 'None':<10}")
            else:
                print("No employees found.")
        except Exception as e:
            print(f"Error retrieving employees: {str(e)}")
        finally:
            if cursor:
                cursor.close()

    def show_all_projects(self):
        try:
            cursor = self.repository.connection.cursor(dictionary=True)
            query = "SELECT * FROM Project ORDER BY start_date"
            cursor.execute(query)
            projects = cursor.fetchall()

            if projects:
                print("\nAll Projects:")
                print("-" * 100)
                print(f"{'ID':<5}{'Name':<22}{'Description':<45}{'Start Date':<15}{'Status':<12}")
                print("-" * 100)
                for proj in projects:
                    print(
                        f"{proj['id']:<5}{proj['project_name']:<22}{proj['description'][:45] +
'...':<45}{str(proj['start_date']):<15}{proj['status']:<12}")
            else:
                print("No projects found.")
        except Exception as e:
```

```
            print(f"Error retrieving projects: {str(e)}")
        finally:
            if cursor:
                cursor.close()

    def show_all_tasks(self):
        try:
            cursor = self.repository.connection.cursor(dictionary=True)
            query = """
            SELECT t.task_id, t.task_name, p.project_name, e.name as employee_name,
                   t.status, t.allocation_date, t.deadline_date
            FROM Task t
            LEFT JOIN Project p ON t.project_id = p.id
            LEFT JOIN Employee e ON t.employee_id = e.id
            ORDER BY t.deadline_date
            """
            cursor.execute(query)
            tasks = cursor.fetchall()

            if tasks:
                print("\nAll Tasks:")
                print("-" * 120)
                print(
                    f"{'ID':<5}{'Task Name':<25}{'Project':<20}{'Assigned
To':<20}{'Status':<12}{'Allocated':<12}{'Deadline':<12}")
                print("-" * 120)
                for task in tasks:
                    print(f"{task['task_id']:<5}{task['task_name']:<25}{task['project_name']:<20}"
                          f"{task['employee_name'] or 'Unassigned':<20}{task['status']:<12}"
                          f"{str(task['allocation_date']):<12}{str(task['deadline_date']):<12}")
            else:
                print("No tasks found.")
        except Exception as e:
            print(f"Error retrieving tasks: {str(e)}")
        finally:
            if cursor:
                cursor.close()

    def generate_task_report_by_date(self):
        print("\nGenerate Expense Report for Time Period")

        start_date = input("Enter start date (YYYY-MM-DD): ")
        end_date = input("Enter end date (YYYY-MM-DD): ")

        expenses = self.repository.get_expenses_by_date_range(start_date, end_date)
```

```
        grand_total = 0

        if expenses:
            print("\n--- Expense Report ---")
            for expense in expenses:
                print(f"Expense ID: {expense['expense_id']}")
                print(f"Project ID: {expense['project_id']}")
                print(f"Employee ID: {expense['employee_id']}")
                print(f"Description: {expense['description']}")
                print(f"Amount: {expense['amount']}")
                print(f"Expense Date: {expense['expense_date']}")
                print("-" * 30)
                grand_total += expense['amount']

            print(f"\nGrand Total for Expenses from {start_date} to {end_date}:
{grand_total:.2f}")
        else:
            print("No expenses found for the given date range.")
            print(
                f"Grand Total for Expenses from {start_date} to {end_date}: {grand_total:.2f}")

if __name__ == "__main__":
    app = MainModule()
    app.display_main_menu()
```

7. Testing Module (test/)

Responsibilities: Unit testing for key functionalities

test_project_management.py:

