# PAYXPERT PAYROLL MANAGEMENT SYSTEM - CASE STUDY REPORT

**Title:** PayXpert Payroll Management System
**Submitted by:** Maha Lakshmi R, Esaq A

## Abstract

The PayXpert Payroll Management System is a comprehensive, modular software solution designed to automate and streamline payroll processes for organizations. It simplifies the management of employee records, salary calculations, tax deductions, and financial reporting through a menu-driven, user-friendly interface.

PayXpert follows object-oriented design principles and implements a layered architecture to ensure clean code, scalability, and maintainability. The system integrates with a secure SQL database to store employee information, payroll records, tax details, and financial transactions.

Key features of the system include automated payroll generation, dynamic net salary calculations considering overtime, deductions, and taxes, as well as detailed financial reporting for management insights. The solution incorporates robust exception handling through custom exceptions to manage errors such as invalid inputs, missing records, and database connectivity issues.

PayXpert emphasizes accurate financial computations, secure data handling, and efficient record management, providing organizations with a reliable tool to ensure payroll accuracy, tax compliance, and streamlined HR operations.

## Introduction

The PayXpert Payroll Management System is designed to automate and simplify payroll processing for organizations. This project focuses on developing a menu-driven, modular system that efficiently manages employee records, payroll calculations, tax details, and financial reports while ensuring database integrity and user-friendly interaction.

## Objectives

- Automate payroll generation and record management.
- Maintain accurate employee records including personal, financial, and employment details.
- Calculate net salary considering overtime, deductions, and applicable taxes.
- Store all records in a secure MySQL database.
- Implement proper exception handling and input validations.
- Provide modular, scalable, and maintainable code structure.

## System Architecture

| Layer | Description |
|---|---|
| **Entity/Model Layer** | Contains simple classes representing real-world entities like Employee, Payroll, Tax, FinancialRecord. These classes hold only data with private properties, getters, setters, and constructors. |
| **DAO/Service Layer** | Defines interfaces and their implementations for performing business operations like employee management, payroll generation, tax calculation and financial record management.This layer interacts with DB |
| **Exception Layer** | Contains custom exception classes to handle application-specific errors, enhancing system reliability and user experience. |
| **Utility Layer** | Provides reusable components for database connectivity and configuration management. Ensures centralized handling of connections and properties. |
| **Main Application Layer** | The entry point of the system, presenting a menu-driven interface that allows users to access various functionalities interactively. |

## Database Design

Create following tables in SQL Schema with appropriate class and write the unit test case for the application. SQL Tables:

### Employee Table

- EmployeeID (Primary Key, INT)
- FirstName, LastName (VARCHAR)
- DateOfBirth (DATE)
- Gender (VARCHAR)
- Email (VARCHAR)
- PhoneNumber (VARCHAR)
- Address (VARCHAR)
- Position (VARCHAR)
- JoiningDate (DATE)
- TerminationDate (DATE, Nullable)

**Query:**

create table employee (employeeid int primary key, firstname varchar(50) not null, lastname varchar(50) not null, dateofbirth date not null, gender varchar(10) not null, email varchar(100) not null, phonenumber varchar(15), address varchar(50), position varchar(50) not null, joiningdate date not null, terminationdate date null);

```
mysql> create database PayXpert;
Query OK, 1 row affected (0.05 sec)

mysql> use payXpert;
Database changed
mysql> CREATE TABLE Employee (EmployeeID INT PRIMARY KEY, FirstName VARCHAR(50) NOT NULL, LastName VARCHAR(50) NOT NULL, DateOfBirth
DATE NOT NULL, Gender VARCHAR(10) NOT NULL, Email VARCHAR(100) NOT NULL, PhoneNumber VARCHAR(15), Address VARCHAR(255), Position VARC
HAR(50) NOT NULL, JoiningDate DATE NOT NULL, TerminationDate DATE NULL);
Query OK, 0 rows affected (0.09 sec)
```

## Payroll Table

- PayrollID (Primary Key, INT)
- EmployeeID (Foreign Key)
- PayPeriodStartDate, PayPeriodEndDate (DATE)
- BasicSalary, OvertimePay, Deductions, NetSalary (DECIMAL)

**Query:**

create table payroll (payrollid int primary key, employeeid int, payperiodstartdate date not null, payperiodenddate date not null, basicsalary decimal(10,2) not null, overtimepay decimal(10,2), deductions decimal(10,2), netsalary decimal(10,2), foreign key (employeeid) references employee(employeeid));

```
mysql> create table payroll (payrollid int primary key, employeeid int, payperiodstartdate date not null, payperiodenddate date not n
ull, basicsalary decimal(10,2) not null, overtimepay decimal(10,2), deductions decimal(10,2), netsalary decimal(10,2), foreign key (e
mployeeid) references employee(employeeid));
Query OK, 0 rows affected (0.16 sec)

mysql>
```

## Tax Table

- TaxID (Primary Key, INT)
- EmployeeID (Foreign Key)
- TaxAmount (DECIMAL)
- TaxYear (INT)

**Query:**

create table tax (taxid int primary key, employeeid int, taxyear int not null, taxableincome decimal(10,2) not null, taxamount decimal(10,2) not null, foreign key (employeeid) references employee(employeeid));

```
mysql> create table tax (taxid int primary key, employeeid int, taxyear int not null, taxableincome decimal(10,2) not null, taxamount
 decimal(10,2) not null, foreign key (employeeid) references employee(employeeid));
Query OK, 0 rows affected (0.12 sec)
```

## Financial Report Table

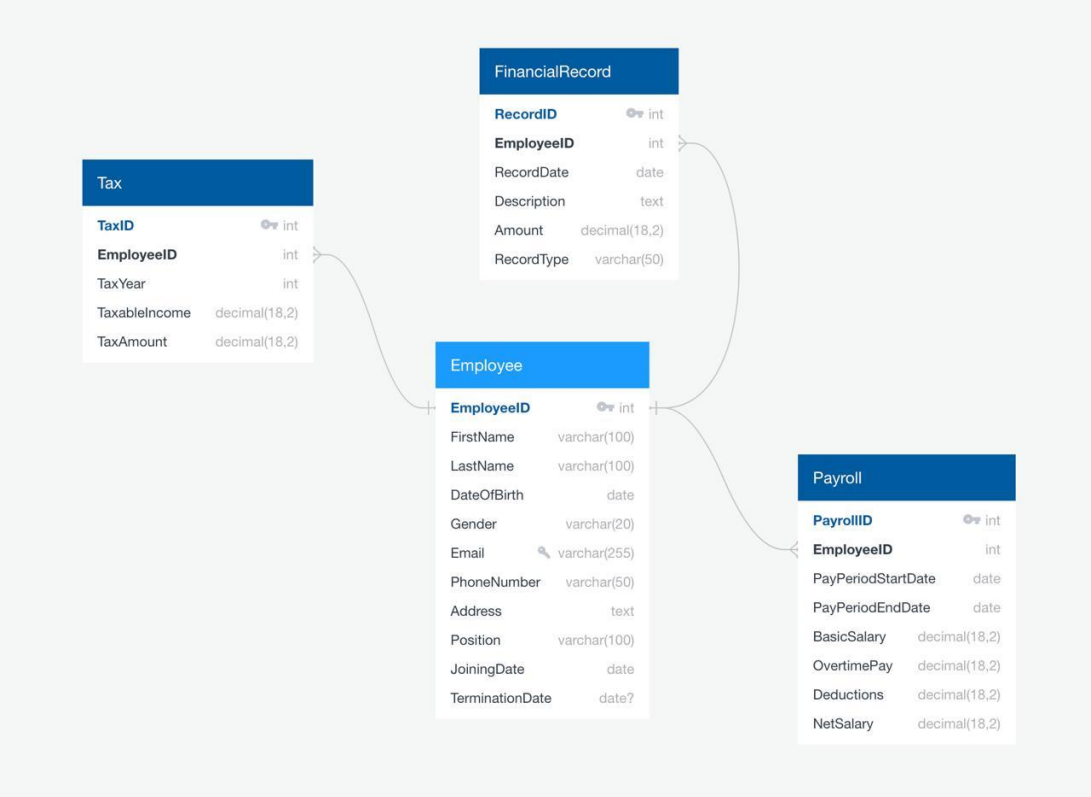
- RecordID.
- EmployeeID
- RecordDate:
- Description:
- Amount
- RecordType

