Case Study on PAYXPERT

HANIF MOHAMMED A

Superset ID: 5020402

ABSTRACT

This case study presents the development of PayXpert, a Python-based employee payroll and taxation management system designed to streamline HR and financial operations. The project is built using object-oriented principles, modular design, and a layered architecture, integrating MySQL for persistent data storage. The system is divided into key functional directories: entity for modeling data objects, dao for service logic and database interactions, exception for handling domain-specific errors, util for database connectivity and validations, test for unit testing, and main for user interface implementation.

PayXpert offers comprehensive features including employee management (CRUD operations), automated payroll processing, tax calculation based on income, and financial record tracking. Services like EmployeeService, PayrollService, TaxService, and FinancialRecordService handle database operations via defined interfaces, ensuring abstraction and flexibility. The application also includes custom exceptions to enhance robustness and a utility module for reusable components.

Unit tests using pytest validate key computations and input handling, ensuring system correctness. The main module provides a menu-driven CLI interface for real-time interaction with users. Overall, PayXpert demonstrates a well-structured and scalable backend system suitable for small to medium organizations seeking efficient employee and payroll management solutions.

1.Introduction

A payroll management system is a software-based solution designed to automate and streamline payroll operations within an organization. It handles various tasks including managing employee data, calculating salaries, computing taxes, and organizing financial records efficiently.

This system is developed using robust Object-Oriented Programming (OOP) principles and structured database design with SQL, adopting modular code practices to ensure scalability and ease of maintenance. The project incorporates core programming concepts such as control structures, loops, data collections, exception management, and unit testing, making it an ideal case study for aspiring software developers.

PayXpert delivers essential functionalities such as comprehensive employee information management, automatic salary and deduction processing, tax computation based on income brackets, and systematic financial documentation and reporting. The application is implemented using Python for its backend logic and MySQL for database integration.

2.Purpose of the Project

The main objective of the PayXpert Payroll Management System is to build a dependable, efficient, and scalable solution for managing payroll processes with ease. This project is designed to simplify the complexities associated with employee compensation, ensuring accuracy and automation. It seeks to resolve the difficulties encountered by HR and finance teams when dealing with manual or legacy payroll systems.

This project fulfills both academic learning and practical implementation goals:

Project Goals

- Automate the entire payroll lifecycle—from employee onboarding to salary disbursement.
- Ensure precise and transparent handling of salary, taxes, and deductions.
- Offer real-time access to employee financial records and payroll history.
- Enable easy generation of tax and financial reports for compliance and analysis.

Learning Objectives

- Apply Python and MySQL to develop a fully functional real-world application.
- Understand and implement Object-Oriented Programming (OOP) in a modular structure.
- Design and interact with relational databases using SQL.
- Utilize Python features such as collections, exception handling, validation, and unit testing.
- Create a user-friendly, menu-driven interface for system interaction.

By the end of this project, learners will demonstrate how Python and SQL can collaboratively solve real-world business challenges, while reinforcing best practices in software design and development.

3.Scope of the Project

The PayXpert Payroll Management System encompasses a variety of features and technical modules aimed at streamlining payroll operations through the use of Python and MySQL. Beyond automating core payroll tasks, the project showcases how to develop a clean, modular, and maintainable application following real-world software development practices.

Here’s what falls within the scope of this project:

Entity Modeling

All core data entities in the system—such as employees, payroll records, tax details, and financial transactions—are modeled using Python classes within the `entity` package.
Each class includes:

- Private attributes for storing field values (e.g., name, joining date, salary).
- Constructors (default and parameterized) to easily instantiate objects.
- Getter and setter methods to encapsulate and manage access to attributes.
- Utility methods, such as `calculateAge()` in the `Employee` class, to perform relevant computations.

This approach ensures clean, organized code and makes the system scalable and easy to enhance in the future.

Data Access Layer (DAO)

The Data Access Layer is responsible for all interactions with the MySQL database. It includes well-defined interfaces and their implementations for performing CRUD operations on tables like Employee, Payroll, Tax, and FinancialRecord.
Each module has its own interface (e.g., IEmployeeService) and a corresponding service class (e.g., EmployeeService) that handles database logic. This separation ensures a clean, modular structure that enhances maintainability and scalability.

Custom Manual Exceptions

To ensure the application handles errors gracefully and improves user experience, custom exceptions are defined in the exception package. These include:

- EmployeeNotFoundException for missing employee data
- PayrollGenerationException for payroll processing errors
- TaxCalculationException for issues during tax computation
- FinancialRecordException for financial transaction errors
- InvalidInputException for incorrect user input
- DatabaseConnectionException for database connectivity problems
  These exceptions make the system more robust, secure, and easier to debug during development or production use.

Database Integration

The PayXpert application integrates seamlessly with a MySQL database to manage all employee, payroll, tax, and financial record data. This is facilitated by two utility classes:

- **DBPropertyUtil**: Reads database connection settings from a property file.
- **DBConnUtil**: Establishes the actual database connection using the retrieved settings.
  All tables—such as Employee, Payroll, Tax, and FinancialRecord—are well-structured with primary and foreign key constraints to ensure data integrity and consistency.

Functionalities

The system provides all essential features expected in a complete payroll management solution:

- Full CRUD operations on employee records
- Payroll generation including salary, overtime, and deduction handling
- Income-based tax calculation

- Logging of financial transactions such as bonuses and tax payments
- Report generation for payroll, taxation, and financial summaries
  Each functionality is implemented through clean, modular, and reusable functions that follow industry-standard coding practices.

Menu-Driven Application

The PayXpert system offers an interactive, user-friendly, menu-driven interface located in the main package. Users can navigate through options to perform actions such as:

- Adding a new employee
- Generating payroll
- Viewing financial and tax records
  All user inputs are validated, and exceptions are handled gracefully to prevent the application from crashing due to incorrect data.

Unit Testing

To ensure system reliability, unit tests are implemented using Python's pytest framework. These tests validate:

- Accuracy of salary and tax calculations
- Proper handling of invalid or unexpected inputs
- Functionality of different service modules
  Testing guarantees the correctness of the system and prepares it for real-world deployment.
-

4.Project Structure

The PayXpert Payroll Management System is structured to ensure modularity, maintainability, and ease of navigation throughout the application. It is divided into clearly defined packages and layers, each responsible for specific tasks such as entity modeling, data access, exception management, utility services, and the overall application flow.

This section highlights the SQL database structure, which serves as the foundation for securely storing and efficiently managing all payroll, employee, tax, and financial record data.

4.1 SQL Structure (Database Schema)

The system uses a **MySQL relational database** to store employee data, payroll information, taxes, and financial records. The schema is designed using normalization principles to avoid data redundancy and ensure
consistency through primary and foreign keys.

Database Creation:

```
create database payxpert;
use payxpert;
```

Table Creation:

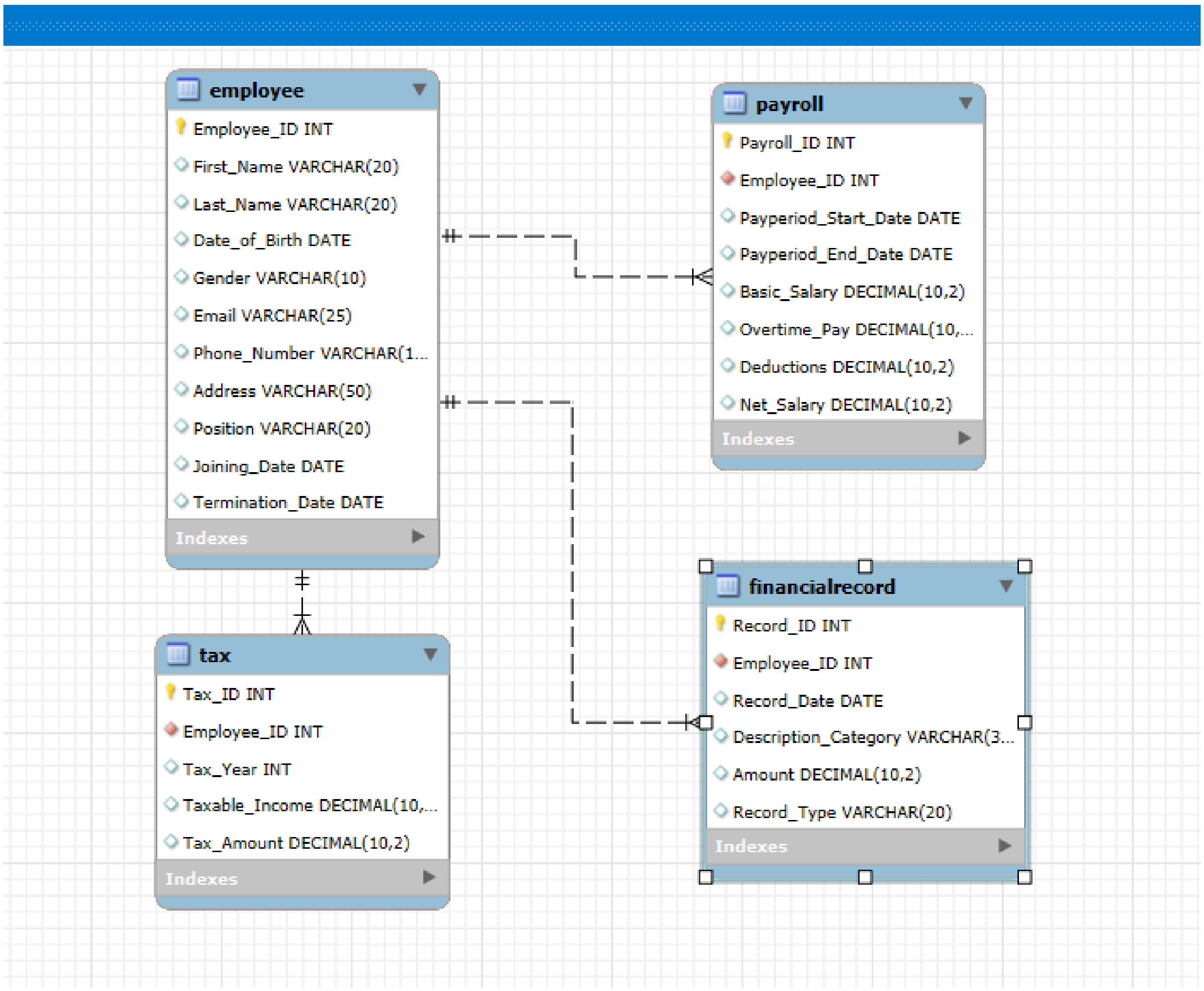
```
create table employee (
Employee_ID int not null auto_increment,
First_Name varchar(20),
Last_Name varchar(20),
Date_of_Birth date,
Gender varchar(10),
Email varchar(25),
Phone_Number varchar(15),
Address varchar(50),
Position varchar(20),
Joining_Date date,
Termination_Date date,
primary key(Employee_ID)
);
```

```
create table payroll(
Payroll_ID int not null auto_increment,
Employee_ID int not null,
Payperiod_Start_Date date,
Payperiod_End_Date date ,
Basic_Salary decimal(10,2),
Overtime_Pay decimal(10,2),
Deductions decimal(10,2),
Net_Salary decimal(10,2),
primary key(Payroll_ID),
foreign key(Employee_ID) references employee(Employee_ID) on delete
cascade
);

create table tax(
Tax_ID int not null auto_increment,
Employee_ID int not null,
Tax_Year int,
Taxable_Income decimal(10,2),
Tax_Amount decimal(10,2),
primary key(Tax_ID),
foreign key(Employee_ID) references employee(Employee_ID) on delete
cascade
);

create table financialrecord(
Record_ID int not null auto_increment,
Employee_ID int not null,
Record_Date date,
Description_Category varchar(30),
Amount decimal(10,2),
Record_Type varchar(20),
primary key(Record_ID),
foreign key(Employee_ID) references employee(Employee_ID) on delete
cascade
);
```