```
#tests/test_project_management.py:
import unittest
from unittest.mock import MagicMock, patch
from entity.employee import Employee
from entity.project import Project
from entity.task import Task
from dao.ProjectRepositoryImpl import ProjectRepositoryImpl
from exception.EmployeeNotFoundException import EmployeeNotFoundException
from exception.ProjectNotFoundException import ProjectNotFoundException
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

class TestProjectManagementSystem(unittest.TestCase):

    def setUp(self):
```

```
        self.mock_connection = MagicMock()
        self.mock_cursor = MagicMock()
        self.mock_connection.cursor.return_value = self.mock_cursor

        patcher = patch('dao.ProjectRepositoryImpl.DBConnUtil.get_connection',
return_value=self.mock_connection)
        self.addCleanup(patcher.stop)
        self.mock_get_connection = patcher.start()

        self.repository = ProjectRepositoryImpl()

    # 1. Test if employee is created successfully
    def test_create_employee_successfully(self):
        emp = Employee("John Doe", "Developer", "M", 50000, 1)
        self.mock_cursor.execute.return_value = None
        self.mock_connection.commit.return_value = None

        result = self.repository.create_employee(emp)

        self.assertTrue(result)
        self.mock_cursor.execute.assert_called_once()
        self.mock_connection.commit.assert_called_once()

    # 2. Test if task is created successfully
    def test_create_task_successfully(self):
        task = Task("Fix Bug", 1, 1, "Assigned", "2025-01-01", "2025-01-10")
        self.mock_cursor.execute.return_value = None
        self.mock_connection.commit.return_value = None

        result = self.repository.create_task(task)

        self.assertTrue(result)
        self.mock_cursor.execute.assert_called_once()
        self.mock_connection.commit.assert_called_once()

    # 3. Test search for projects and tasks assigned to employee
    def test_get_all_tasks_for_employee_in_project(self):
        employee_id = 1
        project_id = 1

        mock_data = [
            {'task_id': 1, 'task_name': 'Task A', 'status': 'Assigned',
             'allocation_date': '2025-01-01', 'deadline_date': '2025-01-10'},
            {'task_id': 2, 'task_name': 'Task B', 'status': 'Started',
             'allocation_date': '2025-01-02', 'deadline_date': '2025-01-12'},
        ]
```

```
        self.mock_cursor.fetchall.return_value = mock_data
        result = self.repository.get_all_tasks(employee_id, project_id)

        self.assertEqual(len(result), 2)
        self.assertEqual(result[0]['task_name'], 'Task A')
        self.assertEqual(result[1]['status'], 'Started')

    # 4. Test if the exceptions are thrown correctly
    def test_assign_project_to_nonexistent_employee_raises_exception(self):
        project_id = 1
        employee_id = 999

        # Simulate: project exists, but employee does not
        self.mock_cursor.fetchone.side_effect = [[1],None]  # First call (project check) returns
project found; second call (employee check) returns None

        with self.assertRaises(EmployeeNotFoundException):
            self.repository.assign_project_to_employee(project_id, employee_id)

    def test_assign_task_to_nonexistent_project_raises_exception(self):
        self.mock_cursor.fetchone.return_value = None  # No such project

        with self.assertRaises(ProjectNotFoundException):
            self.repository.assign_task_in_project_to_employee(task_id=1, project_id=999,
employee_id=1)

    def test_delete_nonexistent_employee_raises_exception(self):
        self.mock_cursor.fetchone.return_value = None  # No such employee

        with self.assertRaises(EmployeeNotFoundException):
            self.repository.delete_employee(employee_id=999)

    def test_delete_project_successfully(self):
        self.mock_cursor.fetchone.return_value = [1]  # Project exists
        self.mock_cursor.execute.return_value = None
        self.mock_connection.commit.return_value = None

        result = self.repository.delete_project(project_id=1)

        self.assertTrue(result)
        self.mock_cursor.execute.assert_called()
        self.mock_connection.commit.assert_called()

    class TestAuthentication(unittest.TestCase):
        def setUp(self):
```

```
        self.mock_connection = MagicMock()
        self.mock_cursor = MagicMock()
        self.mock_connection.cursor.return_value = self.mock_cursor
        self.repository = ProjectRepositoryImpl()
        self.repository.connection = self.mock_connection

