# Project Management System

**Project Title:** Project Management System
**Submitted By:** Keerthika Nagarajan
**Date:** 07/04/2025

## 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)
- **projectName** (VARCHAR)
- **description** (TEXT)
- **startDate** (DATE)
- **status** (ENUM: 'started', 'dev', 'build', 'test', 'deployed')

**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)
- **name** (VARCHAR)
- **designation** (VARCHAR)
- **gender** (VARCHAR)
- **salary** (DECIMAL)
- **project_id** (INT, Foreign Key)

**Description:** Stores information about employees and the projects they are assigned to.

**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)
- **task_name** (VARCHAR)
- **project_id** (INT, Foreign Key)
- **employee_id** (INT, Foreign Key)
- **status** (ENUM: 'Assigned', 'Started', 'Completed')

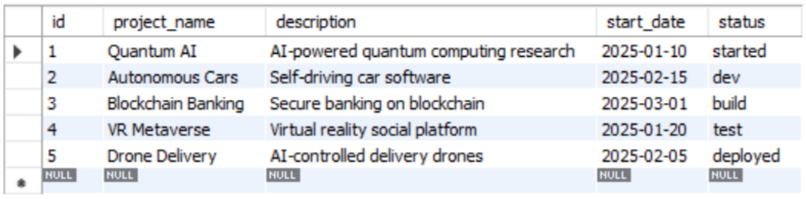
**SQL Query:**

```
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)
);
```

## Inserting Sample Data:

### 1. Insert into Project

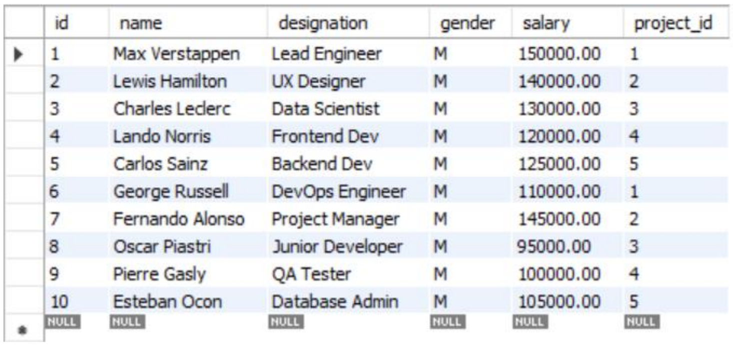
```
INSERT INTO Project (project_name, description, start_date, status) VALUES
('Quantum AI', 'AI-powered quantum computing research', '2025-01-10', 'started'),
('Autonomous Cars', 'Self-driving car software', '2025-02-15', 'dev'),
('Blockchain Banking', 'Secure banking on blockchain', '2025-03-01', '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
('Max Verstappen', 'Lead Engineer', 'M', 150000, 1),
('Lewis Hamilton', 'UX Designer', 'M', 140000, 2),
('Charles Leclerc', 'Data Scientist', 'M', 130000, 3),
('Lando Norris', 'Frontend Dev', 'M', 120000, 4),
('Carlos Sainz', 'Backend Dev', 'M', 125000, 5),
('George Russell', 'DevOps Engineer', 'M', 110000, 1),
('Fernando Alonso', 'Project Manager', 'M', 145000, 2),
('Oscar Piastri', 'Junior Developer', 'M', 95000, 3),
('Pierre Gasly', 'QA Tester', 'M', 100000, 4),
('Esteban Ocon', 'Database Admin', 'M', 105000, 5);
```



| id | name | designation | gender | salary | project_id |
|---|---|---|---|---|---|
| 1 | Max Verstappen | Lead Engineer | M | 150000.00 | 1 |
| 2 | Lewis Hamilton | UX Designer | M | 140000.00 | 2 |
| 3 | Charles Leclerc | Data Scientist | M | 130000.00 | 3 |
| 4 | Lando Norris | Frontend Dev | M | 120000.00 | 4 |
| 5 | Carlos Sainz | Backend Dev | M | 125000.00 | 5 |
| 6 | George Russell | DevOps Engineer | M | 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 | Esteban Ocon | Database Admin | M | 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'),