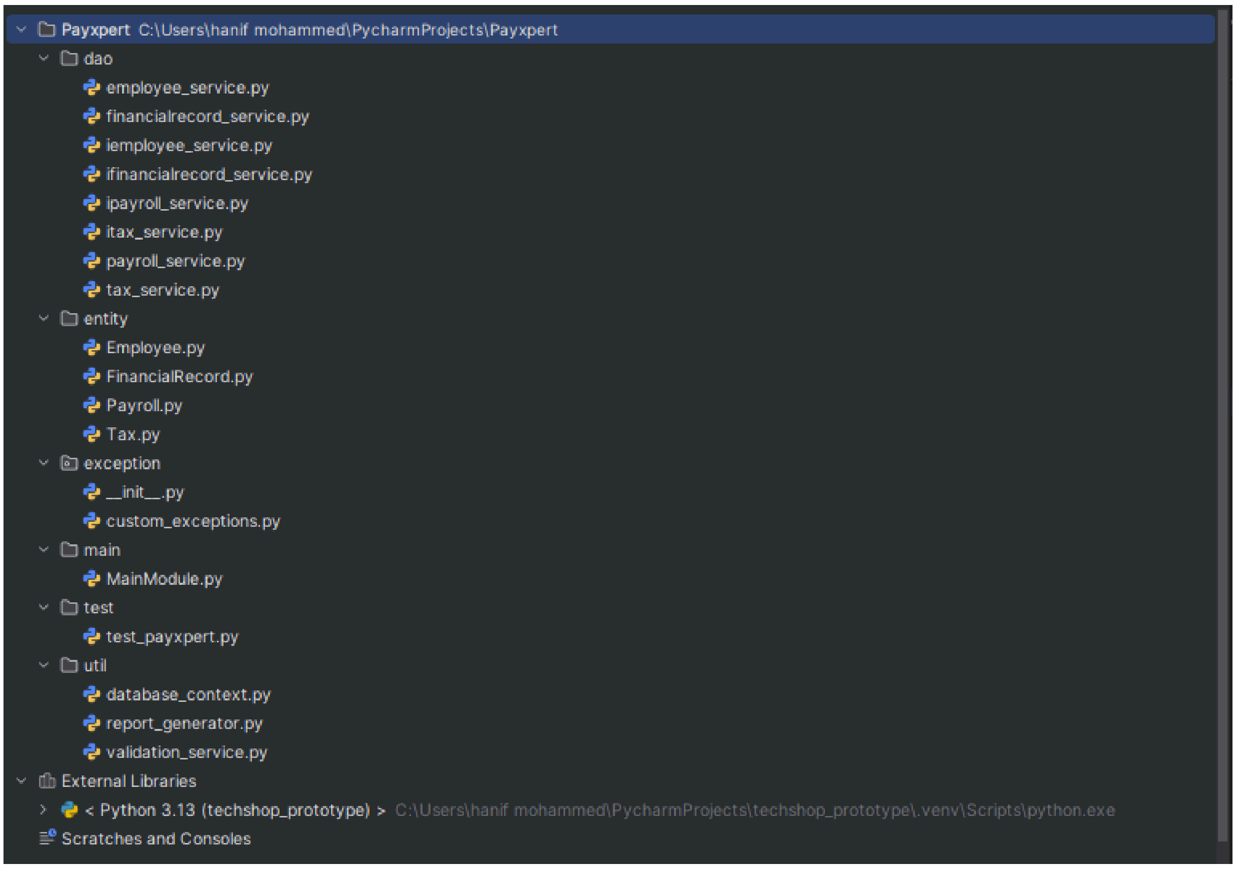
Inserting values

```
insert into employee
(First_Name,Last_Name,Date_of_Birth,Gender,Email,Phone_Number,
Address,Position, Joining_Date, Termination_Date) values
('John','Doe','1985-02-15','Male','john.doe@email.com','9876543210','New
York','Manager','2020-01-10',NULL),
('Jane','Smith','1990-06-20','Female','jane.smith@email.com','9123456789',
'Chicago','Developer','2021-03-22',NULL),
('Mike','Brown','1988-12-12','Male','mike.brown@email.com','9988776655','
Houston','Tester','2022-07-01',NULL),
('Sara','Lee','1992-08-08','Female','sara.lee@email.com','9871234560','Dall
as','Analyst','2023-01-15',NULL),
('Tom','Wilson','1984-10-10','Male','tom.wilson@email.com','9090909090','S
eattle','Designer','2019-05-25',NULL),
('Emily','Clark','1995-11-30','Female','emily.clark@email.com','9988771122'
,'San Jose','HR','2022-09-05',NULL),
('Robert','King','1980-04-04','Male','robert.king@email.com','9876000000','A
tlanta','Admin','2018-12-10','2023-12-31'),
('Lucy','Turner','1993-03-03','Female','lucy.turner@email.com','9012345678'
,'Boston','Support','2020-06-18',NULL),
('Steve','Jobs','1980-02-24','Male','steve.jobs@email.com','9091234567','Cal
ifornia','CEO','2010-01-01',NULL),
('Nina','Patel','1997-07-07','Female','nina.patel@email.com','9911223344','
Phoenix','Intern','2024-02-01',NULL);


insert into payroll
(Employee_ID,Payperiod_Start_Date,Payperiod_End_Date,
Basic_Salary, Overtime_Pay,Deductions, Net_Salary) values
(1,'2024-03-01','2024-03-31',70000.00,5000.00,2000.00,73000.00),
(2,'2024-03-01','2024-03-31',60000.00,3000.00,1500.00,61500.00),
(3,'2024-03-01','2024-03-31',50000.00,2000.00,1000.00,51000.00),
(4,'2024-03-01','2024-03-31',55000.00,2500.00,1800.00,55700.00),
(5,'2024-03-01','2024-03-31',65000.00,4000.00,2200.00,66800.00),
(6,'2024-03-01','2024-03-31',48000.00,1200.00,1000.00,48200.00),
(7,'2024-03-01','2024-03-31',40000.00,0.00,500.00, 39500.00),
(8,'2024-03-01','2024-03-31',43000.00,1000.00,700.00,43300.00),
(9,'2024-03-01','2024-03-31',90000.00,10000.00,3000.00,97000.00),
(10,'2024-03-01','2024-03-31',20000.00,500.00,200.00,20300.00);

insert into tax (Employee_ID,Tax_Year,Taxable_Income,Tax_Amount)
values
(1,2023,840000.00,84000.00),
(2,2023,720000.00,72000.00),
(3,2023,600000.00,60000.00),
(4,2023,660000.00,66000.00),
(5,2023,780000.00,78000.00),
(6,2023,576000.00,57600.00),
(7,2023,480000.00,48000.00),
(8,2023,516000.00,51600.00),
(9,2023,1080000.00,108000.00),
(10,2023,240000.00,24000.00);

insert into financialrecord
(Employee_ID,Record_Date,Description_Category,Amount,Record_Type)
values
(1,'2024-03-05','Travel',1500.00,'Expense'),
(2,'2024-03-10','Bonus',3000.00,'Income'),
(3,'2024-03-12','Lunch',500.00,'Expense'),
(4,'2024-03-15','Project Allowance',2000.00,'Income'),
(5,'2024-03-18','Conference',800.00,'Expense'),
(6,'2024-03-20','Referral Bonus',1000.00,'Income'),
(7,'2024-03-22','Training',1200.00,'Expense'),
(8,'2024-03-24','Performance Bonus',2500.00,'Income'),
(9,'2024-03-26','Medical',900.00,'Expense'),
(10,'2024-03-28','Stipend',1500.00,'Income');
```

ERR Diagram



File Directories



```
Payxpert C:\Users\hanif mohammed\PycharmProjects\Payxpert
  dao
    employee_service.py
    financialrecord_service.py
    iemployee_service.py
    ifinancialrecord_service.py
    ipayroll_service.py
    itax_service.py
    payroll_service.py
    tax_service.py
  entity
    Employee.py
    FinancialRecord.py
    Payroll.py
    Tax.py
  exception
    __init__.py
    custom_exceptions.py
  main
    MainModule.py
  test
    test_payxpert.py
  util
    database_context.py
    report_generator.py
    validation_service.py
External Libraries
  < Python 3.13 (techshop_prototype) > C:\Users\hanif mohammed\PycharmProjects\techshop_prototype\.venv\Scripts\python.exe
Scratches and Consoles
```

4.2 OOP Structure (Object-Oriented Programming)

The **PayXpert Payroll Management System** is designed using core **Object- Oriented Programming (OOP)** principles in Python. This structure allows the system to be modular, reusable, maintainable, and easy to extend. The entire application is organized into packages based on responsibility, and each class represents a real-world entity or logical functionality.

Below is an overview of how OOP is applied in this project:

1.Entity Directory:

The Entity directory contains the data model classes representing core business entities like Employee, Payroll, Tax, and FinancialRecord.
Each class encapsulates attributes as private variables with appropriate getters, setters, and constructors.
This directory ensures object-oriented representation of database tables, supporting clean data handling across the application.