    def test_admin_authentication_success(self):
        mock_user = {'id': 1, 'username': 'admin1', 'role': 'admin'}
        self.mock_cursor.fetchone.return_value = mock_user

        result = self.repository.authenticate_user('admin1', 'admin123')
        self.assertEqual(result, mock_user)

    def test_employee_authentication_success(self):
        mock_user = {'id': 2, 'username': 'maxinev', 'role': 'employee', 'employee_id': 1}
        self.mock_cursor.fetchone.return_value = mock_user

        result = self.repository.authenticate_user('maxinev', 'quantumai')
        self.assertEqual(result, mock_user)

    def test_authentication_failure(self):
        self.mock_cursor.fetchone.return_value = None

        result = self.repository.authenticate_user('wrong', 'credentials')
        self.assertIsNone(result)

    def test_get_employee_details(self):
        mock_employee = {'id': 1, 'name': 'Maxine Verstappen', 'project_name': 'Quantum AI'}
        self.mock_cursor.fetchone.return_value = mock_employee

        result = self.repository.get_employee_details(1)
        self.assertEqual(result, mock_employee)

    def test_admin_login_success(self):
        mock_admin = {'id': 1, 'username': 'superadmin', 'role': 'admin'}
        self.mock_cursor.fetchone.return_value = mock_admin

        result = self.repository.authenticate_admin('superadmin', 'admin@123')
        self.assertEqual(result, mock_admin)

    def test_admin_login_fail_wrong_role(self):
        mock_user = {'id': 2, 'username': 'maxinev', 'role': 'employee'}
        self.mock_cursor.fetchone.return_value = mock_user

        result = self.repository.authenticate_admin('maxinev', 'quantumai')
        self.assertIsNone(result)
```

```
    def test_employee_login_success(self):
        mock_emp = {'id': 3, 'username': 'lewish', 'role': 'employee', 'employee_id': 2}
        self.mock_cursor.fetchone.return_value = mock_emp

        result = self.repository.authenticate_employee('lewish', 'autodrive')
        self.assertEqual(result, mock_emp)

    def test_employee_login_fail_wrong_role(self):
        mock_user = {'id': 1, 'username': 'superadmin', 'role': 'admin'}
        self.mock_cursor.fetchone.return_value = mock_user

        result = self.repository.authenticate_employee('superadmin', 'admin@123')
        self.assertIsNone(result)

if __name__ == '__main__':
    unittest.main()
```

8. db.properties:

```
hostname=localhost
dbname=project_management_system
username=root
password=root
port=3306
```

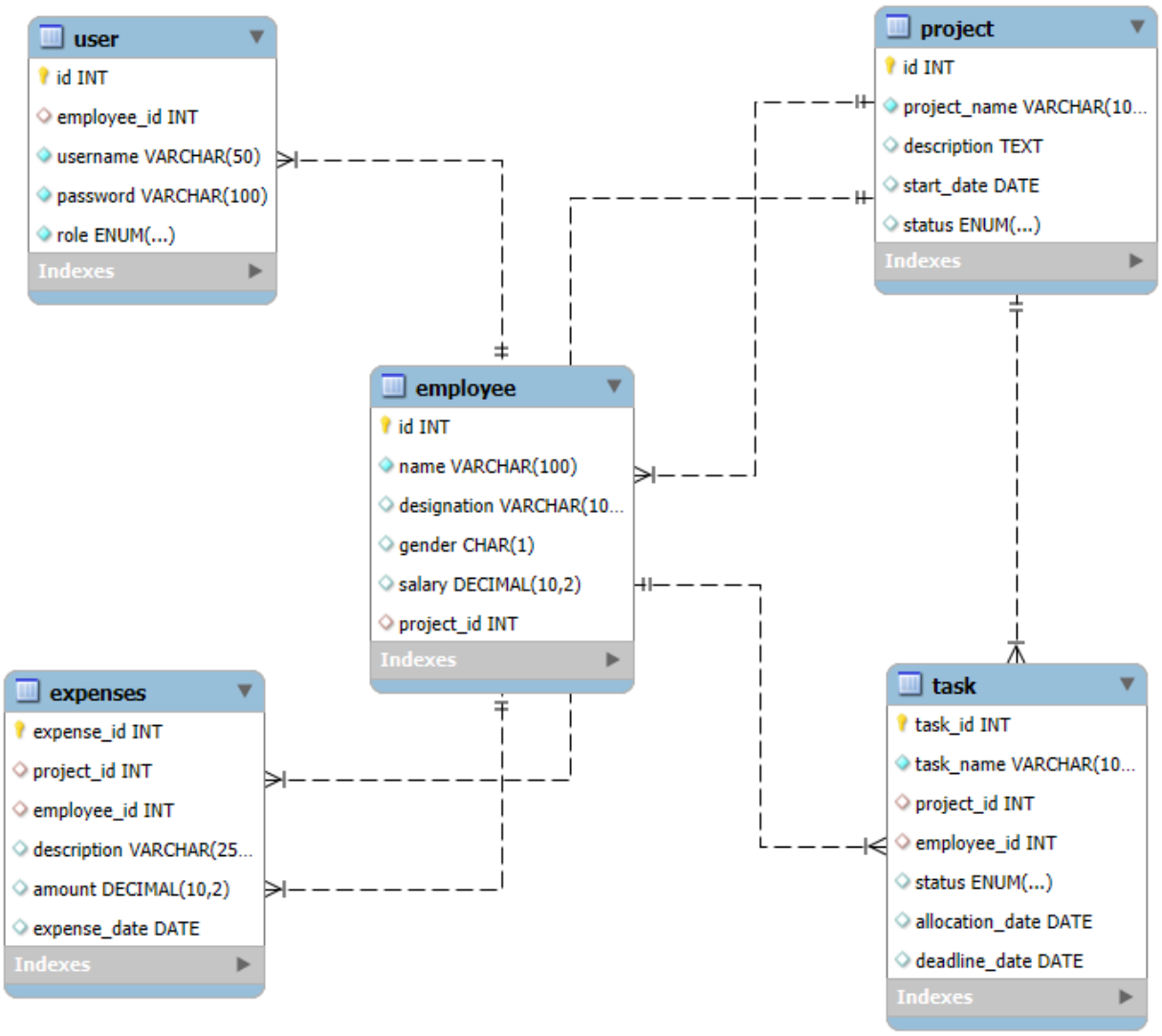
5. TECHNOLOGIES USED

The following technologies and tools were utilized for the development and implementation of the Project Management System:

1. **MySQL** - Used as the **Relational Database Management System (RDBMS)** to store and manage structured data related to employees, projects, and tasks.
2. **PyCharm** - An **Integrated Development Environment (IDE)** for Python programming. Used for writing, debugging, and running the Python code for the project.
3. **GitHub** - A **version control platform** used to manage and host the project's source code.

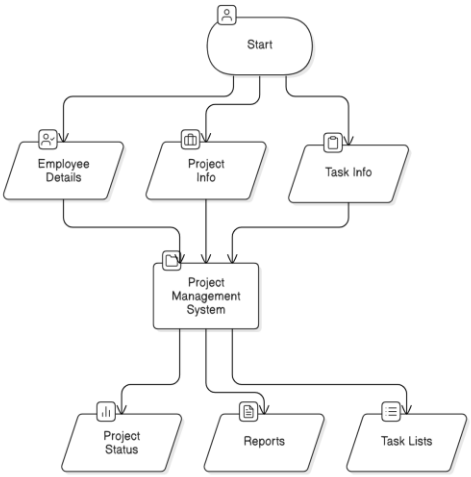
6. ER DIAGRAM

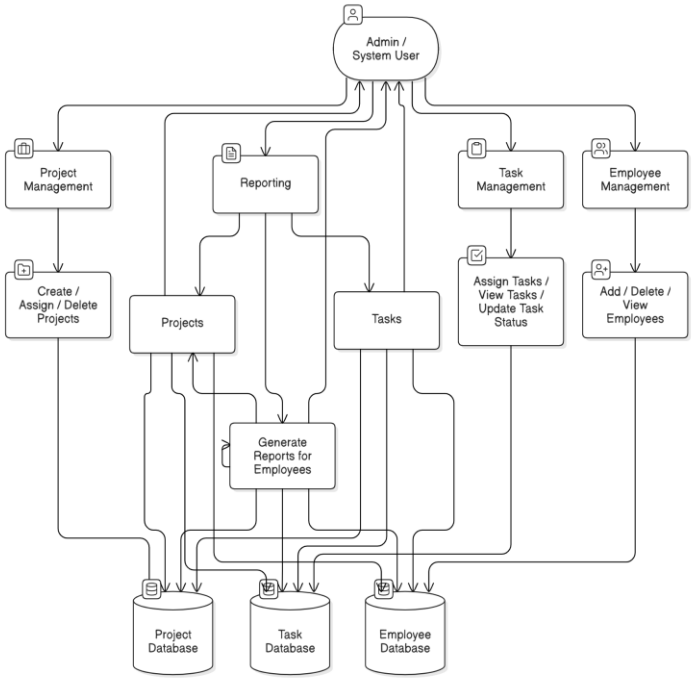
The Entity-Relationship (ER) Diagram represents the logical structure of the database used in the Project Management System. It defines the major entities involved, their key attributes, and the relationships between them.



7. DATA FLOW DIAGRAM

The Data Flow Diagram (DFD) visually represents how data moves through the Project Management System. It outlines the inputs, processes, storage, and outputs of the system.





8. OUTPUT

8.1 Login Menu Outputs

1. Main Login