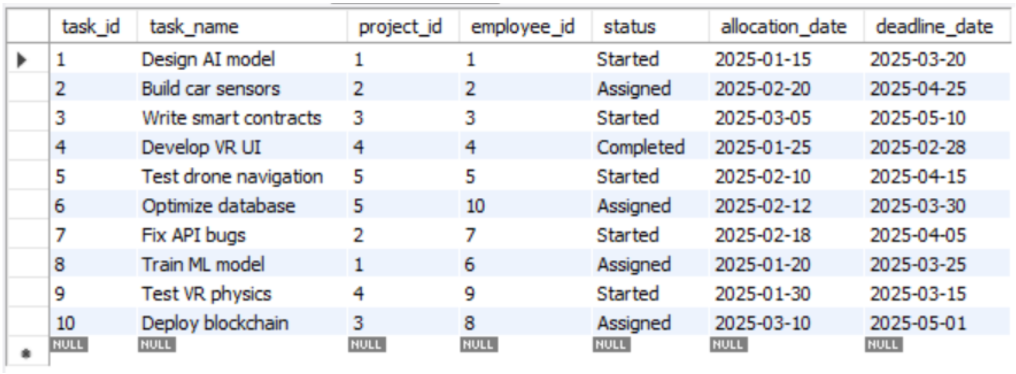
('Optimize database', 5, 10, 'Assigned', '2025-02-12', '2025-03-30'),

('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 | Started | 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 |
| NULL | NULL | 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:**

```
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

    # Getters
    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

    # Setters
    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:**

```
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

    # Getters
    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

    # Setters
    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:**

```
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

    # Getters
    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

    # Setters
    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
```

## 2. DAO Module (dao/)

**Responsibilities:** Interface and implementation for DB operations.

**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:**

```
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, t.allocation_date, t.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 []
```

**3. Exception Module (exception/)**

**Responsibilities:** Custom exception classes

**EmployeeNotFoundException.py:**

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

**ProjectNotFoundException.py:**

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

**4. Utility Module (util/)**

**Responsibilities:** Database connection and properties utility

**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:**

```
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)}")
```

**5. Main Application (main/)**

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

**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


class MainModule:
    def __init__(self):
        self.repository = ProjectRepositoryImpl()

    def display_menu(self):
        while True:
            print("\nProject Management System (2025)")
            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. Exit")

            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':
                print("Exiting...")
                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':<10}{'Project
ID':<10}")
                print("-" * 80)
                for emp in employees:
                    print(

f"{emp['id']:<5}{emp['name']:<20}{emp['designation']:<20}{emp['gender']:<8}{emp['salary'
]:<10} {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 (2025):")
                print("-" * 100)
                print(f"{'ID':<5} {'Name':<20} {'Description':<30} {'Start Date':<12} {'Status':<10}")
                print("-" * 100)
                for proj in projects:
                    print(
                        f"{proj['id']:<5} {proj['project_name']:<20} {proj['description'][:27] +
'...':<30} {str(proj['start_date']):<12} {proj['status']:<10}")
            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()


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

**6. Testing Module (test/)**

**Responsibilities:** Unit testing for key functionalities

**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


class TestProjectManagementSystem(unittest.TestCase):

    def setUp(self):
        # Create a mock database connection for testing
        self.mock_connection = MagicMock()
        self.mock_cursor = MagicMock()
        self.mock_connection.cursor.return_value = self.mock_cursor
        # Patch the DBConnUtil to return our mock connection
        self.patcher = patch('dao.ProjectRepositoryImpl.DBConnUtil.get_connection')
        self.mock_get_connection = self.patcher.start()
        self.mock_get_connection.return_value = self.mock_connection

        # Create repository instance
        self.repository = ProjectRepositoryImpl()
        self.repository.connection = self.mock_connection

    def tearDown(self):
        self.patcher.stop()

    # Test Case 1: Test if employee is created successfully
    def test_create_employee_successfully(self):
        # Setup
        emp = Employee(
            name="Test Employee",
            designation="Developer",
            gender="M",
            salary=50000,
            project_id=1
        )

        # Mock database response
        self.mock_cursor.execute.return_value = None
        self.mock_connection.commit.return_value = None

        # Execute
        result = self.repository.create_employee(emp)

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

    # Test Case 2: Test if task is created successfully
    def test_create_task_successfully(self):
        # Setup
        task = Task(
            task_name="Test Task",
            project_id=1,
            employee_id=1,
            status="Assigned",
            allocation_date="2025-01-01",
            deadline_date="2025-02-01"
        )
        # Mock database response
        self.mock_cursor.execute.return_value = None
        self.mock_connection.commit.return_value = None