**Query:**

create table financialrecord (recordid int primary key, employeeid int, recorddate date not null, description varchar(255), amount decimal(10,2) not null, recordtype varchar(50) not null, foreign key (employeeid) references employee(employeeid));

```
mysql> create table financialrecord (recordid int primary key, employeeid int, recorddate date not null, description varchar(255), am
ount decimal(10,2) not null, recordtype varchar(50) not null, foreign key (employeeid) references employee(employeeid));
ERROR 1050 (42S01): Table 'financialrecord' already exists
mysql>
```

# ER Diagram



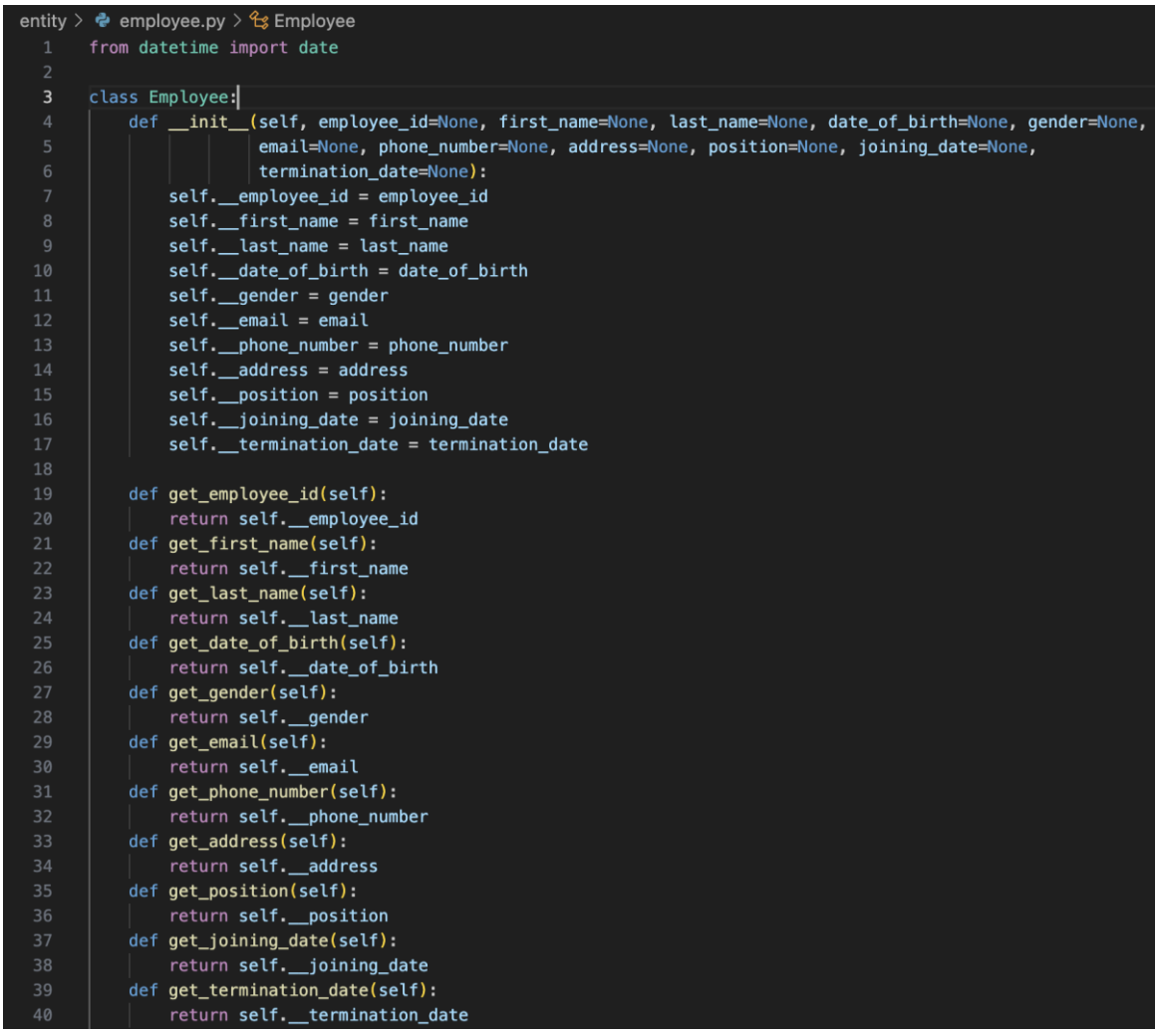
Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters )

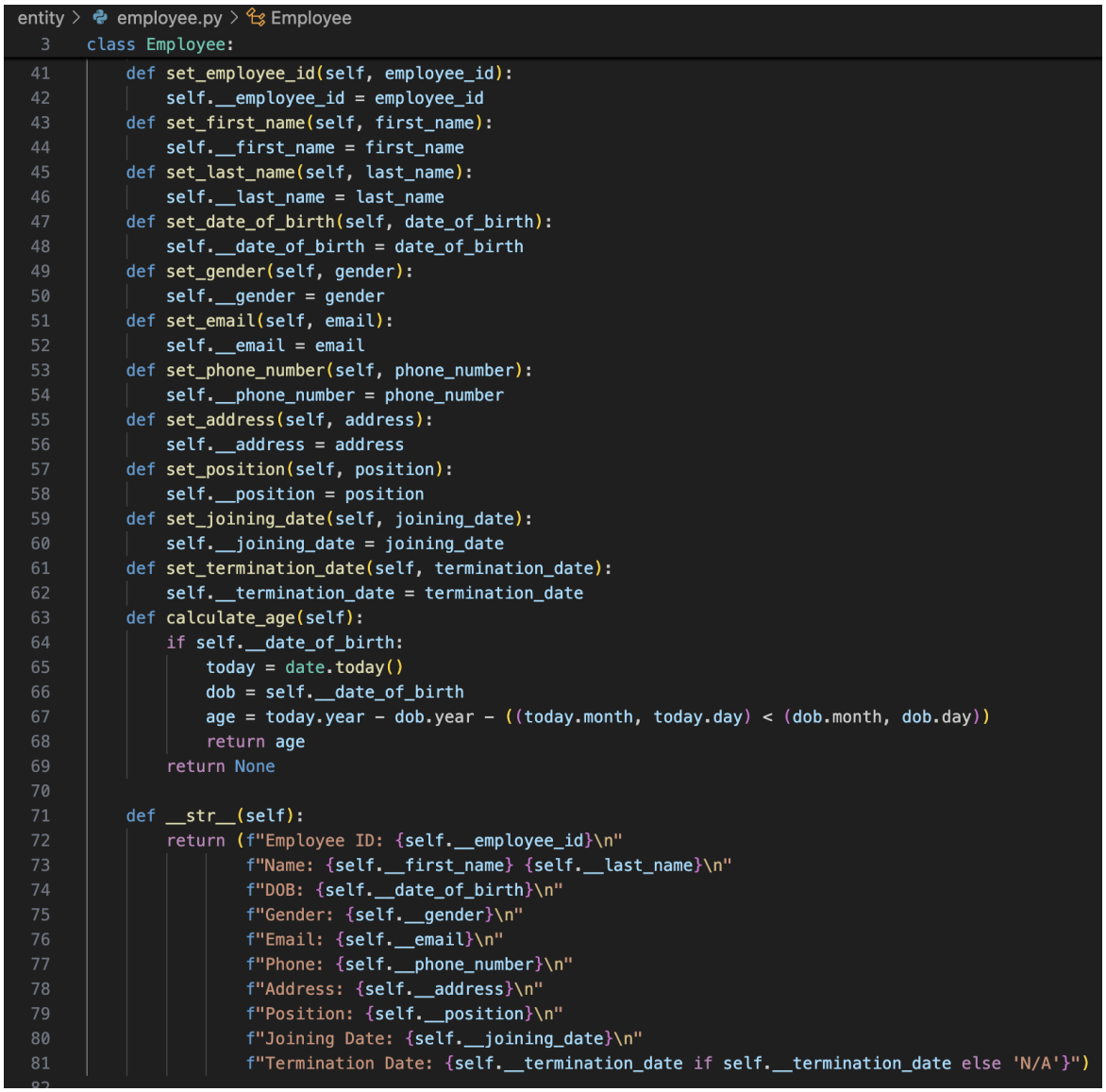
# Classes:

## Employee:

- Properties: EmployeeID, FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber, Address, Position, JoiningDate, TerminationDate
- Methods: CalculateAge()

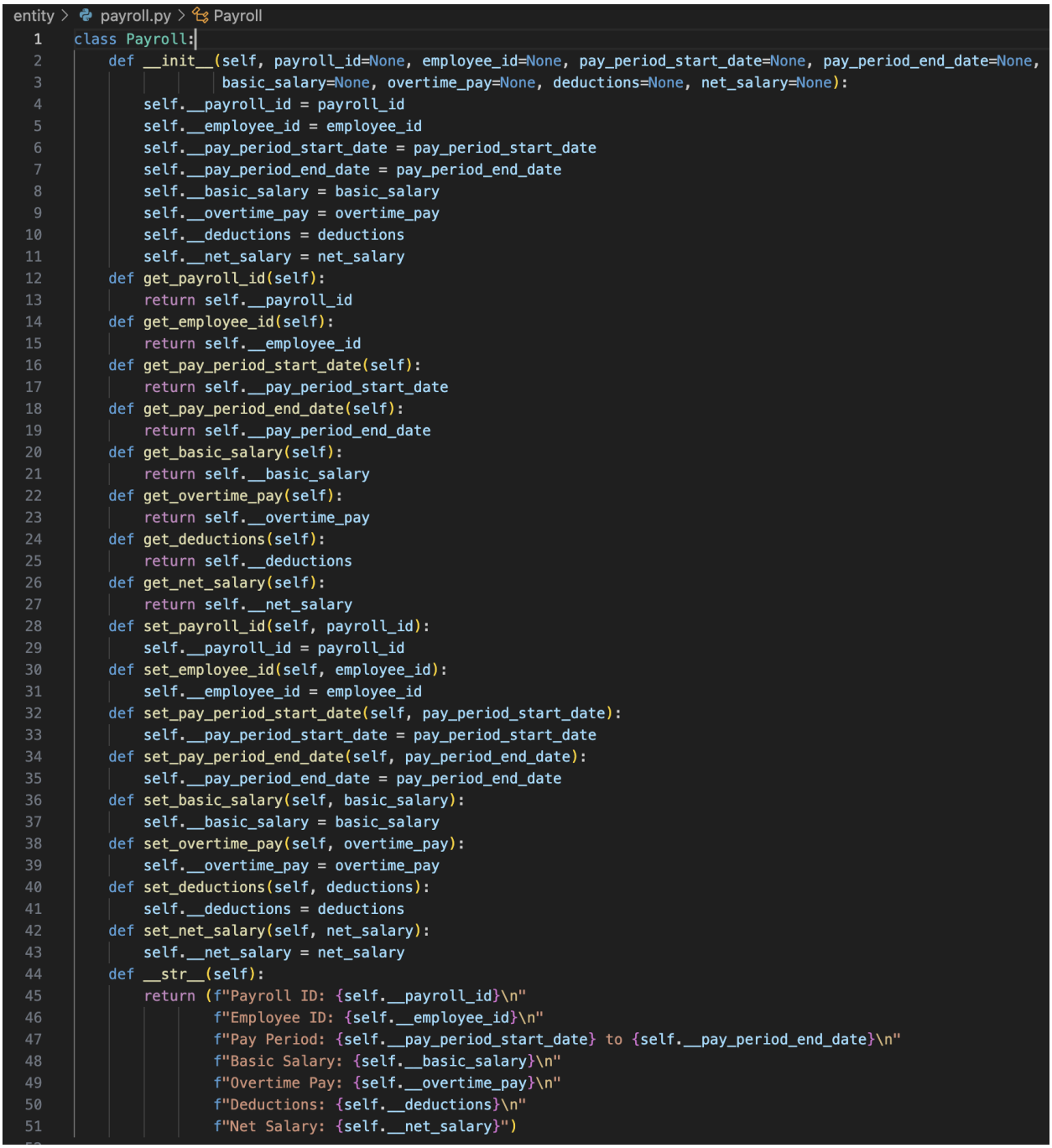
Code:




entity > employee.py > Employee
3 class Employee:
41 def set_employee_id(self, employee_id):
42 self.__employee_id = employee_id
43 def set_first_name(self, first_name):
44 self.__first_name = first_name
45 def set_last_name(self, last_name):
46 self.__last_name = last_name
47 def set_date_of_birth(self, date_of_birth):
48 self.__date_of_birth = date_of_birth
49 def set_gender(self, gender):
50 self.__gender = gender
51 def set_email(self, email):
52 self.__email = email
53 def set_phone_number(self, phone_number):
54 self.__phone_number = phone_number
55 def set_address(self, address):
56 self.__address = address
57 def set_position(self, position):
58 self.__position = position
59 def set_joining_date(self, joining_date):
60 self.__joining_date = joining_date
61 def set_termination_date(self, termination_date):
62 self.__termination_date = termination_date
63 def calculate_age(self):
64 if self.__date_of_birth:
65 today = date.today()
66 dob = self.__date_of_birth
67 age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))
68 return age
69 return None
70
71 def __str__(self):
72 return (f"Employee ID: {self.__employee_id}\n"
73 f"Name: {self.__first_name} {self.__last_name}\n"
74 f"DOB: {self.__date_of_birth}\n"
75 f"Gender: {self.__gender}\n"
76 f"Email: {self.__email}\n"
77 f"Phone: {self.__phone_number}\n"
78 f"Address: {self.__address}\n"
79 f"Position: {self.__position}\n"
80 f"Joining Date: {self.__joining_date}\n"
81 f"Termination Date: {self.__termination_date if self.__termination_date else 'N/A'}")