```
PROJECT MANAGEMENT SYSTEM LOGIN
1. Admin Login
2. Employee Login
3. Exit
Enter your choice (1-3):
```

2. Admin Login

```
Enter your choice (1-3): 1

ADMIN LOGIN
Username: admin1
Password: admin123

Welcome Admin admin1!
```

3. Employee Login

```
Enter your choice (1-3): 2

EMPLOYEE LOGIN
Username: carlos_s
Password: dronedel

Welcome Carlos Sainz!
```

8.2 Admin Module

1. Admin Dashboard

```
Welcome Admin admin1!

Admin Dashboard
1. Add Employee
2. Add Project
3. Add Task
4. Assign project to employee
5. Assign task within a project to employee
6. Delete Employee
7. Delete Project
8. List all tasks assigned to an employee in a project
9. Show all employees
10. Show all projects
11. Show all tasks
12. Generate Expense Report
13. Logout
Enter your choice:
```

2. Add Employee

```
Enter your choice: 1

Add New Employee
Enter employee name: Lance Stroll
Enter designation: AI Engineer
Enter gender (M/F/O): M
Enter salary: 700000
Enter project ID (leave empty if none): 1
Employee created successfully!
```

3. Add Project

```
Enter your choice: 2

Add New Project
Enter project name: AI automation
Enter description: Automation of testing
Enter start date (YYYY-MM-DD): 2025-02-01
Enter status (started/dev/build/test/deployed): started
Project created successfully!
```

4. Add Task

```
Enter your choice: 3

Add New Task
Enter task name: Automation using AI
Enter project ID: 1
Enter employee ID (leave empty if none): 1
Enter status (Assigned/Started/Completed): Assigned
Enter allocation date (YYYY-MM-DD): 2025-02-20
Enter deadline date (YYYY-MM-DD): 2025-02-25
Task created successfully!
```

5. Assign Project to Employee

```
Enter your choice: 4

Assign Project to Employee
Enter project ID: 1
Enter employee ID: 1
Project assigned successfully!
```

6. Assign Task to Employee