        # Execute
        result = self.repository.create_task(task)

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

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

        # Mock database response
        mock_tasks = [
            {'task_id': 1, 'task_name': 'Task 1', 'status': 'Started',
             'allocation_date': '2025-01-01', 'deadline_date': '2025-02-01'},
            {'task_id': 2, 'task_name': 'Task 2', 'status': 'Assigned',
             'allocation_date': '2025-01-15', 'deadline_date': '2025-02-15'}
        ]
        self.mock_cursor.fetchall.return_value = mock_tasks

        # Execute
        result = self.repository.get_all_tasks(employee_id, project_id)

        # Assert
        self.assertEqual(len(result), 2)
        self.assertEqual(result[0]['task_name'], 'Task 1')
        self.assertEqual(result[1]['task_name'], 'Task 2')
        self.mock_cursor.execute.assert_called()

    # Test Case 4: Test if exceptions are thrown correctly
    def test_assign_project_to_nonexistent_employee_throws_exception(self):
        # Setup
        project_id = 1
        employee_id = 999  # Non-existent employee

        # Mock database response for employee check
        self.mock_cursor.fetchone.return_value = None

        # Execute and Assert
        with self.assertRaises(EmployeeNotFoundException):
            self.repository.assign_project_to_employee(project_id, employee_id)

    def test_assign_task_to_nonexistent_project_throws_exception(self):
        # Setup
        task_id = 1
        project_id = 999  # Non-existent project
        employee_id = 1

        # Mock database response for project check
        self.mock_cursor.fetchone.return_value = None

        # Execute and Assert
        with self.assertRaises(ProjectNotFoundException):
            self.repository.assign_task_in_project_to_employee(task_id, project_id, employee_id)

    # Additional test cases for better coverage
    def test_delete_nonexistent_employee_throws_exception(self):
        # Setup
        employee_id = 999  # Non-existent employee

        # Mock database response for employee check
        self.mock_cursor.fetchone.return_value = None

        # Execute and Assert
        with self.assertRaises(EmployeeNotFoundException):
            self.repository.delete_employee(employee_id)

    def test_delete_project_successfully(self):
        # Setup
        project_id = 1

        # Mock database responses
        self.mock_cursor.fetchone.return_value = [project_id]  # Project exists
        self.mock_cursor.execute.return_value = None
        self.mock_connection.commit.return_value = None

        result = self.repository.delete_project(project_id)