## Payroll:

- Properties: PayrollID, EmployeeID, PayPeriodStartDate, PayPeriodEndDate, BasicSalary, OvertimePay, Deductions, NetSalary



```
entity > payroll.py > Payroll
class Payroll:
    def __init__(self, payroll_id=None, employee_id=None, pay_period_start_date=None, pay_period_end_date=None,
                 basic_salary=None, overtime_pay=None, deductions=None, net_salary=None):
        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 get_employee_id(self):
        return self.__employee_id
    def get_pay_period_start_date(self):
        return self.__pay_period_start_date
    def get_pay_period_end_date(self):
        return self.__pay_period_end_date
    def get_basic_salary(self):
        return self.__basic_salary
    def get_overtime_pay(self):
        return self.__overtime_pay
    def get_deductions(self):
        return self.__deductions
    def get_net_salary(self):
        return self.__net_salary
    def set_payroll_id(self, payroll_id):
        self.__payroll_id = payroll_id
    def set_employee_id(self, employee_id):
        self.__employee_id = employee_id
    def set_pay_period_start_date(self, pay_period_start_date):
        self.__pay_period_start_date = pay_period_start_date
    def set_pay_period_end_date(self, pay_period_end_date):
        self.__pay_period_end_date = pay_period_end_date
    def set_basic_salary(self, basic_salary):
        self.__basic_salary = basic_salary
    def set_overtime_pay(self, overtime_pay):
        self.__overtime_pay = overtime_pay
    def set_deductions(self, deductions):
        self.__deductions = deductions
    def set_net_salary(self, net_salary):
        self.__net_salary = net_salary
    def __str__(self):
        return (f"Payroll ID: {self.__payroll_id}\n"
                f"Employee ID: {self.__employee_id}\n"
                f"Pay Period: {self.__pay_period_start_date} to {self.__pay_period_end_date}\n"
                f"Basic Salary: {self.__basic_salary}\n"
                f"Overtime Pay: {self.__overtime_pay}\n"
                f"Deductions: {self.__deductions}\n"
                f"Net Salary: {self.__net_salary}")
```

## Tax:

- Properties: TaxID, EmployeeID, TaxYear, TaxableIncome, TaxAmount

entity > tax.py > Tax

```python
class Tax:
    def __init__(self, tax_id=None, employee_id=None, tax_year=None, taxable_income=None, tax_amount=None):
        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 get_employee_id(self):
        return self.__employee_id

    def get_tax_year(self):
        return self.__tax_year

    def get_taxable_income(self):
        return self.__taxable_income

    def get_tax_amount(self):
        return self.__tax_amount

    def set_tax_id(self, tax_id):
        self.__tax_id = tax_id

    def set_employee_id(self, employee_id):
        self.__employee_id = employee_id

    def set_tax_year(self, tax_year):
        self.__tax_year = tax_year

    def set_taxable_income(self, taxable_income):
        self.__taxable_income = taxable_income

    def set_tax_amount(self, tax_amount):
        self.__tax_amount = tax_amount

    def __str__(self):
        return (f"Tax ID: {self.__tax_id}\n"
                f"Employee ID: {self.__employee_id}\n"
                f"Tax Year: {self.__tax_year}\n"
                f"Taxable Income: {self.__taxable_income}\n"
                f"Tax Amount: {self.__tax_amount}")
```

## FinancialRecord:

- Properties: RecordID, EmployeeID, RecordDate, Description, Amount, RecordType

entity > financial_record.py > FinancialRecord > set_amount

```python
class FinancialRecord:
    def __init__(self, record_id=None, employee_id=None, record_date=None, description=None, amount=None,
                 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 get_employee_id(self):
        return self.__employee_id
    def get_record_date(self):
        return self.__record_date
    def get_description(self):
        return self.__description
    def get_amount(self):
        return self.__amount
    def get_record_type(self):
        return self.__record_type

    def set_record_id(self, record_id):
        self.__record_id = record_id
    def set_employee_id(self, employee_id):
        self.__employee_id = employee_id
    def set_record_date(self, record_date):
        self.__record_date = record_date
    def set_description(self, description):
        self.__description = description
    def set_amount(self, amount):
        self.__amount = amount
    def set_record_type(self, record_type):
        self.__record_type = record_type

    def __str__(self):
        return (f"Record ID: {self.__record_id}\n"
                f"Employee ID: {self.__employee_id}\n"
                f"Record Date: {self.__record_date}\n"
                f"Description: {self.__description}\n"
                f"Amount: {self.__amount}\n"
                f"Record Type: {self.__record_type}")

```

## EmployeeService (implements IEmployeeService):

- Methods: GetEmployeeById, GetAllEmployees, AddEmployee, UpdateEmployee, RemoveEmployee

```python
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_data):
        pass

    @abstractmethod
    def update_employee(self, employee_data):
        pass

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

## PayrollService (implements IPayrollService):

- Methods: GeneratePayroll, GetPayrollById, GetPayrollsForEmployee, GetPayrollsForPeriod

```python
from abc import ABC, abstractmethod
from datetime import date

class IPayrollService(ABC):
    @abstractmethod
    def generate_payroll(self, employee_id: int, start_date_obj: date, end_date_obj: date,
                         basic_salary: float, overtime_pay: float = 0.0, deductions: float = 0.0):
        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
```

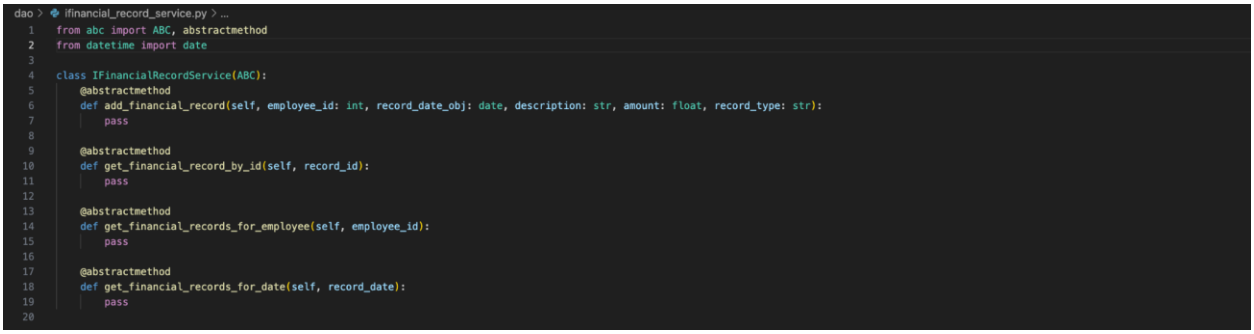
## TaxService (implements ITaxService):

- Methods: CalculateTax, GetTaxById, GetTaxesForEmployee, GetTaxesForYear

```python
from abc import ABC, abstractmethod

class ITaxService(ABC):
    @abstractmethod
    def calculate_tax(self, employee_id: int, tax_year: int, taxable_income: float, tax_rate: float = 0.15):
        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, tax_year):
        pass
```

**FinancialRecordService (implements IFinancialRecordService):**

- Methods: AddFinancialRecord, GetFinancialRecordById, GetFinancialRecordsForEmployee, GetFinancialRecordsForDate



```python
from abc import ABC, abstractmethod
from datetime import date

class IFinancialRecordService(ABC):
    @abstractmethod
    def add_financial_record(self, employee_id: int, record_date_obj: date, description: str, amount: float, record_type: str):
        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, record_date):
        pass
```

**DatabaseContext:**

- A class responsible for handling database connections and interactions.