```
Enter your choice: 5

Assign Task to Employee in Project
Enter task ID: 1
Enter project ID: 1
Enter employee ID: 1
Task assigned successfully!
```

7. Delete Employee

```
Enter your choice: 6

Delete Employee
Enter employee ID to delete: 10
Employee deleted successfully!
```

8. Delete Project

```
Enter your choice: 7

Delete Project
Enter project ID to delete: 8
Project deleted successfully!
```

9. List Employee Tasks

```
Enter your choice: 8

List Tasks for Employee in Project
Enter employee ID: 9
Enter project ID: 4

Tasks assigned to employee in project:
Task ID: 9, Name: Test VR physics, Status: Started
Allocation Date: 2025-01-30, Deadline: 2025-03-15
---------------------------------------
```

10. Show All Employees

```
Enter your choice: 9

All Employees:
-------------------------------------------------------------------------------
ID   Name               Designation        Gender  Salary       Project ID
-------------------------------------------------------------------------------
5    Carlos Sainz       Backend Dev        M       125000.00    5
3    Charlotte Leclerc  Data Scientist     F       130000.00    3
10   Estelle Ocon       Database Admin     F       105000.00    5
7    Fernando Alonso    Project Manager    M       145000.00    2
6    Georgina Russell   DevOps Engineer    F       110000.00    1
4    Lana Norris        Frontend Dev       F       120000.00    4
2    Lewis Hamilton     UX Designer        M       140000.00    2
1    Maxine Verstappen  Lead Engineer      F       150000.00    1
8    Oscar Piastri      Junior Developer   M       95000.00     3
9    Pierre Gasly       QA Tester          M       100000.00    4
```

11. Show All Projects

```
Enter your choice: 10

All Projects:
-----------------------------------------------------------------------------------------------
ID   Name               Description                                 Start Date    Status
-----------------------------------------------------------------------------------------------
1    Quantum AI         AI-powered quantum computing research...    2025-01-10    started
4    VR Metaverse       Virtual reality social platform...          2025-01-20    test
5    Drone Delivery     AI-controlled delivery drones...            2025-02-05    deployed
2    Autonomous Cars    Self-driving car software...                2025-02-15    dev
3    Blockchain Banking Secure banking on blockchain...             2025-03-01    build
```

12. Show All Tasks

```
Enter your choice: 11

All Tasks:
-----------------------------------------------------------------------------------------------------------
ID   Task Name          Project           Assigned To       Status     Allocated   Deadline
-----------------------------------------------------------------------------------------------------------
4    Develop VR UI      VR Metaverse      Lana Norris       Completed  2025-01-25  2025-02-28
9    Test VR physics    VR Metaverse      Pierre Gasly      Started    2025-01-30  2025-03-15
1    Design AI model    Quantum AI        Maxine Verstappen Started    2025-01-15  2025-03-20
8    Train ML model     Quantum AI        Georgina Russell  Assigned   2025-01-20  2025-03-25
6    Optimize database  Drone Delivery    Estelle Ocon      Assigned   2025-02-12  2025-03-30
7    Fix API bugs       Autonomous Cars   Fernando Alonso   Started    2025-02-18  2025-04-05
5    Test drone navigation Drone Delivery Carlos Sainz      Started    2025-02-10  2025-04-15
2    Build car sensors  Autonomous Cars   Lewis Hamilton    Assigned   2025-02-20  2025-04-25
10   Deploy blockchain  Blockchain Banking Oscar Piastri    Assigned   2025-03-10  2025-05-01
3    Write smart contracts Blockchain Banking Charlotte Leclerc Started 2025-03-05  2025-05-10
```

13. Generate Expense Report

```
Enter your choice: 12

Generate Expense Report for Time Period
Enter start date (YYYY-MM-DD): 2025-01-01
Enter end date (YYYY-MM-DD): 2025-01-10