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


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

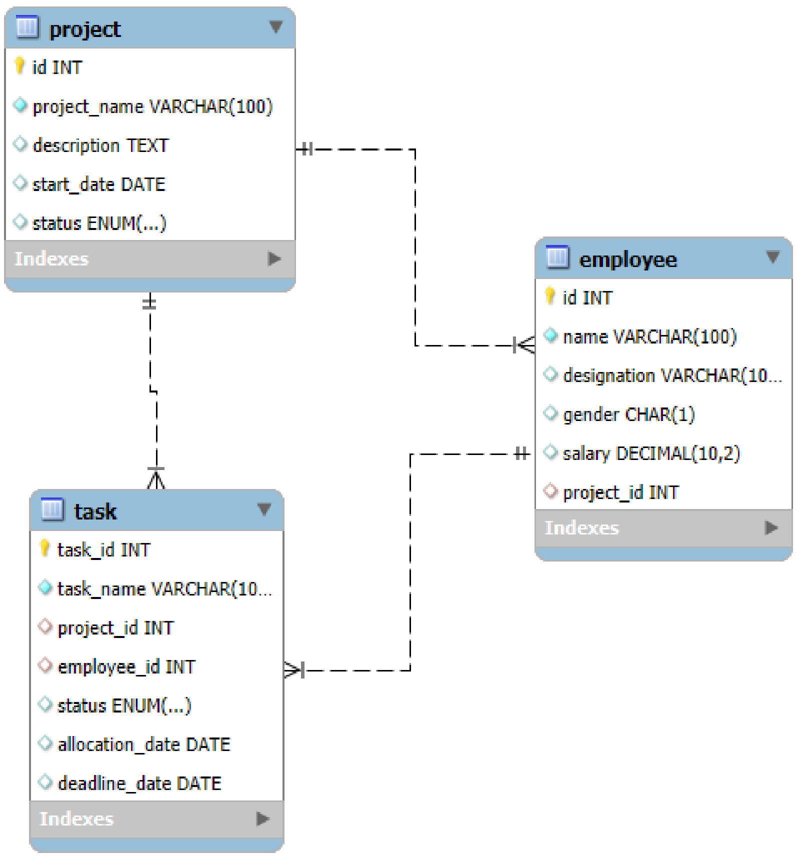
## 5. TECHNOLOGIES USED

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.

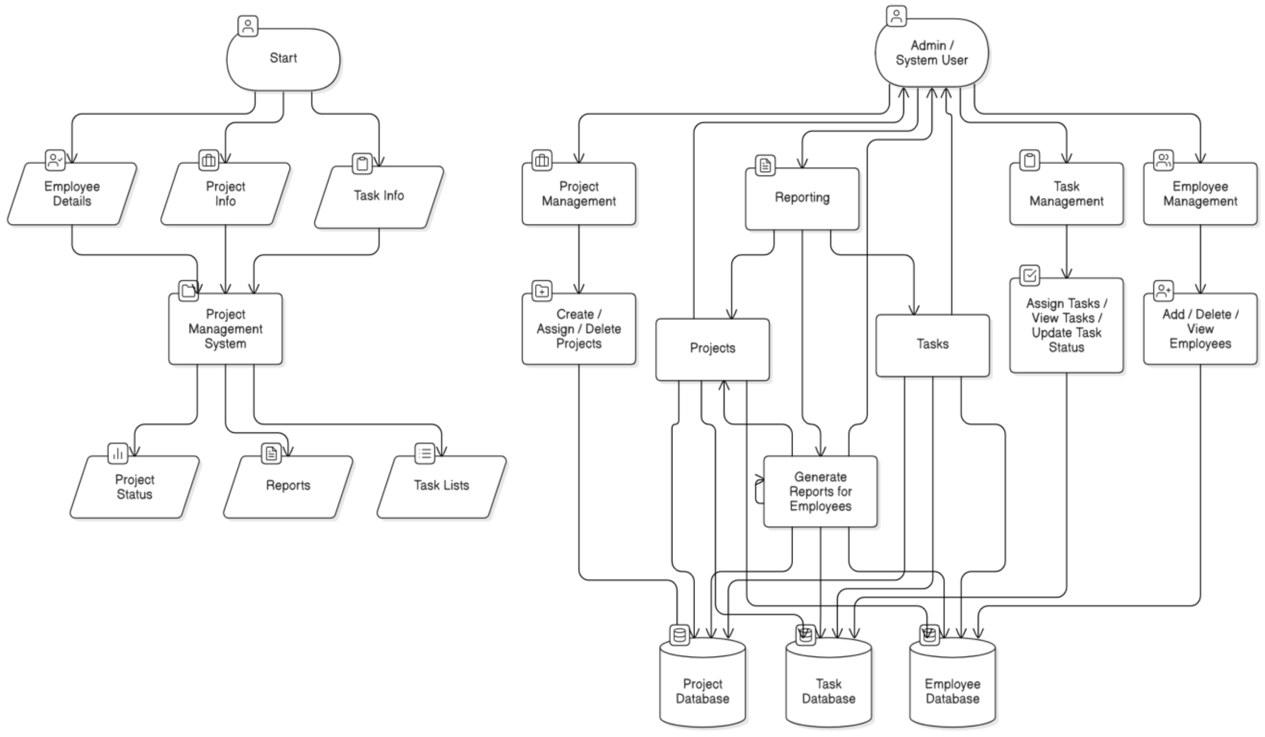
## 6. ER DIAGRAM

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.