```python
import configparser
import os

class DBPropertyUtil:
    @staticmethod
    def get_connection_properties(config_file_name="db.properties"):
        """
        Reads database connection properties from a .properties file.
        Expects a section like [mysql] with keys: host, user, password, database, port (optional).
        Returns a dictionary of these properties.
        """
        config = configparser.ConfigParser()

        if not os.path.exists(config_file_name):
            raise FileNotFoundError(f"Configuration file '{config_file_name}' not found.")

        config.read(config_file_name)

        if 'mysql' in config:
            mysql_config = config['mysql']

            required_keys = ['host', 'user', 'password', 'database']
            for key in required_keys:
                if key not in mysql_config:
                    raise ValueError(f"Missing required key '{key}' in [mysql] section of '{config_file_name}'.")

            connection_props = {
                'host': mysql_config['host'],
                'user': mysql_config['user'],
                'password': mysql_config['password'],
                'database': mysql_config['database']
            }

            if 'port' in mysql_config:
                try:
                    connection_props['port'] = int(mysql_config['port'])
                except ValueError:
                    raise ValueError(f"Invalid port number '{mysql_config['port']}' in [mysql] section. Port must be an integer.")

            return connection_props
        else:
            raise ValueError(f"[mysql] section not found in '{config_file_name}'.")
```

**ValidationService:**

- A class for input validation and business rule enforcement.

**ReportGenerator:**

- A class for generating various reports based on payroll, tax, and financial record data.

util > report_generator.py > ...

```python
from datetime import date
from dao.payroll_service import PayrollService
from dao.tax_service import TaxService
from dao.financial_record_service import FinancialRecordService
from dao.employee_service import EmployeeService
from entity.employee import Employee
from entity.payroll import Payroll
from entity.tax import Tax
from entity.financial_record import FinancialRecord
from exception.custom_exceptions import EmployeeNotFoundException, DatabaseConnectionException, PayrollNotFoundException


class ReportGenerator:
    def __init__(self, db_file_name="db.properties"):
        """
        Initializes the ReportGenerator with necessary service instances.
        """
        try:
            self.employee_service = EmployeeService(db_file_name)
            self.payroll_service = PayrollService(db_file_name)
            self.tax_service = TaxService(db_file_name)
            self.financial_record_service = FinancialRecordService(db_file_name)
        except DatabaseConnectionException as e:
            print(f"Failed to initialize ReportGenerator due to DB connection issue: {e}")
            raise

    def generate_employee_payslip(self, employee_id: int, payroll_id: int, output_format: str = "text"):
        """
        Generates a payslip for a specific employee and payroll period.
        output_format can be "text" or "pdf".
        output_path is used if format is "pdf".
        """
        try:
            employee = self.employee_service.get_employee_by_id(employee_id)
            payroll = self.payroll_service.get_payroll_by_id(payroll_id)

            if not employee:
                raise EmployeeNotFoundException(f"Employee with ID {employee_id} not found (unexpectedly not raised by service).")
            if not payroll:
                raise PayrollNotFoundException(f"Payroll with ID {payroll_id} not found (unexpectedly not raised by service).")
            if payroll.get_employee_id() != employee_id:
                raise Exception(f"Data integrity issue: Payroll ID {payroll_id} does not belong to Employee ID {employee_id}.")

            payslip_data = {
                "Employee Name": f"{employee.get_first_name()} {employee.get_last_name()}",
                "Employee ID": employee.get_employee_id(),
                "Position": employee.get_position(),
                "Pay Period Start": payroll.get_pay_period_start_date().isoformat(),
                "Pay Period End": payroll.get_pay_period_end_date().isoformat(),
                "Basic Salary": f"{payroll.get_basic_salary():.2f}",
                "Overtime Pay": f"{payroll.get_overtime_pay():.2f}",
                "Gross Salary": f"{(payroll.get_basic_salary() + payroll.get_overtime_pay()):.2f}",
                "Deductions": f"{payroll.get_deductions():.2f}",
                "Net Salary": f"{payroll.get_net_salary():.2f}",
                "Generation Date": date.today().isoformat()
            }

            if output_format.lower() == "text":
                report_str = f"--- PAYSLIP ---\n"
                for key, value in payslip_data.items():
                    report_str += f"{key}: {value}\n"
                report_str += "--- END OF PAYSLIP ---"
                return report_str
            else:
                return "Unsupported output format."

        except EmployeeNotFoundException as enfe:
            return str(enfe)
        except Exception as e:
            return f"Error generating payslip: {e}"

    def generate_employee_tax_report(self, employee_id: int, tax_year: int, output_format: str = "text"):
        """Generates a tax report for a specific employee and tax year."""
```

util > report_generator.py > ...

```python
class ReportGenerator:
    def generate_employee_tax_report(self, employee_id: int, tax_year: int, output_format: str = "text"):
        try:
            employee = self.employee_service.get_employee_by_id(employee_id)
            if not employee:
                raise EmployeeNotFoundException(f"Employee with ID {employee_id} not found.")

            taxes = self.tax_service.get_taxes_for_employee(employee_id)
            year_specific_taxes = [t for t in taxes if t.get_tax_year() == tax_year]

            if not year_specific_taxes:
                return f"No tax records found for Employee ID {employee_id} for tax year {tax_year}."

            report_title = f"Tax Report for {employee.get_first_name()} {employee.get_last_name()} (ID: {employee_id}) - Year {tax_year}"
            tax_data_list = []
            total_taxable_income = 0
            total_tax_amount = 0

            for tax_record in year_specific_taxes:
                tax_data_list.append({
                    "Tax ID": tax_record.get_tax_id(),
                    "Taxable Income": f"{tax_record.get_taxable_income():.2f}",
                    "Tax Amount": f"{tax_record.get_tax_amount():.2f}"
                })
                total_taxable_income += tax_record.get_taxable_income()
                total_tax_amount += tax_record.get_tax_amount()

            summary = {
                "Total Taxable Income for Year": f"{total_taxable_income:.2f}",
                "Total Tax Paid for Year": f"{total_tax_amount:.2f}"
            }

            if output_format.lower() == "text":
                report_str = f"--- {report_title} ---\n"
                for record_dict in tax_data_list:
                    for key, value in record_dict.items():
                        report_str += f"  {key}: {value}\n"
                    report_str += "  ---\n"
                report_str += "Summary:\n"
                for key, value in summary.items():
                    report_str += f"  {key}: {value}\n"
                report_str += "--- END OF TAX REPORT ---"
                return report_str
            else:
                return "Unsupported output format for tax report."

        except EmployeeNotFoundException as enfe:
            return str(enfe)
        except Exception as e:
            return f"Error generating tax report: {e}"

    def generate_financial_summary_report(self, employee_id: int, start_date: date, end_date: date, output_format: str = "text"):
        """Generates a financial summary report for an employee over a period."""
        try:
            employee = self.employee_service.get_employee_by_id(employee_id)
            if not employee:
                raise EmployeeNotFoundException(f"Employee with ID {employee_id} not found.")

            all_records = self.financial_record_service.get_financial_records_for_employee(employee_id)

            period_records = [
                r for r in all_records
                if r.get_record_date() and start_date <= r.get_record_date() <= end_date
            ]

            if not period_records:
                return f"No financial records found for Employee ID {employee_id} between {start_date} and {end_date}."

            report_title = (f"Financial Summary for {employee.get_first_name()} {employee.get_last_name()} (ID: {employee_id})\n"
                            f"Period: {start_date.isoformat()} to {end_date.isoformat()}")

            records_details = []
            total_income = 0.0
            total_expenses = 0.0

```

# Business Logic Implementation

The PayXpert system incorporates well-defined business logic to ensure accurate payroll processing and financial management. Key business rules applied across modules include:

## Employee Module Logic

- Unique Employee ID validation before adding new employees.
- Age calculation derived from Date of Birth to ensure eligibility for employment.
- Restriction: Employees marked with Termination Date are excluded from active payroll processing.