Payxpert/entity/Employee.py

```
from datetime import date

class Employee:
    def __init__(self, employee_id=None, first_name=None,
last_name=None, date_of_birth=None,
                 gender=None, email=None, phone_number=None,
address=None,
                 position=None, joining_date=None, termination_date=None):
        self.__employee_id = employee_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__date_of_birth = date_of_birth
        self.__gender = gender
        self.__email = email
        self.__phone_number = phone_number
        self.__address = address
        self.__position = position
        self.__joining_date = joining_date
        self.__termination_date = termination_date

    # Getters and setters
    def get_employee_id(self):
        return self.__employee_id
    def set_employee_id(self, value):
        self.__employee_id = value

    def get_first_name(self):
        return self.__first_name
    def set_first_name(self, value):
        self.__first_name = value

    def get_last_name(self):
        return self.__last_name
    def set_last_name(self, value):
        self.__last_name = value

    def get_date_of_birth(self):
        return self.__date_of_birth
    def set_date_of_birth(self, value):
        self.__date_of_birth = value

    def get_gender(self):
        return self.__gender
    def set_gender(self, value):
        self.__gender = value

    def get_email(self):
        return self.__email
    def set_email(self, value):
        self.__email = value

    def get_phone_number(self):
        return self.__phone_number
    def set_phone_number(self, value):
        self.__phone_number = value

    def get_address(self):
        return self.__address
    def set_address(self, value):
        self.__address = value

    def get_position(self):
        return self.__position
    def set_position(self, value):
        self.__position = value

    def get_joining_date(self):
        return self.__joining_date
    def set_joining_date(self, value):
        self.__joining_date = value

    def get_termination_date(self):
        return self.__termination_date
    def set_termination_date(self, value):
        self.__termination_date = value

    # Method
    def calculate_age(self):
        if self.__date_of_birth:
            today = date.today()
            return today.year - self.__date_of_birth.year - (
                (today.month, today.day) < (self.__date_of_birth.month,
self.__date_of_birth.day))
        return None
```

Payxpert/entity/FinancialRecord.py

```
class FinancialRecord:
    def __init__(self, record_id=None, employee_id=None,
record_date=None,
                 description=None, amount=0.0, record_type=None):
        self.__record_id = record_id
        self.__employee_id = employee_id
        self.__record_date = record_date
        self.__description = description
        self.__amount = amount
        self.__record_type = record_type

    def get_record_id(self):
        return self.__record_id
    def set_record_id(self, value):
        self.__record_id = value
    def get_employee_id(self):
        return self.__employee_id
    def set_employee_id(self, value):
        self.__employee_id = value

    def get_record_date(self):
        return self.__record_date
    def set_record_date(self, value):
        self.__record_date = value

    def get_description(self):
        return self.__description
    def set_description(self, value):
        self.__description = value

    def get_amount(self):
        return self.__amount
    def set_amount(self, value):
        self.__amount = value

    def get_record_type(self):
        return self.__record_type
    def set_record_type(self, value):
        self.__record_type = value
```

Payxpert/entity/Payroll.py

```
class Payroll:
    def __init__(self, payroll_id=None, employee_id=None,
pay_period_start_date=None,
                 pay_period_end_date=None, basic_salary=0.0,
overtime_pay=0.0,
                 deductions=0.0, net_salary=0.0):
        self.__payroll_id = payroll_id
        self.__employee_id = employee_id
        self.__pay_period_start_date = pay_period_start_date
        self.__pay_period_end_date = pay_period_end_date
        self.__basic_salary = basic_salary
        self.__overtime_pay = overtime_pay
        self.__deductions = deductions
        self.__net_salary = net_salary
    def get_payroll_id(self):
        return self.__payroll_id
    def set_payroll_id(self, value):
        self.__payroll_id = value

    def get_employee_id(self):
        return self.__employee_id
    def set_employee_id(self, value):
        self.__employee_id = value

    def get_pay_period_start_date(self):
        return self.__pay_period_start_date
    def set_pay_period_start_date(self, value):
        self.__pay_period_start_date = value

    def get_pay_period_end_date(self):
        return self.__pay_period_end_date
    def set_pay_period_end_date(self, value):
        self.__pay_period_end_date = value

    def get_basic_salary(self):
        return self.__basic_salary
    def set_basic_salary(self, value):
        self.__basic_salary = value

    def get_overtime_pay(self):
        return self.__overtime_pay
    def set_overtime_pay(self, value):
        self.__overtime_pay = value

    def get_deductions(self):
        return self.__deductions
    def set_deductions(self, value):
        self.__deductions = value

    def get_net_salary(self):
        return self.__net_salary
    def set_net_salary(self, value):
        self.__net_salary = value
```

Payxpert/entity/Tax.py

```
class Tax:
    def __init__(self, tax_id=None, employee_id=None, tax_year=None,
                 taxable_income=0.0, tax_amount=0.0):
        self.__tax_id = tax_id
        self.__employee_id = employee_id
        self.__tax_year = tax_year
        self.__taxable_income = taxable_income
        self.__tax_amount = tax_amount

    def get_tax_id(self):
        return self.__tax_id
    def set_tax_id(self, value):
        self.__tax_id = value

    def get_employee_id(self):
        return self.__employee_id
    def set_employee_id(self, value):
        self.__employee_id = value

    def get_tax_year(self):
        return self.__tax_year
    def set_tax_year(self, value):
        self.__tax_year = value

    def get_taxable_income(self):
        return self.__taxable_income
    def set_taxable_income(self, value):
        self.__taxable_income = value

    def get_tax_amount(self):
        return self.__tax_amount
    def set_tax_amount(self, value):
        self.__tax_amount = value
```

2.DAO (Data Access Object) Directory:

The **dao (Data Access Object) directory** contains all service interfaces and their concrete implementations for managing business logic and database interactions.

It follows an abstraction pattern by defining base interfaces like IEmployeeService and implementing them in classes like EmployeeService.

This structure promotes clean separation between service logic and UI, making the application modular, testable, and maintainable.

Payxpert/dao/iemployees_service.py

```
from abc import ABC, abstractmethod

class IEmployeeService(ABC):
    @abstractmethod
    def get_employee_by_id(self, employee_id):
        pass

    @abstractmethod
    def get_all_employees(self):
        pass

    @abstractmethod
    def add_employee(self, employee):
        pass

    @abstractmethod
    def update_employee(self, employee):
        pass

    @abstractmethod
    def remove_employee(self, employee_id):
        pass
```

Payxpert/dao/employees_service.py

```
from dao.iemployee_service import IEmployeeService
from entity.Employee import Employee


class EmployeeService(IEmployeeService):
    def __init__(self, connection):
        self.conn = connection


    def get_employee_by_id(self, employee_id):
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM employee WHERE Employee_ID = 
%s", (employee_id,))
        row = cursor.fetchone()
        if row:
            return Employee(
                employee_id=row['Employee_ID'],
                first_name=row['First_Name'],
                last_name=row['Last_Name'],
                date_of_birth=row['Date_of_Birth'],
                gender=row['Gender'],
                email=row['Email'],
                phone_number=row['Phone_Number'],
                address=row['Address'],
                position=row['Position'],
                joining_date=row['Joining_Date'],
                termination_date=row['Termination_Date']
            )

        return None

    def get_all_employees(self):
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM employee")
        rows = cursor.fetchall()
        return [Employee(
            employee_id=row['Employee_ID'],
            first_name=row['First_Name'],
            last_name=row['Last_Name'],
            date_of_birth=row['Date_of_Birth'],
            gender=row['Gender'],
            email=row['Email'],
            phone_number=row['Phone_Number'],
            address=row['Address'],
            position=row['Position'],
            joining_date=row['Joining_Date'],
            termination_date=row['Termination_Date']
        ) for row in rows]

    def add_employee(self, employee):
        cursor = self.conn.cursor()
        query = """
            INSERT INTO employee (Employee_ID, First_Name,
Last_Name, Date_of_Birth, Gender,
                                  Email, Phone_Number, Address, Position,
Joining_Date, Termination_Date)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        values = (
            employee.get_employee_id(), employee.get_first_name(),
employee.get_last_name(),
            employee.get_date_of_birth(), employee.get_gender(),
employee.get_email(),
            employee.get_phone_number(), employee.get_address(),
employee.get_position(),
            employee.get_joining_date(),
employee.get_termination_date()
        )
        cursor.execute(query, values)
        self.conn.commit()

    def update_employee(self, employee):
        cursor = self.conn.cursor()
        query = """
            UPDATE employee
            SET First_Name=%s, Last_Name=%s, Email=%s,
Phone_Number=%s
            WHERE Employee_ID = %s
        """
        values = (
            employee.get_first_name(), employee.get_last_name(),
            employee.get_email(), employee.get_phone_number(),
            employee.get_employee_id()
        )
        cursor.execute(query, values)
        self.conn.commit()

    def remove_employee(self, employee_id):
        cursor = self.conn.cursor()
        cursor.execute("DELETE FROM employee WHERE Employee_ID =
%s", (employee_id,))
        self.conn.commit()
```

Payxpert/dao/ifinancialrecord_service.py

```
from abc import ABC, abstractmethod

class IFinancialRecordService(ABC):
    @abstractmethod
    def add_financial_record(self, record):
        pass

    @abstractmethod
    def get_financial_record_by_id(self, record_id):
        pass

    @abstractmethod
    def get_financial_records_for_employee(self, employee_id):
        pass

    @abstractmethod
    def get_financial_records_for_date(self, date):
        pass
```

Payxpert/dao/financialrecord_service.py

```
from dao.ifinancialrecord_service import IFinancialRecordService


class FinancialRecordService(IFinancialRecordService):
    def __init__(self, connection):
        self.conn = connection

    def add_financial_record(self, record):
        self.records[record.record_id] = record

    def get_financial_record_by_id(self, record_id):
        return self.records.get(record_id)

    def get_financial_records_for_employee(self, employee_id):
        return [r for r in self.records.values() if r.employee_id ==
employee_id]

    def get_financial_records_for_date(self, date):
        return [r for r in self.records.values() if r.record_date == date]
```

Payxpert/dao/ipayroll_service.py

```
from abc import ABC, abstractmethod


class IPayrollService(ABC):
    @abstractmethod
    def generate_payroll(self, employee):
        pass


    @abstractmethod
    def get_payroll_by_id(self, payroll_id):
        pass


    @abstractmethod
    def get_payrolls_for_employee(self, employee_id):
        pass


    @abstractmethod
    def get_payrolls_for_period(self, start_date, end_date):
        pass
```