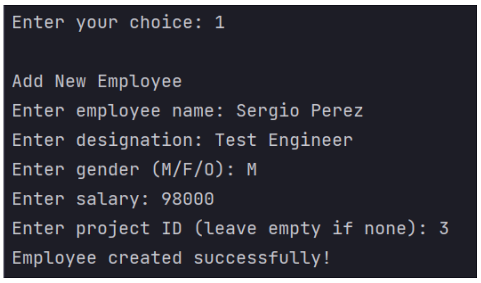
## 7. DATA FLOW DIAGRAM

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

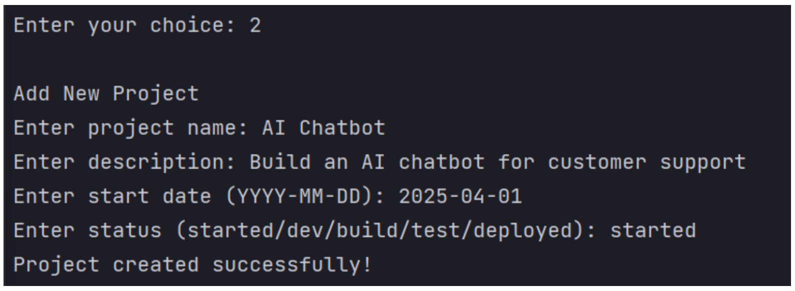
### 1. Add Employee



```
Enter your choice: 1

Add New Employee
Enter employee name: Sergio Perez
Enter designation: Test Engineer
Enter gender (M/F/O): M
Enter salary: 98000
Enter project ID (leave empty if none): 3
Employee created successfully!
```

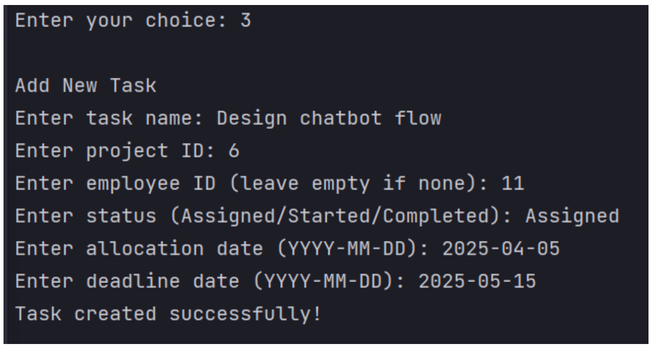
### 2. Add Project



```
Enter your choice: 2

Add New Project
Enter project name: AI Chatbot
Enter description: Build an AI chatbot for customer support
Enter start date (YYYY-MM-DD): 2025-04-01
Enter status (started/dev/build/test/deployed): started
Project created successfully!
```

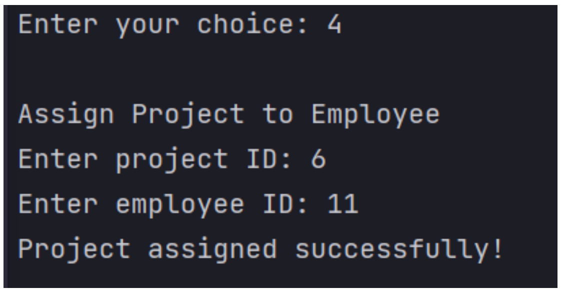
**3. Add Task**



```
Enter your choice: 3

Add New Task
Enter task name: Design chatbot flow
Enter project ID: 6
Enter employee ID (leave empty if none): 11
Enter status (Assigned/Started/Completed): Assigned
Enter allocation date (YYYY-MM-DD): 2025-04-05
Enter deadline date (YYYY-MM-DD): 2025-05-15
Task created successfully!
```

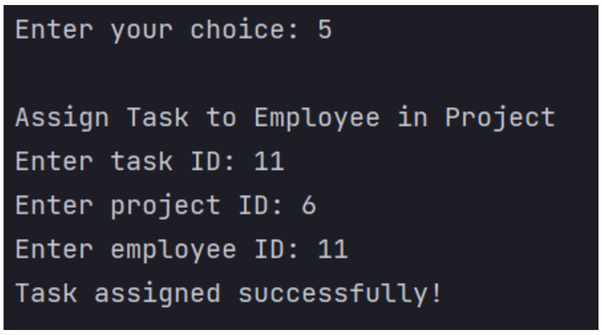
**4. Assign project to employee**



```
Enter your choice: 4

Assign Project to Employee
Enter project ID: 6
Enter employee ID: 11
Project assigned successfully!
```