```
dao > employee_service.py > EmployeeService > _map_row_to_employee
 8  class EmployeeService(IEmployeeService):
66      def add_employee(self, employee: Employee):
77              params = (
78                  employee.get_first_name(), employee.get_last_name(), employee.get_date_of_birth(),
79                  employee.get_gender(), employee.get_email(), employee.get_phone_number(),
80                  employee.get_address(), employee.get_position(), employee.get_joining_date(),
81                  employee.get_termination_date()
82              )
83              cursor.execute(sql, params)
84              connection.commit()
85
86              last_id = cursor.lastrowid
87              if last_id:
88                  employee.set_employee_id(last_id)
89              else:
90                  cursor.execute("SELECT LAST_INSERT_ID()")
91                  last_id = cursor.fetchone()['LAST_INSERT_ID()']
92                  employee.set_employee_id(last_id)
93
94              return employee
95          except mysql.connector.Error as err:
96              if connection: connection.rollback()
97              raise DatabaseConnectionException(f"Database error adding employee: {err}")
98          except Exception as e:
99              if connection: connection.rollback()
100             raise DatabaseConnectionException(f"Unexpected error adding employee: {e}")
101         finally:
102             self._close_cursor_connection(connection, cursor)
103
104     def update_employee(self, employee: Employee):
105         if not isinstance(employee, Employee) or not employee.get_employee_id():
106             raise InvalidInputException("Invalid employee data or missing Employee ID for update.")
107         connection, cursor = None, None
108         try:
109             connection, cursor = self._get_cursor()
110             cursor.execute("SELECT EmployeeID FROM Employee WHERE EmployeeID = %s", (employee.get_employee_id(),))
111             if not cursor.fetchone():
112                 raise EmployeeNotFoundException(f"Employee with ID {employee.get_employee_id()} not found for update.")
113
114             sql = """
115             UPDATE Employee SET
116             FirstName = %s, LastName = %s, DateOfBirth = %s, Gender = %s, Email = %s,
117             PhoneNumber = %s, Address = %s, Position = %s, JoiningDate = %s, TerminationDate = %s
118             WHERE EmployeeID = %s
119             """
120             params = (
121                 employee.get_first_name(), employee.get_last_name(), employee.get_date_of_birth(),
122                 employee.get_gender(), employee.get_email(), employee.get_phone_number(),
123                 employee.get_address(), employee.get_position(), employee.get_joining_date(),
124                 employee.get_termination_date(), employee.get_employee_id()
125             )
126             cursor.execute(sql, params)
127             connection.commit()
128             if cursor.rowcount == 0:
129                 raise EmployeeNotFoundException(f"Employee with ID {employee.get_employee_id()} was not updated (possibly deleted after check).")
130             return employee
131         except EmployeeNotFoundException:
132             if connection: connection.rollback()
133             raise
134         except mysql.connector.Error as err:
135             if connection: connection.rollback()
136             raise DatabaseConnectionException(f"Database error updating employee ID {employee.get_employee_id()}: {err}")
137         except Exception as e:
138             if connection: connection.rollback()
139             raise DatabaseConnectionException(f"Unexpected error updating employee ID {employee.get_employee_id()}: {e}")
140         finally:
141             self._close_cursor_connection(connection, cursor)
142
143     def remove_employee(self, employee_id):
144         if not isinstance(employee_id, int) or employee_id <= 0:
145             raise InvalidInputException("Employee ID must be a positive integer.")
146         connection, cursor = None, None
```

## Payroll Module Logic

- Net Salary Calculation:
  Net Salary = Basic Salary + Overtime Pay - Deductions - Applicable Taxes
- Automatic calculation of Net Salary based on entered salary components.
- Enforcement of unique Payroll ID for each record to prevent duplication.
- Ensures payroll is only processed for active employees

```
dao > payroll_service.py > PayrollService > _map_row_to_payroll
from datetime import date
from dao.ipayroll_service import IPayrollService
from entity.payroll import Payroll
from util.db_conn_util import DBConnUtil
from exception.custom_exceptions import PayrollGenerationException, EmployeeNotFoundException, DatabaseConnectionException, InvalidInputException, PayrollNotFoundException
from dao.employee_service import EmployeeService
import mysql.connector

class PayrollService(IPayrollService):
    def __init__(self, db_file_name="db.properties"):
        self.db_file_name = db_file_name
        self.employee_service = EmployeeService(db_file_name)


    def _get_cursor(self):
        connection = DBConnUtil.get_connection(self.db_file_name)
        if not connection:
            raise DatabaseConnectionException("Failed to get database connection for PayrollService.")
        return connection, connection.cursor(dictionary=True)

    def _close_cursor_connection(self, connection, cursor):
        if cursor:
            cursor.close()

    def _map_row_to_payroll(self, row):
        if row:

            return Payroll(
                payroll_id=row.get("PayrollID"),
                employee_id=row.get("EmployeeID"),
                pay_period_start_date=row.get("PayPeriodStartDate"),
                pay_period_end_date=row.get("PayPeriodEndDate"),
                basic_salary=float(row.get("BasicSalary", 0.0)) if row.get("BasicSalary") is not None else 0.0,
                overtime_pay=float(row.get("OvertimePay", 0.0)) if row.get("OvertimePay") is not None else 0.0,
                deductions=float(row.get("Deductions", 0.0)) if row.get("Deductions") is not None else 0.0,
                net_salary=float(row.get("NetSalary", 0.0)) if row.get("NetSalary") is not None else 0.0
            )
        return None

    def generate_payroll(self, employee_id: int, start_date_obj: date, end_date_obj: date,
                         basic_salary: float, overtime_pay: float = 0.0, deductions: float = 0.0):
        if not all([isinstance(employee_id, int), employee_id > 0,
                    isinstance(start_date_obj, date), isinstance(end_date_obj, date),
                    isinstance(basic_salary, (int, float)) and basic_salary >= 0,
                    isinstance(overtime_pay, (int, float)) and overtime_pay >= 0,
                    isinstance(deductions, (int, float)) and deductions >= 0]):
            raise InvalidInputException("Invalid input for generating payroll. Check types and values.")

        if start_date_obj > end_date_obj:
            raise InvalidInputException("Pay period start date cannot be after end date.")

        connection, cursor = None, None
        try:
            self.employee_service.get_employee_by_id(employee_id)

            net_salary = (basic_salary + overtime_pay) - deductions
            if net_salary < 0:
                raise PayrollGenerationException("Net salary cannot be negative. Adjust salary components or deductions.")