Payxpert/dao/payroll_service.py

```
from dao.ipayroll_service import IPayrollService
from entity.Payroll import Payroll


class PayrollService(IPayrollService):
    def __init__(self, connection):
        self.conn = connection

    def generate_payroll(self, payroll: Payroll):
        cursor = self.conn.cursor()
        query = """
            INSERT INTO payroll (Employee_ID, Payperiod_Start_Date, 
Payperiod_End_Date,
                                 Basic_Salary, Overtime_Pay, Deductions, 
Net_Salary)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """
        values = (
            payroll.get_employee_id(),
            payroll.get_pay_period_start_date(),
            payroll.get_pay_period_end_date(),
            payroll.get_basic_salary(),
            payroll.get_overtime_pay(),
            payroll.get_deductions(),
            payroll.get_net_salary()
        )
        cursor.execute(query, values)
        self.conn.commit()

    def get_payroll_by_id(self, payroll_id):
        return self.payrolls.get(payroll_id)

    def get_payrolls_for_employee(self, employee_id):
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM payroll WHERE Employee_ID =
%s", (employee_id,))
        rows = cursor.fetchall()

        from entity.Payroll import Payroll
        return [Payroll(
            payroll_id=row["Payroll_ID"],
            employee_id=row["Employee_ID"],
            pay_period_start_date=row["Payperiod_Start_Date"],
            pay_period_end_date=row["Payperiod_End_Date"],
            basic_salary=row["Basic_Salary"],
            overtime_pay=row["Overtime_Pay"],
            deductions=row["Deductions"],
            net_salary=row["Net_Salary"]
        ) for row in rows]

    def get_payrolls_for_period(self, start_date, end_date):
        return [p for p in self.payrolls.values() if start_date <=
p.pay_period_start_date <= end_date]
```

Payxpert/dao/itax_service.py

```
from abc import ABC, abstractmethod

class ITaxService(ABC):
    @abstractmethod
    def calculate_tax(self, employee):
        pass

    @abstractmethod
    def get_tax_by_id(self, tax_id):
        pass

    @abstractmethod
    def get_taxes_for_employee(self, employee_id):
        pass

    @abstractmethod
    def get_taxes_for_year(self, year):
        pass
```

Payxpert/dao/tax_service.py

```
from dao.itax_service import ITaxService
from entity.Tax import Tax

class TaxService(ITaxService):
    def __init__(self, connection):
        self.conn = connection

    def calculate_tax(self, employee_id, tax_year):
        from decimal import Decimal
        cursor = self.conn.cursor()

        # Calculate taxable income from payroll
        cursor.execute("""
            SELECT SUM(Basic_Salary + Overtime_Pay - Deductions) AS
TaxableIncome
            FROM payroll
            WHERE Employee_ID = %s
        """, (employee_id,))
        result = cursor.fetchone()

        taxable_income = result["TaxableIncome"] if
result["TaxableIncome"] else Decimal('0.00')
        tax_amount = taxable_income * Decimal('0.10')  # 10% tax

        # Insert into tax table
        cursor.execute("""
            INSERT INTO tax (Employee_ID, Tax_Year, Taxable_Income,
Tax_Amount)
            VALUES (%s, %s, %s, %s)
        """, (employee_id, tax_year, taxable_income, tax_amount))

        self.conn.commit()

    def get_taxes_for_employee(self, employee_id):
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM tax WHERE Employee_ID = %s",
(employee_id,))
        rows = cursor.fetchall()

        return [
            Tax(
                tax_id=row["Tax_ID"],
                employee_id=row["Employee_ID"],
                tax_year=row["Tax_Year"],
                taxable_income=row["Taxable_Income"],
                tax_amount=row["Tax_Amount"]
            )
            for row in rows
        ]

    def get_tax_by_id(self, tax_id):
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM tax WHERE Tax_ID = %s",
(tax_id,))
        row = cursor.fetchone()

        if row:
            return Tax(
                tax_id=row["Tax_ID"],
                employee_id=row["Employee_ID"],
                tax_year=row["Tax_Year"],
                taxable_income=row["Taxable_Income"],
                tax_amount=row["Tax_Amount"]
            )
        return None

    def get_taxes_for_year(self, tax_year):
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM tax WHERE Tax_Year = %s",
(tax_year,))
        rows = cursor.fetchall()

        return [
            Tax(
                tax_id=row["Tax_ID"],
                employee_id=row["Employee_ID"],
                tax_year=row["Tax_Year"],
                taxable_income=row["Taxable_Income"],
                tax_amount=row["Tax_Amount"]
            )
            for row in rows
        ]
```

3.Exception Directory:

The Exception directory defines custom exception classes tailored to the application's business logic.

It includes specific exceptions like EmployeeNotFoundException, TaxCalculationException, and DatabaseConnectionException.

These help in providing clear, meaningful error handling and improve the application's robustness and maintainability.

Payxpert/exception/__init__.py

```
#EMPTY FILE
```

Payxpert/exception/custom_exceptions.py

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



class PayrollGenerationException(Exception):
    def __init__(self, message="Error generating payroll."):
        super().__init__(message)



class TaxCalculationException(Exception):
    def __init__(self, message="Error calculating tax."):
        super().__init__(message)



class FinancialRecordException(Exception):
    def __init__(self, message="Error in financial record operation."):
        super().__init__(message)



class InvalidInputException(Exception):
    def __init__(self, message="Invalid input provided."):
        super().__init__(message)
```

```
class DatabaseConnectionException(Exception):
    def __init__(self, message="Failed to connect to the database."):
        super().__init__(message)
```

4.Util Directory:

The Util directory contains utility classes that support core functionalities like database connection and input validation.
It includes DatabaseContext for establishing MySQL connections and ValidationService for validating emails and phone numbers.
These helper classes promote code reuse and keep the main logic clean and modular.

Payxpert/util/database_context.py

```
import pymysql


class DatabaseContext:
    @staticmethod
    def get_connection():
        try:
            connection = pymysql.connect(
                host='localhost',
                user='root',
                password='root',
                database='payxpert1',
                cursorclass=pymysql.cursors.DictCursor
            )
            return connection
        except pymysql.MySQLError as e:
            print("Database connection failed:", e)
            return None
```

Payxpert/util/report_generator.py

```
class ReportGenerator:
    @staticmethod
    def generate_payroll_report(payrolls):
        for p in payrolls:
            print(f"{p.payroll_id} - {p.employee_id} - {p.net_salary}")

    @staticmethod
    def generate_tax_summary(taxes):
        for t in taxes:
            print(f"{t.tax_id} - {t.employee_id} - {t.tax_amount}")

    @staticmethod
    def generate_financial_report(records):
        for r in records:
            print(f"{r.record_id} - {r.description} - {r.amount}
({r.record_type})")
```

Payxpert/util/validation_service.py

```
import re

class ValidationService:

    @staticmethod

    def validate_email(email):

        return bool(re.match(r"[^@]+@[^@]+\.[^@]+", email))


    @staticmethod

    def validate_phone(phone):

        return phone.isdigit() and len(phone) == 10
```

5.Main Directory:

The **Main directory** holds the MainModule.py file, which serves as the entry point of the application.

It provides a menu-driven interface to interact with features like employee management, payroll processing, and tax calculation.

This module ties together all services and allows users to perform operations in a structured, user-friendly way.

Payxpert/main/MainModule.py

```
from dao.employee_service import EmployeeService

from dao.payroll_service import PayrollService

from dao.tax_service import TaxService

from dao.financialrecord_service import FinancialRecordService
from util.database_context import DatabaseContext
from exception.custom_exceptions import *
from entity.Employee import Employee
from entity.Payroll import Payroll
from entity.Tax import Tax

import datetime


class MainModule:
    def __init__(self):
        # Establishing DB connection
        self.connection = DatabaseContext.get_connection()
        if not self.connection:
            raise DatabaseConnectionException("Could not connect to
database.")

        # Injecting DB connection into services
        self.employee_service = EmployeeService(self.connection)
        self.payroll_service = PayrollService(self.connection)
        self.tax_service = TaxService(self.connection)
        self.financial_service = FinancialRecordService(self.connection)

    def menu(self):
        while True:
            print("\n========== EMPLOYEE MANAGEMENT SYSTEM
==========")
            print("1. Add Employee")
            print("2. View All Employees")
            print("3. Get Employee by ID")
            print("4. Update Employee")
            print("5. Delete Employee")
            print("6. Generate Payroll")
            print("7. View Payrolls for an Employee")
            print("8. Calculate Tax")
            print("9. View Taxes for an Employee")
            print("10. Exit")

            choice = input("Enter your choice: ")

            try:
                if choice == '1':
                    self.add_employee()
                elif choice == '2':
                    self.view_all_employees()
                elif choice == '3':
                    self.get_employee_by_id()
                elif choice == '4':
                    self.update_employee()
                elif choice == '5':
                    self.delete_employee()
                elif choice == '6':
                    self.generate_payroll()
                elif choice == '7':
                    self.view_payrolls_for_employee()
                elif choice == '8':
                    self.calculate_tax()
                elif choice == '9':
                    self.view_taxes_for_employee()
                elif choice == '10':
                    print("Exiting...")
                    break




                else:
                    print("Invalid choice. Try again.")

            except Exception as e:
                print(f"Error: {str(e)}")

    def add_employee(self):
        print("\nEnter Employee Details:")
        try:
            emp = Employee(
                employee_id=input("Employee ID: "),
                first_name=input("First Name: "),
                last_name=input("Last Name: "),
                date_of_birth=input("Date of Birth (YYYY-MM-DD): "),
                gender=input("Gender: "),
                email=input("Email: "),
                phone_number=input("Phone Number: "),
                address=input("Address: "),
                position=input("Position: "),
                joining_date=input("Joining Date (YYYY-MM-DD): "),
                termination_date=input("Termination Date (YYYY-MM-DD) or
leave blank: ") or None
            )
            self.employee_service.add_employee(emp)
            print("Employee added successfully.")
        except InvalidInputException as e:
            print(f"Invalid input: {str(e)}")

    def view_all_employees(self):
        employees = self.employee_service.get_all_employees()
        if not employees:
            print("No employees found.")
        for emp in employees:
            print(f"ID: {emp.get_employee_id()} | Name: 
{emp.get_first_name()} {emp.get_last_name()} | Email: 
{emp.get_email()}")


    def get_employee_by_id(self):
        emp_id = input("Enter Employee ID: ")
        emp = self.employee_service.get_employee_by_id(emp_id)
        if emp:
            print(f"""
            Employee Details:
            ID: {emp.get_employee_id()}
            First Name: {emp.get_first_name()}
            Last Name: {emp.get_last_name()}
            Date of Birth: {emp.get_date_of_birth()}
            Gender: {emp.get_gender()}
            Email: {emp.get_email()}
            Phone Number: {emp.get_phone_number()}
            Address: {emp.get_address()}
            Position: {emp.get_position()}
            Joining Date: {emp.get_joining_date()}
            Termination Date: {emp.get_termination_date()}
            """)
        else:
            print("Employee not found.")