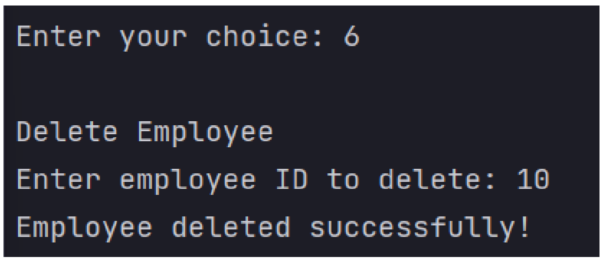
**5. Assign task within a project to employee**



```
Enter your choice: 5

Assign Task to Employee in Project
Enter task ID: 11
Enter project ID: 6
Enter employee ID: 11
Task assigned successfully!
```

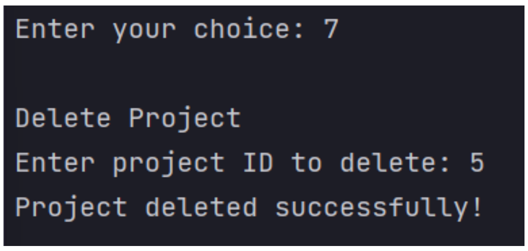
**6. Delete Employee**



```
Enter your choice: 6

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

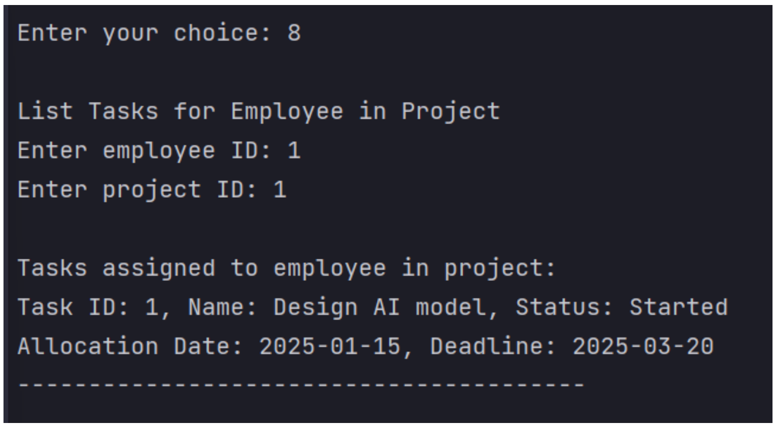
**7. Delete Project**



```
Enter your choice: 7

Delete Project
Enter project ID to delete: 5
Project deleted successfully!
```

**8. List all tasks assigned to an employee in a project**

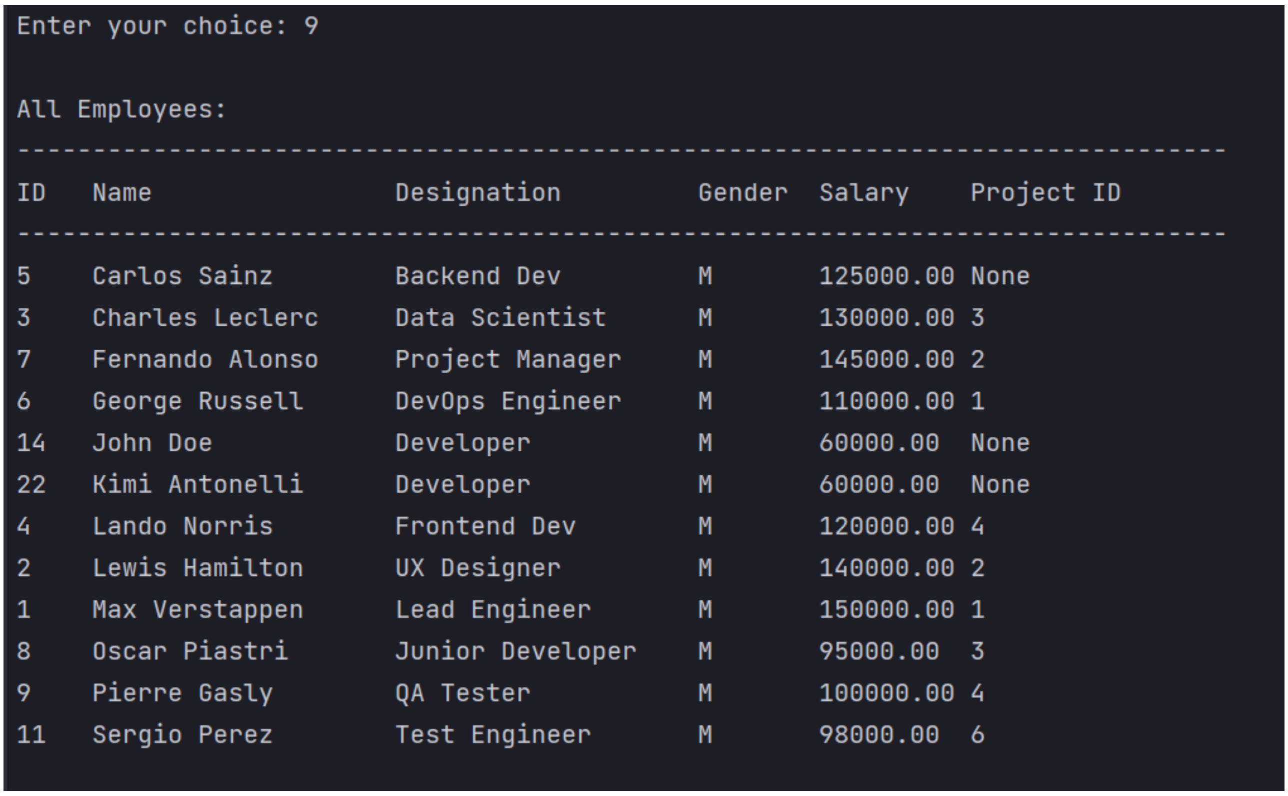