            connection, cursor = self._get_cursor()
            sql = """
            INSERT INTO Payroll
            (EmployeeID, PayPeriodStartDate, PayPeriodEndDate, BasicSalary, OvertimePay, Deductions, NetSalary)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
            """
            params = (
                employee_id, start_date_obj, end_date_obj,
                basic_salary, overtime_pay, deductions, net_salary
            )
            cursor.execute(sql, params)
            connection.commit()

            last_id = cursor.lastrowid
            if not last_id:
                cursor.execute("SELECT LAST_INSERT_ID()")
                last_id = cursor.fetchone()['LAST_INSERT_ID()']
```

```
dao > payroll_service.py > PayrollService > _map_row_to_payroll
 9  class PayrollService(IPayrollService):
40      def generate_payroll(self, employee_id: int, start_date_obj: date, end_date_obj: date,
78              if not last_id:
79                  raise PayrollGenerationException("Failed to retrieve ID for newly generated payroll.")
80
81              return self.get_payroll_by_id(last_id)
82
83          except EmployeeNotFoundException:
84              raise PayrollGenerationException(f"Cannot generate payroll: Employee with ID {employee_id} not found.")
85          except mysql.connector.Error as err:
86              if connection: connection.rollback()
87              raise PayrollGenerationException(f"Database error generating payroll for employee ID {employee_id}: {err}")
88          except Exception as e:
89              if connection: connection.rollback()
90              if isinstance(e, PayrollGenerationException):
91                  raise
92              raise PayrollGenerationException(f"Unexpected error generating payroll for employee ID {employee_id}: {e}")
93          finally:
94              self._close_cursor_connection(connection, cursor)
95
96      def get_payroll_by_id(self, payroll_id):
97          if not isinstance(payroll_id, int) or payroll_id <= 0:
98              raise InvalidInputException("Payroll ID must be a positive integer.")
99          connection, cursor = None, None
100         try:
101             connection, cursor = self._get_cursor()
102             cursor.execute("SELECT * FROM Payroll WHERE PayrollID = %s", (payroll_id,))
103             row = cursor.fetchone()
104             if not row:
105                 raise PayrollNotFoundException(f"Payroll record with ID {payroll_id} not found.")
106             return self._map_row_to_payroll(row)
107         except PayrollNotFoundException:
108             raise
109         except mysql.connector.Error as err:
110             raise DatabaseConnectionException(f"Database error fetching payroll by ID {payroll_id}: {err}")
111         except Exception as e:
112             raise DatabaseConnectionException(f"Unexpected error fetching payroll by ID {payroll_id}: {e}")
113         finally:
114             self._close_cursor_connection(connection, cursor)
115
116     def get_payrolls_for_employee(self, employee_id):
117         if not isinstance(employee_id, int) or employee_id <= 0:
118             raise InvalidInputException("Employee ID must be a positive integer.")
119
120         connection, cursor = None, None
121         try:
122             connection, cursor = self._get_cursor()
123             cursor.execute("SELECT * FROM Payroll WHERE EmployeeID = %s ORDER BY PayPeriodStartDate DESC", (employee_id,))
124             rows = cursor.fetchall()
125             return [self._map_row_to_payroll(row) for row in rows]
126         except mysql.connector.Error as err:
127             raise DatabaseConnectionException(f"Database error fetching payrolls for employee ID {employee_id}: {err}")
128         except Exception as e:
129             raise DatabaseConnectionException(f"Unexpected error fetching payrolls for employee ID {employee_id}: {e}")
130         finally:
131             self._close_cursor_connection(connection, cursor)
132
133     def get_payrolls_for_period(self, start_date_obj: date, end_date_obj: date):
134         if not (isinstance(start_date_obj, date) and isinstance(end_date_obj, date)):
135             raise InvalidInputException("Start date and end date must be valid date objects.")
136         if start_date_obj > end_date_obj:
137             raise InvalidInputException("Period start date cannot be after end date.")
138
139         connection, cursor = None, None
140         try:
141             connection, cursor = self._get_cursor()
142             sql = """
143             SELECT * FROM Payroll
144             WHERE PayPeriodStartDate <= %s AND PayPeriodEndDate >= %s
145             ORDER BY PayPeriodStartDate
146             """
147
148             cursor.execute(sql, (end_date_obj, start_date_obj))
149             rows = cursor.fetchall()
150             return [self._map_row_to_payroll(row) for row in rows]
151         except mysql.connector.Error as err:
152             raise DatabaseConnectionException(f"Database error fetching payrolls for period: {err}")
```

## Tax Module Logic

- Tax deductions based on defined salary brackets.
- Each tax record linked to a valid Employee ID.
- Prevents duplicate tax entries for the same employee and tax year.
- Integrates calculated tax amounts into the Net Salary deduction process.

dao > tax_service.py > ...

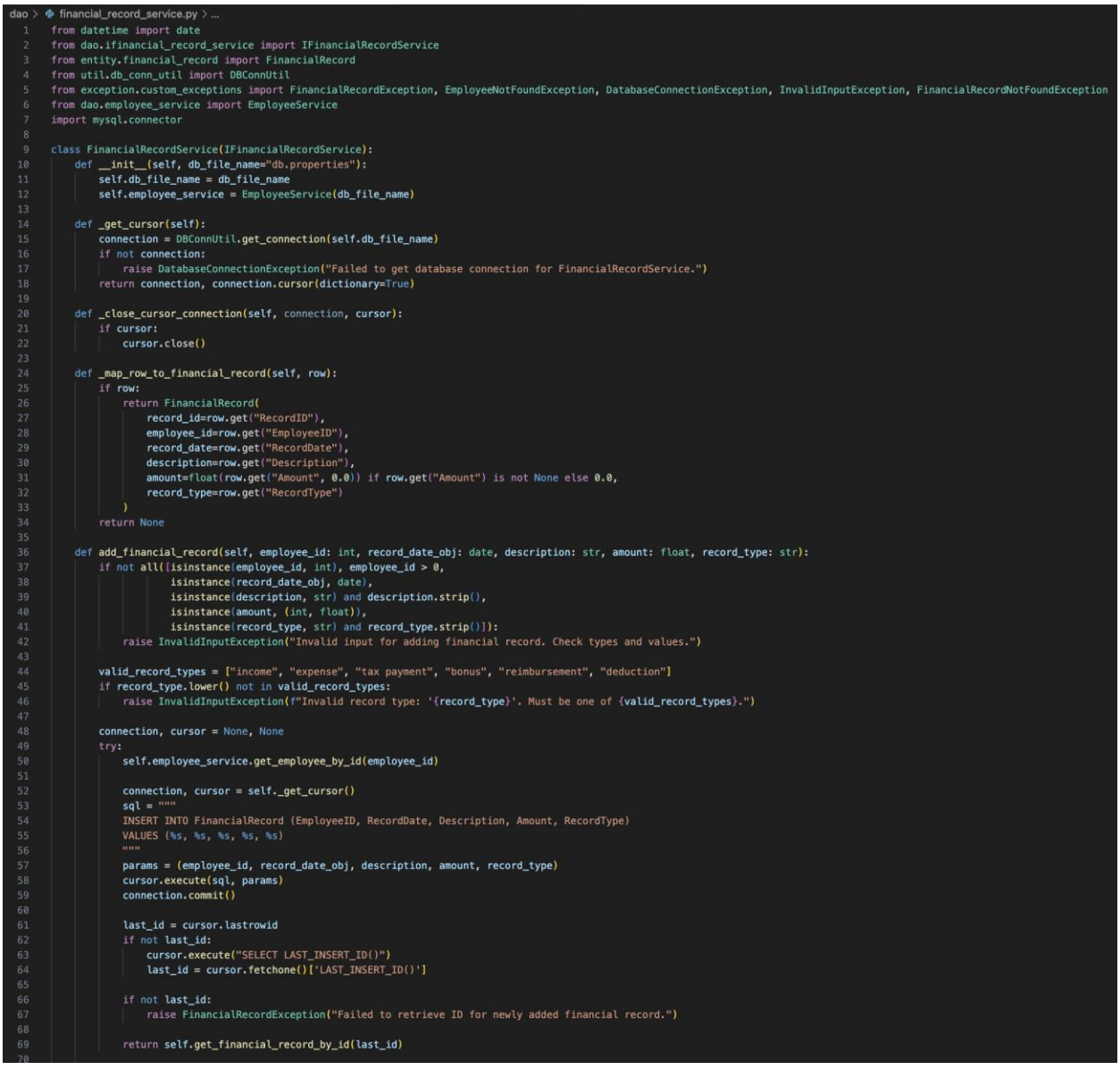
```python
from datetime import date
from dao.itax_service import ITaxService
from entity.tax import Tax
from util.db_conn_util import DBConnUtil
from exception.custom_exceptions import TaxCalculationException, EmployeeNotFoundException, DatabaseConnectionException, InvalidInputException, TaxRecordNotFoundException
from dao.employee_service import EmployeeService
import mysql.connector