    def update_employee(self):
        emp_id = input("Enter Employee ID to update: ")
        employee = self.employee_service.get_employee_by_id(emp_id)
        if not employee:
            print("Employee not found.")
            return
        print("Leave field blank to keep current value.")
        employee.set_first_name(input("First Name: ") or 
employee.get_first_name())
        employee.set_last_name(input("Last Name: ") or 
employee.get_last_name())
        employee.set_email(input("Email: ") or employee.get_email())
        employee.set_phone_number(input("Phone Number: ") or 
employee.get_phone_number())
        self.employee_service.update_employee(employee)
        print("Employee updated.")


    def delete_employee(self):
        emp_id = input("Enter Employee ID to delete: ")
        self.employee_service.remove_employee(emp_id)
        print("Employee deleted.")


    def generate_payroll(self):
```

```
emp_id = input("Enter Employee ID: ")

start_date = input("Enter Pay Period Start Date (YYYY-MM-DD): ")

end_date = input("Enter Pay Period End Date (YYYY-MM-DD): ")

basic_salary = float(input("Enter Basic Salary: "))

overtime = float(input("Enter Overtime Pay: "))

deductions = float(input("Enter Deductions: "))


# Calculate net salary

net_salary = basic_salary + overtime - deductions


# Create Payroll object

payroll = Payroll(

    employee_id=emp_id,

    pay_period_start_date=start_date,

    pay_period_end_date=end_date,

    basic_salary=basic_salary,

    overtime_pay=overtime,

    deductions=deductions,

    net_salary=net_salary

)


# Pass Payroll object to service

self.payroll_service.generate_payroll(payroll)

print("Payroll generated.")
```

```
    def calculate_tax(self):
        emp_id = input("Enter Employee ID: ")
        year = input("Enter Tax Year (e.g., 2024): ")
        self.tax_service.calculate_tax(emp_id, year)
        print("Tax calculated.")

    def view_payrolls_for_employee(self):
        emp_id = input("Enter Employee ID: ")
        payrolls = self.payroll_service.get_payrolls_for_employee(emp_id)

        if not payrolls:
            print("No payroll records found.")
            return

        for p in payrolls:
            print(f"""
    Payroll ID: {p.get_payroll_id()}
    Employee ID: {p.get_employee_id()}
    Pay Period: {p.get_pay_period_start_date()} to
{p.get_pay_period_end_date()}
    Basic Salary: {p.get_basic_salary()}
    Overtime Pay: {p.get_overtime_pay()}
    Deductions: {p.get_deductions()}
    Net Salary: {p.get_net_salary()}
    ----------------------------""")

    def view_taxes_for_employee(self):
        emp_id = input("Enter Employee ID: ")
        taxes = self.tax_service.get_taxes_for_employee(emp_id)

        if not taxes:
            print("No tax records found.")
            return

        for tax in taxes:
            print(f"""
    Tax ID: {tax.get_tax_id()}
    Employee ID: {tax.get_employee_id()}
    Tax Year: {tax.get_tax_year()}
    Taxable Income: ₹{tax.get_taxable_income():.2f}
    Tax Amount: ₹{tax.get_tax_amount():.2f}
    ----------------------------""")


# Entry point
if __name__ == "__main__":
    MainModule().menu()
```

4.3 Unit Testing:

Unit testing in the PayXpert system is implemented using the `pytest` framework to ensure the reliability and accuracy of core functionalities.

Test cases are written to validate salary calculations, tax computation, and error handling for invalid inputs.

These tests simulate real-world scenarios like processing payrolls for multiple employees and verifying high-income tax logic.

By running these automated tests, developers can confidently make changes without breaking existing features.

Unit testing improves code quality, supports debugging, and ensures system correctness before deployment.

Test Directory:

The **Test directory** contains unit test cases to verify the correctness of the application's core functionalities.

It uses the pytest framework to test payroll calculations, tax logic, and input validation.

This ensures the system behaves as expected and helps catch errors early during development.

Payxpert/test/test_payxpert.py

```
import pytest

from decimal import Decimal

from entity.Employee import Employee
from entity.Payroll import Payroll

from entity.Tax import Tax

from dao.payroll_service import PayrollService

from dao.tax_service import TaxService

from util.database_context import DatabaseContext

from util.validation_service import ValidationService

from exception.custom_exceptions import InvalidInputException


conn = DatabaseContext.get_connection()


#Test Case 1: Calculate Gross Salary

def test_calculate_gross_salary_for_employee():

    basic = 60000.00

    overtime = 5000.00

    gross = basic + overtime

    assert gross == 65000.00


#Test Case 2: Calculate Net Salary

def test_calculate_net_salary_after_deductions():

    basic = 60000.00

    overtime = 5000.00

    deductions = 2000.00

    net = basic + overtime - deductions

    assert net == 63000.00
```

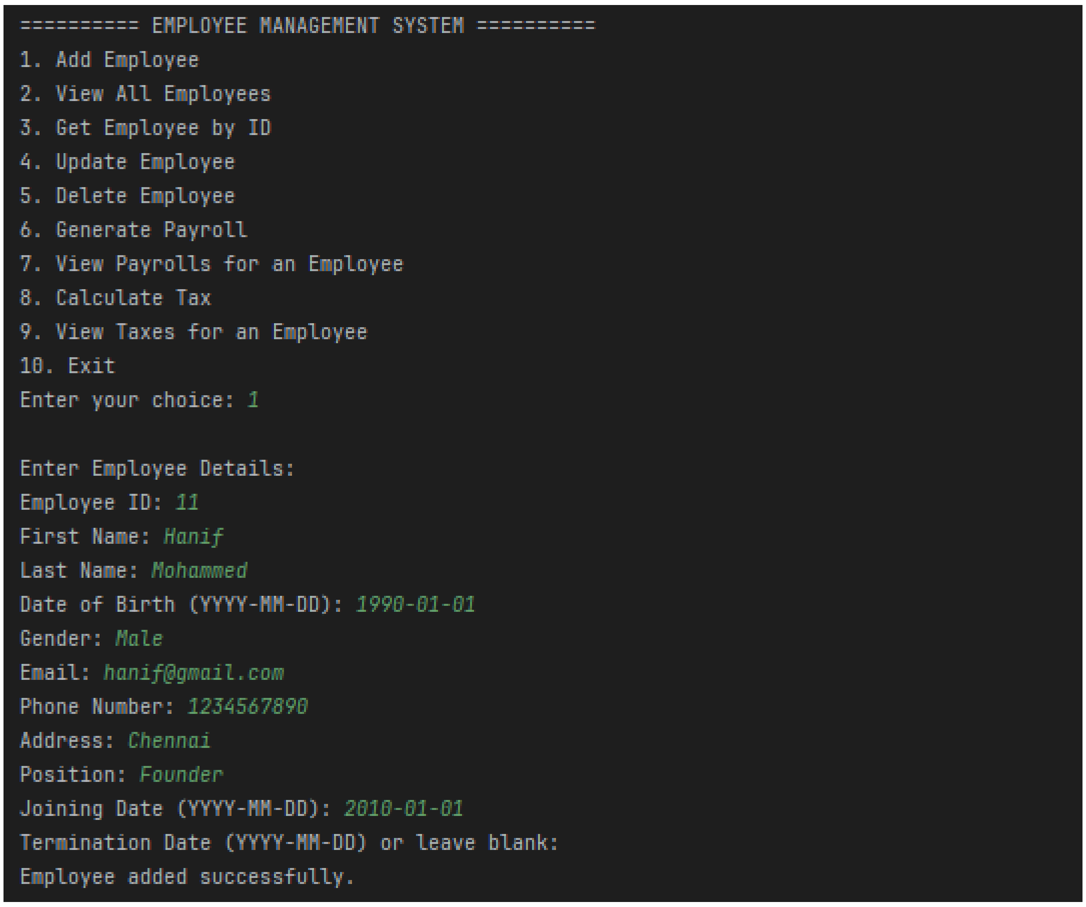
```
#Test Case 3: Tax Calculation for High Income

def test_verify_tax_calculation_for_high_income_employee():

    tax_service = TaxService(conn)

    high_income_tax = Tax(tax_id=1, employee_id=1, tax_year=2024,
taxable_income=Decimal('1200000.00'), tax_amount=Decimal('0.00'))


high_income_tax.set_tax_amount(high_income_tax.get_taxable_income
() * Decimal('0.10'))

    assert high_income_tax.get_tax_amount() == Decimal('120000.00')


# Test Case 4: Process Payroll for Multiple Employees

def test_process_payroll_for_multiple_employees():

    payroll_service = PayrollService(conn)

    employee_ids = [1, 2, 3]



    count = 0

    for i in employee_ids:

        payroll = Payroll(payroll_id=None, employee_id=i,
basic_salary=40000, overtime_pay=2000,

                          deductions=1000, net_salary=41000)

        payroll_service.generate_payroll(payroll)

        count += 1



    assert count == 3
```

```
#Test Case 5: Error Handling for Invalid Input

def test_verify_error_handling_for_invalid_email():

    invalid_email = "john[at]email.com"

    with pytest.raises(Exception):

        if not ValidationService.validate_email(invalid_email):