```
Enter your choice: 8

List Tasks for Employee in Project
Enter employee ID: 1
Enter project ID: 1

Tasks assigned to employee in project:
Task ID: 1, Name: Design AI model, Status: Started
Allocation Date: 2025-01-15, Deadline: 2025-03-20
----------------------------------------
```

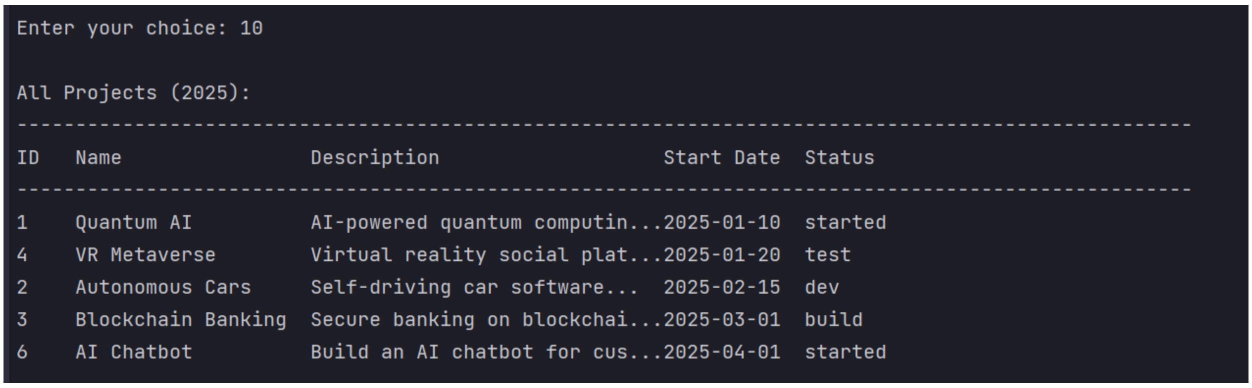
**9. Show all employees**