--- Expense Report ---
Expense ID: 1
Project ID: 1
Employee ID: 1
Description: Travel expenses for client meeting
Amount: 150.00
Expense Date: 2025-01-10
------------------------------
Expense ID: 10
Project ID: 3
Employee ID: 5
Description: Travel expenses for development workshop
Amount: 220.00
Expense Date: 2025-01-05
------------------------------

Grand Total for Expenses from 2025-01-01 to 2025-01-10: 370.00
```

14. Admin Logout

```
Enter your choice: 13
Logging out...
```

8.3 Employee Module

1. Employee Dashboard

```
Welcome Carlos Sainz!
Connected to MySQL database

Employee Dashboard - Carlos Sainz
Project: Drone Delivery
1. View my tasks
2. Update task status
3. View project details
4. View my expenses
5. Generate expense report
6. View team members
7. Logout
Enter your choice:
```

2. View My Tasks

```
Enter your choice: 1

Your Tasks:
------------------------------------------------------------------
ID   Task Name                    Status         Deadline
------------------------------------------------------------------
5    Test drone navigation        Started        2025-04-15
```

3. Update Task Status

```
Enter your choice: 2

Your Tasks:
------------------------------------------------------------------
ID   Task Name                    Status         Deadline
------------------------------------------------------------------
5    Test drone navigation        Started        2025-04-15
Enter task ID to update: 5
Enter new status (Assigned/Started/Completed): Completed
Task status updated successfully!
```

4. View Project Details

```
Enter your choice: 3

Project Details:
Name: Drone Delivery
Description: AI-controlled delivery drones
Start Date: 2025-02-05
Status: deployed
```

5. View My Expenses

```
Enter your choice: 4

Your Expenses:
--------------------------------------------------------------------------------
ID   Date            Description                          Amount
--------------------------------------------------------------------------------
11   2025-01-12      Project-related travel insurance       50.00
10   2025-01-05      Travel expenses for development workshop220.00
```

6. Generate Expense Report

```
Enter your choice: 5
Enter start date (YYYY-MM-DD): 2025-01-01
Enter end date (YYYY-MM-DD): 2025-01-31

Expense Report for 2025-01-01 to 2025-01-31
------------------------------------------------------------------------
2025-01-05: Travel expenses for development workshop - $220.00
2025-01-12: Project-related travel insurance - $50.00
------------------------------------------------------------------------
Total Expenses: $270.00
```

7. View Team Members

```
Enter your choice: 6

Team Members:
----------------------------------------
ID   Name                 Designation
----------------------------------------
5    Carlos Sainz         Backend Dev
10   Estelle Ocon         Database Admin
```

8. Employee Logout

```
Enter your choice: 7
Logging out...
```

8.4 Unit Testing

```
PS C:\Users\Lenovo\PycharmProjects\Case-Study-Project-Management-System
\project-management> python -m tests.test_project_management
.......
----------------------------------------------------------------------
Ran 7 tests in 0.020s
OK
```

8.5 Exception Handling

1. Employee Not Found Error

```
Assign Project to Employee
Enter project ID: 1
Enter employee ID: 11
Error: Employee with ID 11 not found
```

2. Project Not Found Error

```
Assign Project to Employee
Enter project ID: 11
Enter employee ID: 1
Error: Project with ID 11 not found
```

8.6 Invalid Login

1. Admin Invalid Login

```
ADMIN LOGIN
Username: admin
Password: admin00
Invalid admin credentials.
```

2. Employee Invalid Login

```
EMPLOYEE LOGIN
Username: Lance Stroll
Password: Lance00
Invalid employee credentials.
```

9. FUTURE ENHANCEMENTS

- Add front-end using Flask/Django
- Export reports to PDF/Excel
- REST API integration

10. CONCLUSION

This Project Management System showcases practical implementation of object-oriented principles, SQL integration, and modular design. It provides a real-time, persistent solution for managing employees, projects, and tasks within any team or organization.