            raise InvalidInputException("Invalid Email Format")
```

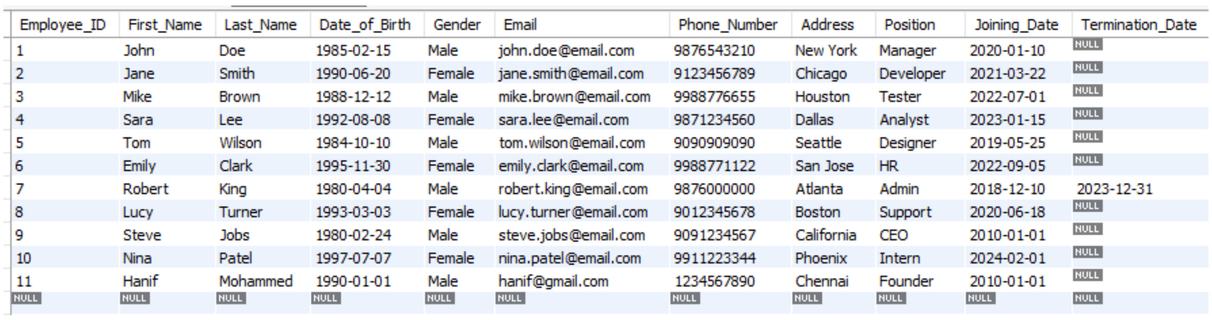
OUTPUT

Running the MainModule.py file launches a menu-driven console application for managing employees, payroll, and taxes.
Users can interactively perform operations like adding employees, generating payroll, and viewing reports with real-time database updates.
The system ensures smooth navigation, accurate data handling, and displays results in a clean, readable format.

1. **OUTPUT while choosing 1:**

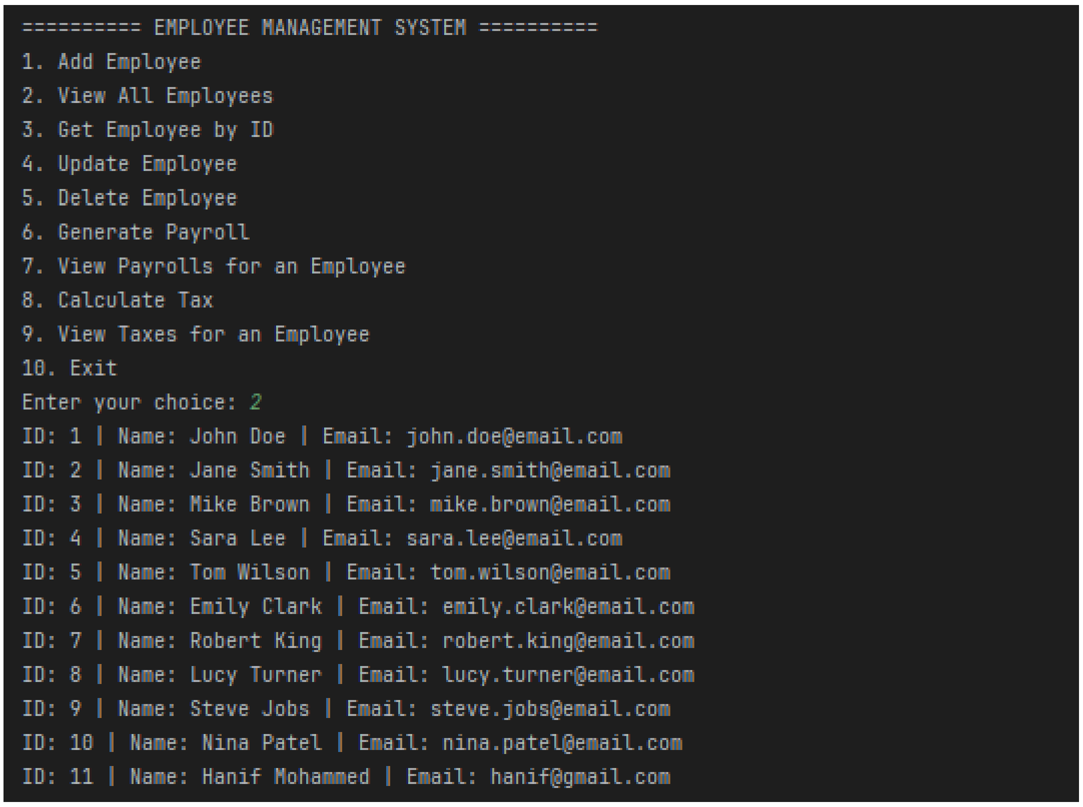


```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 1

Enter Employee Details:
Employee ID: 11
First Name: Hanif
Last Name: Mohammed
Date of Birth (YYYY-MM-DD): 1990-01-01
Gender: Male
Email: hanif@gmail.com
Phone Number: 1234567890
Address: Chennai
Position: Founder
Joining Date (YYYY-MM-DD): 2010-01-01
Termination Date (YYYY-MM-DD) or leave blank:
Employee added successfully.
```



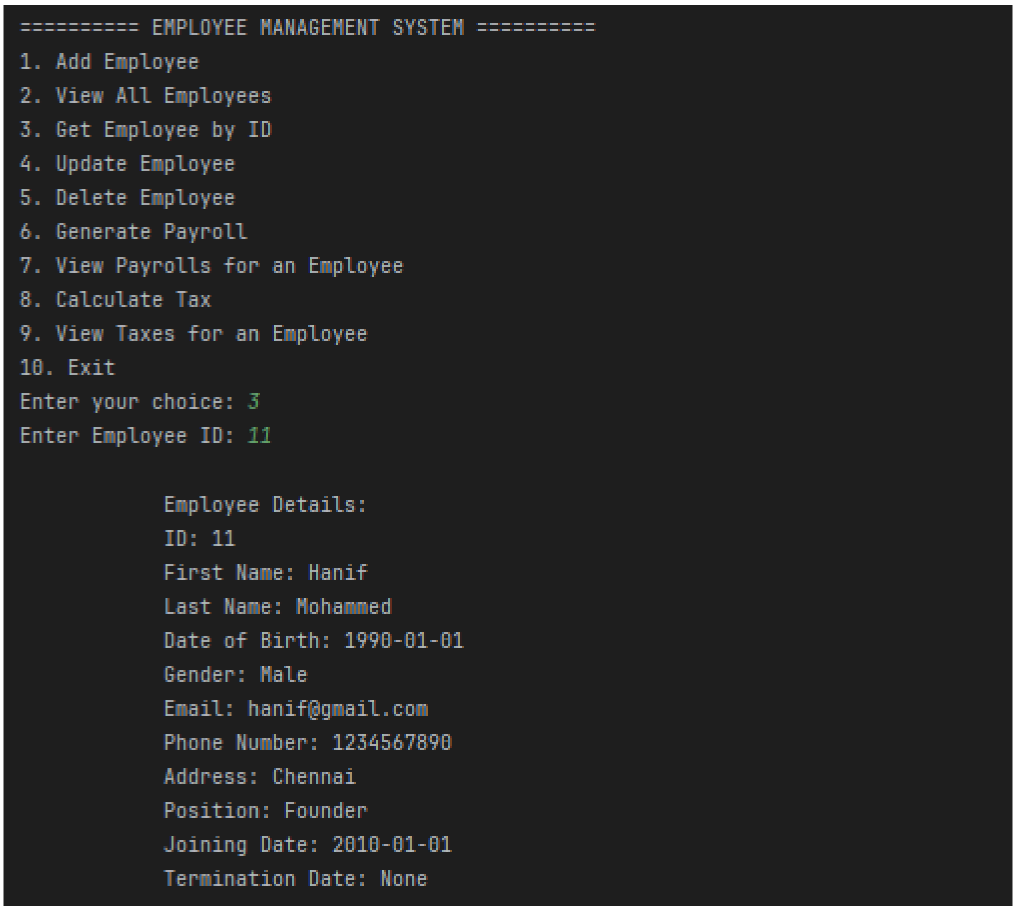
Employee_ID	First_Name	Last_Name	Date_of_Birth	Gender	Email	Phone_Number	Address	Position	Joining_Date	Termination_Date
1	John	Doe	1985-02-15	Male	john.doe@email.com	9876543210	New York	Manager	2020-01-10	NULL
2	Jane	Smith	1990-06-20	Female	jane.smith@email.com	9123456789	Chicago	Developer	2021-03-22	NULL
3	Mike	Brown	1988-12-12	Male	mike.brown@email.com	9988776655	Houston	Tester	2022-07-01	NULL
4	Sara	Lee	1992-08-08	Female	sara.lee@email.com	9871234560	Dallas	Analyst	2023-01-15	NULL
5	Tom	Wilson	1984-10-10	Male	tom.wilson@email.com	9090909090	Seattle	Designer	2019-05-25	NULL
6	Emily	Clark	1995-11-30	Female	emily.clark@email.com	9988771122	San Jose	HR	2022-09-05	NULL
7	Robert	King	1980-04-04	Male	robert.king@email.com	9876000000	Atlanta	Admin	2018-12-10	2023-12-31
8	Lucy	Turner	1993-03-03	Female	lucy.turner@email.com	9012345678	Boston	Support	2020-06-18	NULL
9	Steve	Jobs	1980-02-24	Male	steve.jobs@email.com	9091234567	California	CEO	2010-01-01	NULL
10	Nina	Patel	1997-07-07	Female	nina.patel@email.com	9911223344	Phoenix	Intern	2024-02-01	NULL
11	Hanif	Mohammed	1990-01-01	Male	hanif@gmail.com	1234567890	Chennai	Founder	2010-01-01	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2.OUTPUT while choosing 2:



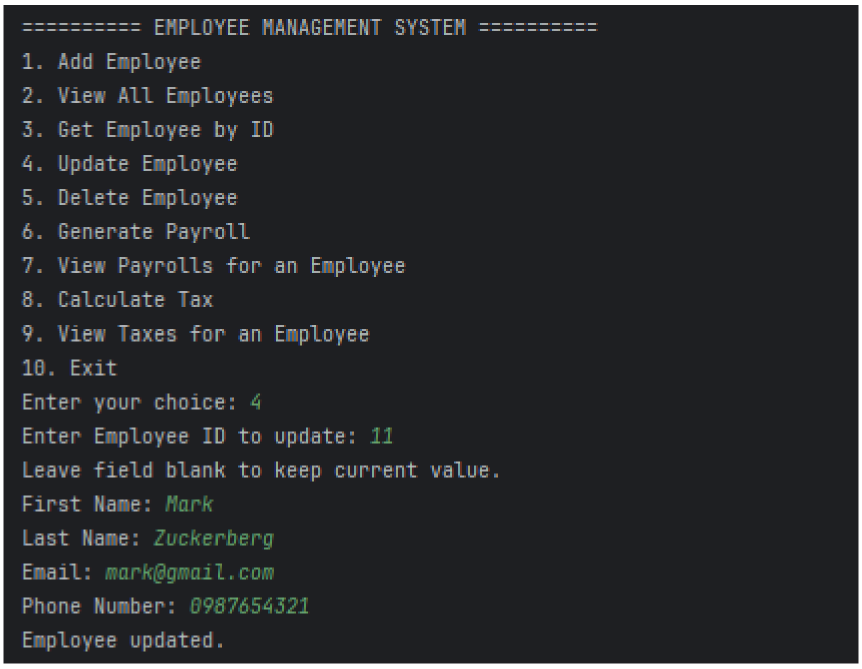
```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 2
ID: 1 | Name: John Doe | Email: john.doe@email.com
ID: 2 | Name: Jane Smith | Email: jane.smith@email.com
ID: 3 | Name: Mike Brown | Email: mike.brown@email.com
ID: 4 | Name: Sara Lee | Email: sara.lee@email.com
ID: 5 | Name: Tom Wilson | Email: tom.wilson@email.com
ID: 6 | Name: Emily Clark | Email: emily.clark@email.com
ID: 7 | Name: Robert King | Email: robert.king@email.com
ID: 8 | Name: Lucy Turner | Email: lucy.turner@email.com
ID: 9 | Name: Steve Jobs | Email: steve.jobs@email.com
ID: 10 | Name: Nina Patel | Email: nina.patel@email.com
ID: 11 | Name: Hanif Mohammed | Email: hanif@gmail.com
```

3.OUTPUT while choosing 3:

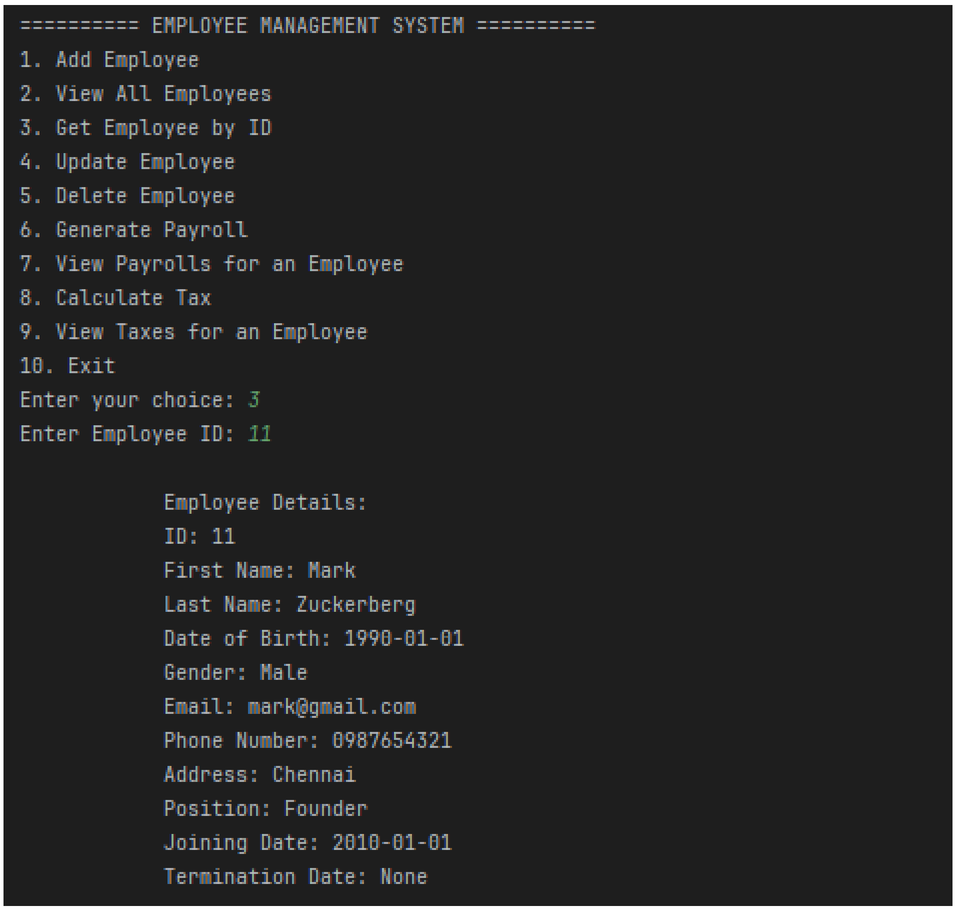