```
Enter your choice: 9

All Employees:
--------------------------------------------------------------------------------
ID   Name                Designation         Gender  Salary    Project ID
--------------------------------------------------------------------------------
5    Carlos Sainz        Backend Dev         M       125000.00 None
3    Charles Leclerc     Data Scientist      M       130000.00 3
7    Fernando Alonso     Project Manager     M       145000.00 2
6    George Russell      DevOps Engineer     M       110000.00 1
14   John Doe            Developer           M       60000.00  None
22   Kimi Antonelli      Developer           M       60000.00  None
4    Lando Norris        Frontend Dev        M       120000.00 4
2    Lewis Hamilton      UX Designer         M       140000.00 2
1    Max Verstappen      Lead Engineer       M       150000.00 1
8    Oscar Piastri       Junior Developer    M       95000.00  3
9    Pierre Gasly        QA Tester           M       100000.00 4
11   Sergio Perez        Test Engineer       M       98000.00  6
```

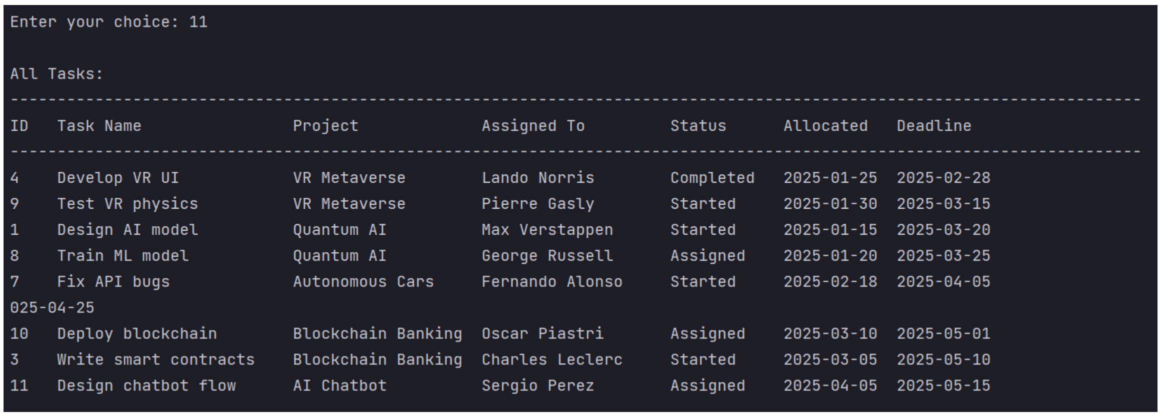
**10. Show all projects**



```
Enter your choice: 10

All Projects (2025):
----------------------------------------------------------------------------------------------------
ID   Name                Description                    Start Date  Status
----------------------------------------------------------------------------------------------------
1    Quantum AI          AI-powered quantum computin...2025-01-10  started
4    VR Metaverse        Virtual reality social plat...2025-01-20  test
2    Autonomous Cars     Self-driving car software...   2025-02-15  dev
3    Blockchain Banking  Secure banking on blockchai...2025-03-01  build
6    AI Chatbot          Build an AI chatbot for cus...2025-04-01  started
```

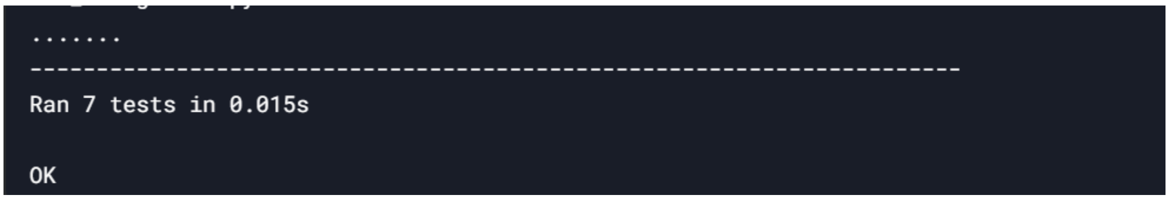
**11. Show all tasks**



```
Enter your choice: 11

All Tasks:
------------------------------------------------------------------------------------------------------------------------
ID   Task Name                Project             Assigned To         Status      Allocated   Deadline
------------------------------------------------------------------------------------------------------------------------
4    Develop VR UI            VR Metaverse        Lando 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          Max Verstappen      Started     2025-01-15  2025-03-20
8    Train ML model           Quantum AI          George Russell      Assigned    2025-01-20  2025-03-25
7    Fix API bugs             Autonomous Cars     Fernando Alonso     Started     2025-02-18  2025-04-05
025-04-25
10   Deploy blockchain        Blockchain Banking  Oscar Piastri       Assigned    2025-03-10  2025-05-01
3    Write smart contracts    Blockchain Banking  Charles Leclerc     Started     2025-03-05  2025-05-10
11   Design chatbot flow      AI Chatbot          Sergio Perez        Assigned    2025-04-05  2025-05-15
```

**12. Tests:**



```
.......
----------------------------------------------------------------------
Ran 7 tests in 0.015s

OK
```

## <u>9. FUTURE ENHANCEMENTS</u>

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

## <u>10. CONCLUSION</u>

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.