```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 3
Enter Employee ID: 11

            Employee Details:
            ID: 11
            First Name: Hanif
            Last Name: Mohammed
            Date of Birth: 1990-01-01
            Gender: Male
            Email: hanif@gmail.com
            Phone Number: 1234567890
            Address: Chennai
            Position: Founder
            Joining Date: 2010-01-01
            Termination Date: None
```

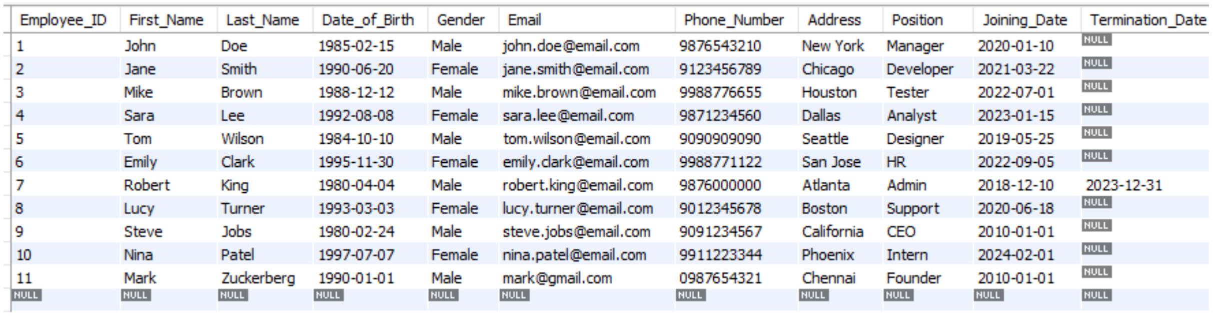
4.OUTPUT while choosing 4:



```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 4
Enter Employee ID to update: 11
Leave field blank to keep current value.
First Name: Mark
Last Name: Zuckerberg
Email: mark@gmail.com
Phone Number: 0987654321
Employee updated.
```

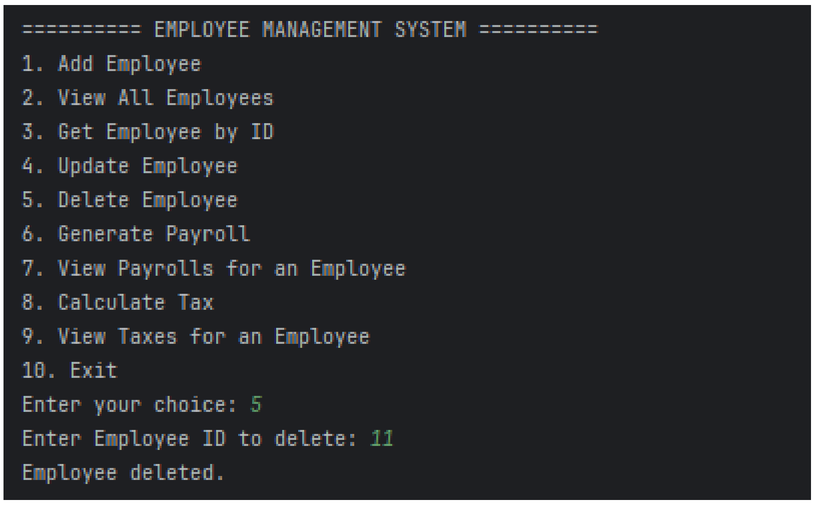


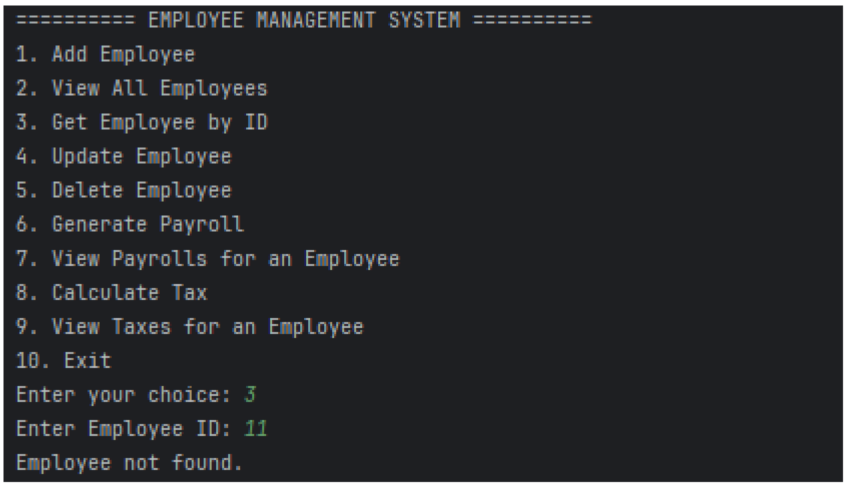
```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 3
Enter Employee ID: 11

            Employee Details:
            ID: 11
            First Name: Mark
            Last Name: Zuckerberg
            Date of Birth: 1990-01-01
            Gender: Male
            Email: mark@gmail.com
            Phone Number: 0987654321
            Address: Chennai
            Position: Founder
            Joining Date: 2010-01-01
            Termination Date: None
```



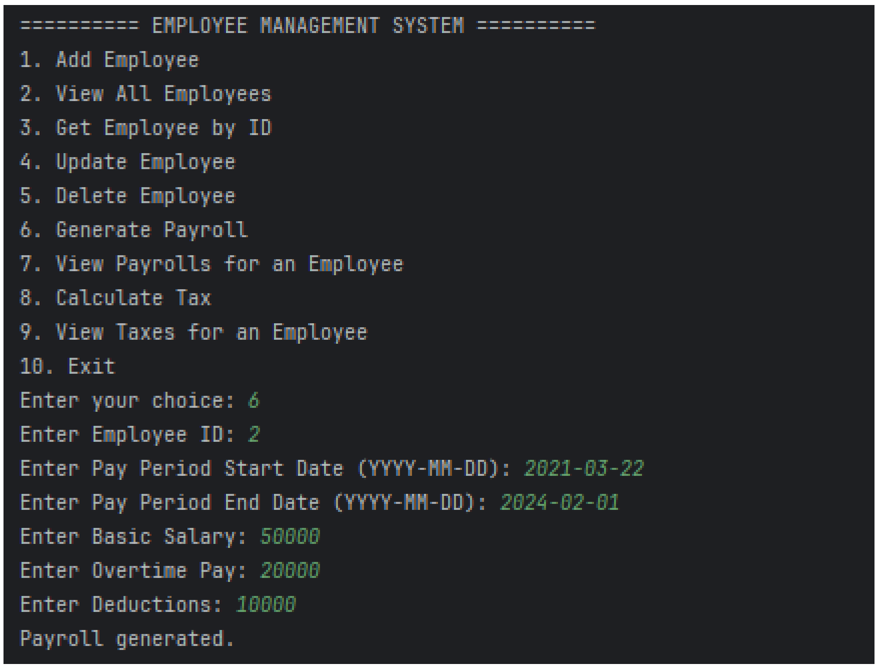
Employee_ID	First_Name	Last_Name	Date_of_Birth	Gender	Email	Phone_Number	Address	Position	Joining_Date	Termination_Date
1	John	Doe	1985-02-15	Male	john.doe@email.com	9876543210	New York	Manager	2020-01-10	NULL
2	Jane	Smith	1990-06-20	Female	jane.smith@email.com	9123456789	Chicago	Developer	2021-03-22	NULL
3	Mike	Brown	1988-12-12	Male	mike.brown@email.com	9988776655	Houston	Tester	2022-07-01	NULL
4	Sara	Lee	1992-08-08	Female	sara.lee@email.com	9871234560	Dallas	Analyst	2023-01-15	NULL
5	Tom	Wilson	1984-10-10	Male	tom.wilson@email.com	9090909090	Seattle	Designer	2019-05-25	NULL
6	Emily	Clark	1995-11-30	Female	emily.clark@email.com	9988771122	San Jose	HR	2022-09-05	NULL
7	Robert	King	1980-04-04	Male	robert.king@email.com	9876000000	Atlanta	Admin	2018-12-10	2023-12-31
8	Lucy	Turner	1993-03-03	Female	lucy.turner@email.com	9012345678	Boston	Support	2020-06-18	NULL
9	Steve	Jobs	1980-02-24	Male	steve.jobs@email.com	9091234567	California	CEO	2010-01-01	NULL
10	Nina	Patel	1997-07-07	Female	nina.patel@email.com	9911223344	Phoenix	Intern	2024-02-01	NULL
11	Mark	Zuckerberg	1990-01-01	Male	mark@gmail.com	0987654321	Chennai	Founder	2010-01-01	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5.OUTPUT while choosing 5:


```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 5
Enter Employee ID to delete: 11
Employee deleted.
```

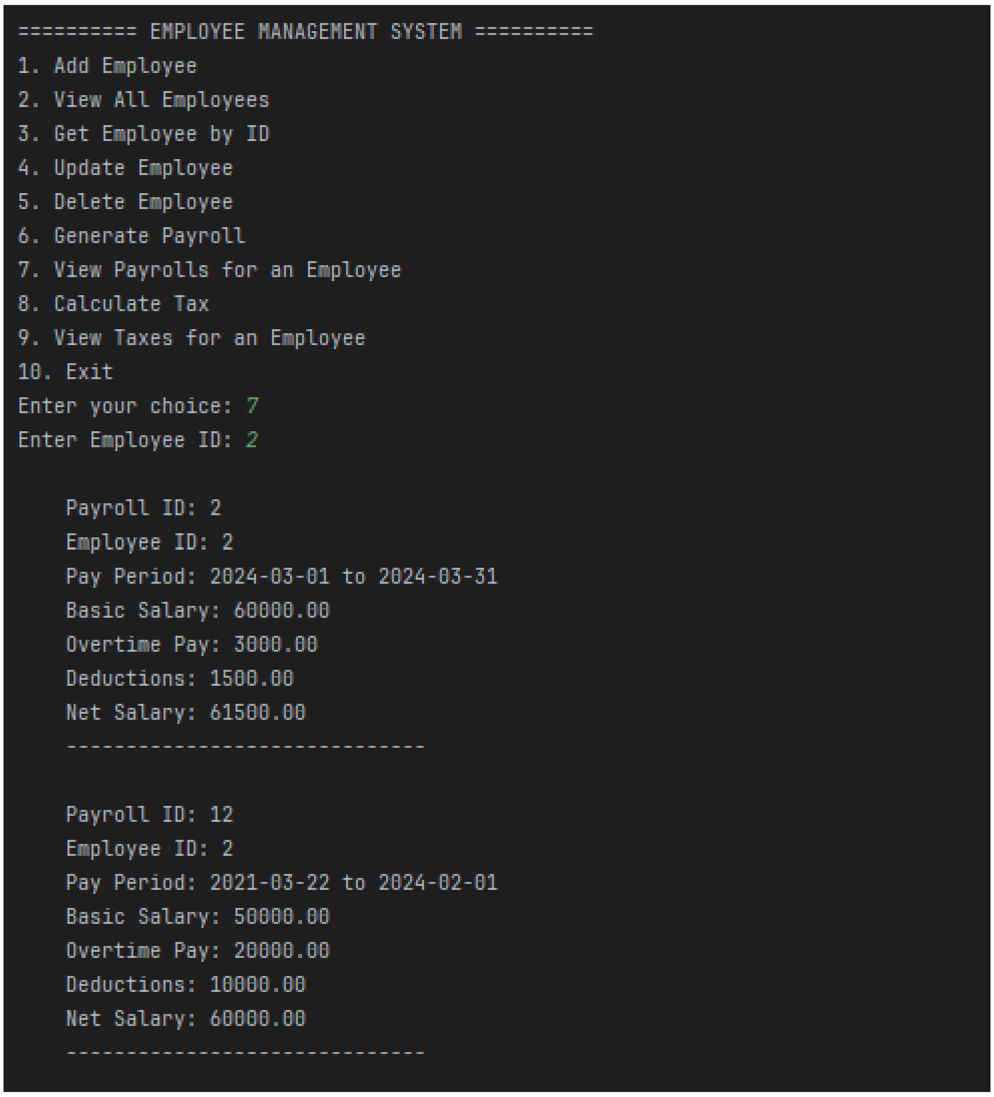

```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 3
Enter Employee ID: 11
Employee not found.
```

6.OUTPUT while choosing 6:



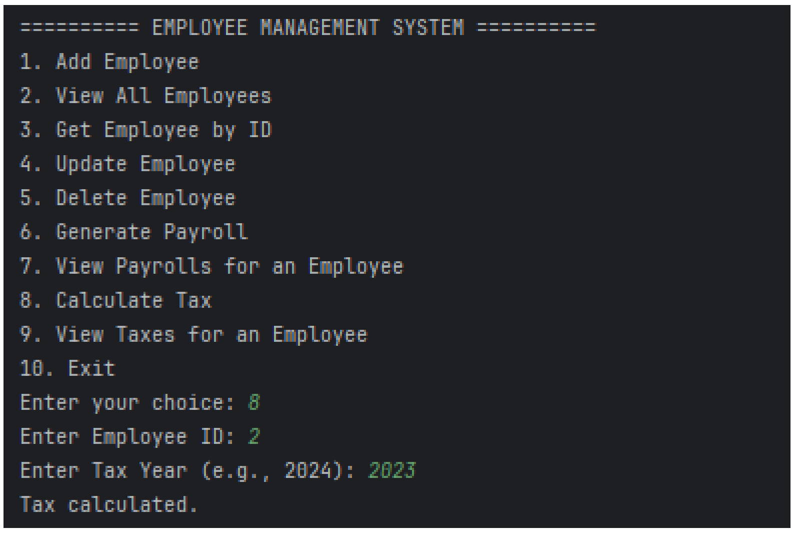
```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 6
Enter Employee ID: 2
Enter Pay Period Start Date (YYYY-MM-DD): 2021-03-22
Enter Pay Period End Date (YYYY-MM-DD): 2024-02-01
Enter Basic Salary: 50000
Enter Overtime Pay: 20000
Enter Deductions: 10000
Payroll generated.
```

7.OUTPUT while choosing 7:



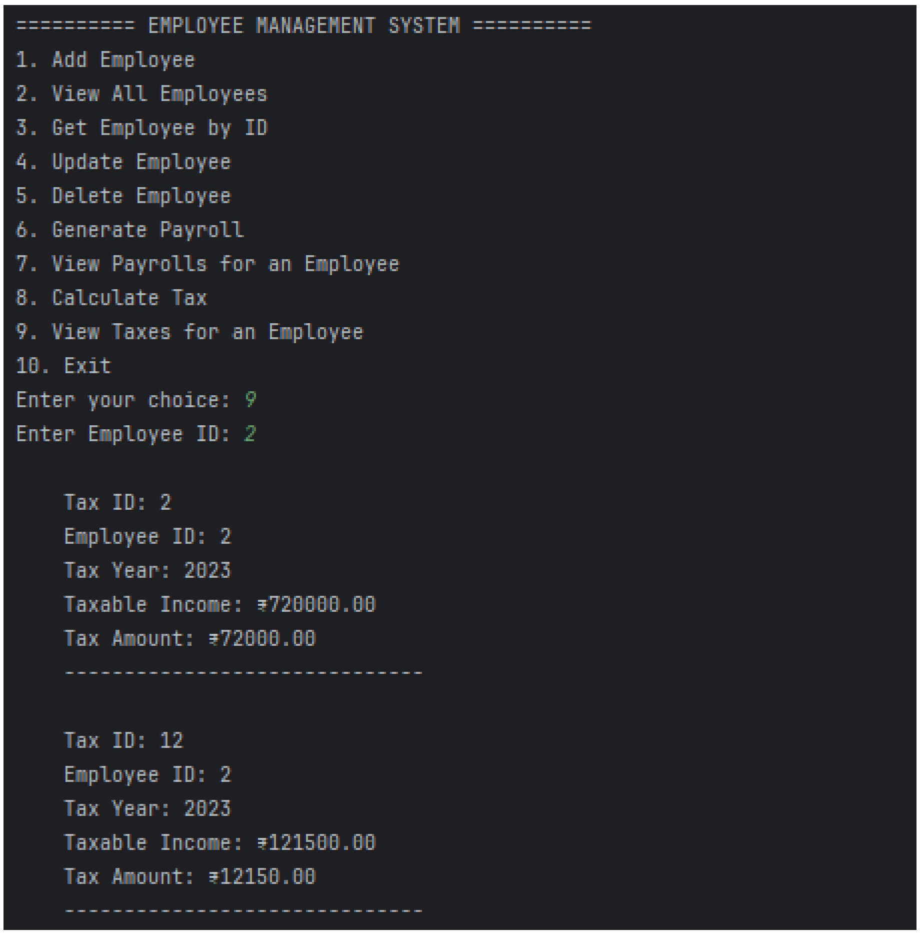
```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 7
Enter Employee ID: 2

    Payroll ID: 2
    Employee ID: 2
    Pay Period: 2024-03-01 to 2024-03-31
    Basic Salary: 60000.00
    Overtime Pay: 3000.00
    Deductions: 1500.00
    Net Salary: 61500.00
    ------------------------------

    Payroll ID: 12
    Employee ID: 2
    Pay Period: 2021-03-22 to 2024-02-01
    Basic Salary: 50000.00
    Overtime Pay: 20000.00
    Deductions: 10000.00
    Net Salary: 60000.00
    ------------------------------
```

8.OUTPUT while choosing 8:



```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 8
Enter Employee ID: 2
Enter Tax Year (e.g., 2024): 2023
Tax calculated.
```

9.OUTPUT while choosing 9:



```
========== EMPLOYEE MANAGEMENT SYSTEM ==========
1. Add Employee
2. View All Employees
3. Get Employee by ID
4. Update Employee
5. Delete Employee
6. Generate Payroll
7. View Payrolls for an Employee
8. Calculate Tax
9. View Taxes for an Employee
10. Exit
Enter your choice: 9
Enter Employee ID: 2

    Tax ID: 2
    Employee ID: 2
    Tax Year: 2023
    Taxable Income: ₹720000.00
    Tax Amount: ₹72000.00
    ------------------------------

    Tax ID: 12
    Employee ID: 2
    Tax Year: 2023
    Taxable Income: ₹121500.00
    Tax Amount: ₹12150.00
    ------------------------------
```