class TaxService(ITaxService):
    def __init__(self, db_file_name="db.properties"):
        self.db_file_name = db_file_name
        self.employee_service = EmployeeService(db_file_name)

    def _get_cursor(self):
        connection = DBConnUtil.get_connection(self.db_file_name)
        if not connection:
            raise DatabaseConnectionException("Failed to get database connection for TaxService.")
        return connection, connection.cursor(dictionary=True)

    def _close_cursor_connection(self, connection, cursor):
        if cursor:
            cursor.close()

    def _map_row_to_tax(self, row):
        if row:
            return Tax(
                tax_id=row.get("TaxID"),
                employee_id=row.get("EmployeeID"),
                tax_year=int(row.get("TaxYear")) if row.get("TaxYear") is not None else None,
                taxable_income=float(row.get("TaxableIncome", 0.0)) if row.get("TaxableIncome") is not None else 0.0,
                tax_amount=float(row.get("TaxAmount", 0.0)) if row.get("TaxAmount") is not None else 0.0
            )
        return None

    def calculate_tax(self, employee_id: int, tax_year: int, taxable_income: float, tax_rate: float = 0.15):
        if not all([isinstance(employee_id, int), employee_id > 0,
                    isinstance(tax_year, int), tax_year > 1900,
                    isinstance(taxable_income, (int, float)) and taxable_income >= 0,
                    isinstance(tax_rate, float) and 0 <= tax_rate <= 1]):
            raise InvalidInputException("Invalid input for calculating tax. Check types and values.")

        connection, cursor = None, None
        try:

            self.employee_service.get_employee_by_id(employee_id)

            tax_amount = taxable_income * tax_rate
            connection, cursor = self._get_cursor()

            cursor.execute(
                "SELECT TaxID FROM Tax WHERE EmployeeID = %s AND TaxYear = %s",
                (employee_id, tax_year)
            )
            existing_tax_record_row = cursor.fetchone()
            tax_id_to_return = None

            if existing_tax_record_row:
                tax_id_to_return = existing_tax_record_row['TaxID']
                sql_update = """
                UPDATE Tax SET TaxableIncome = %s, TaxAmount = %s
                WHERE TaxID = %s
                """
                params_update = (taxable_income, tax_amount, tax_id_to_return)
                cursor.execute(sql_update, params_update)
            else:
                sql_insert = """
                INSERT INTO Tax (EmployeeID, TaxYear, TaxableIncome, TaxAmount)
                VALUES (%s, %s, %s, %s)
                """
                params_insert = (employee_id, tax_year, taxable_income, tax_amount)
                cursor.execute(sql_insert, params_insert)
                tax_id_to_return = cursor.lastrowid
                if not tax_id_to_return:
                    cursor.execute("SELECT LAST_INSERT_ID()")
                    tax_id_to_return = cursor.fetchone()['LAST_INSERT_ID()']
```

dao > tax_service.py > ...

```
 9  class TaxService(ITaxService):
35      def calculate_tax(self, employee_id: int, tax_year: int, taxable_income: float, tax_rate: float = 0.15):
72                  tax_id_to_return = cursor.lastrowid
73                  if not tax_id_to_return:
74                      cursor.execute("SELECT LAST_INSERT_ID()")
75                      tax_id_to_return = cursor.fetchone()['LAST_INSERT_ID()']
76
77              connection.commit()
78              if not tax_id_to_return:
79                  raise TaxCalculationException(f"Failed to get TaxID after upsert operation for EmpID {employee_id}, Year {tax_year}.")
80
81              return self.get_tax_by_id(tax_id_to_return)
82
83          except EmployeeNotFoundException:
84              raise TaxCalculationException(f"Cannot calculate tax: Employee with ID {employee_id} not found.")
85          except mysql.connector.Error as err:
86              if connection: connection.rollback()
87              raise TaxCalculationException(f"Database error calculating tax for EmpID {employee_id}, Year {tax_year}: {err}")
88          except Exception as e:
89              if connection: connection.rollback()
90              if isinstance(e, TaxCalculationException): raise
91              raise TaxCalculationException(f"Unexpected error calculating tax for EmpID {employee_id}, Year {tax_year}: {e}")
92          finally:
93              self._close_cursor_connection(connection, cursor)
94
95      def get_tax_by_id(self, tax_id):
96          if not isinstance(tax_id, int) or tax_id <= 0:
97              raise InvalidInputException("Tax ID must be a positive integer.")
98          connection, cursor = None, None
99          try:
100             connection, cursor = self._get_cursor()
101             cursor.execute("SELECT * FROM Tax WHERE TaxID = %s", (tax_id,))
102             row = cursor.fetchone()
103             if not row:
104                 raise TaxRecordNotFoundException(f"Tax record with ID {tax_id} not found.")
105             return self._map_row_to_tax(row)
106         except TaxRecordNotFoundException:
107             raise
108         except mysql.connector.Error as err:
109             raise DatabaseConnectionException(f"Database error fetching tax by ID {tax_id}: {err}")
110         except Exception as e:
111             raise DatabaseConnectionException(f"Unexpected error fetching tax by ID {tax_id}: {e}")
112         finally:
113             self._close_cursor_connection(connection, cursor)
114
115     def get_taxes_for_employee(self, employee_id):
116         if not isinstance(employee_id, int) or employee_id <= 0:
117             raise InvalidInputException("Employee ID must be a positive integer.")
118         connection, cursor = None, None
119         try:
120             connection, cursor = self._get_cursor()
121             cursor.execute("SELECT * FROM Tax WHERE EmployeeID = %s ORDER BY TaxYear DESC", (employee_id,))
122             rows = cursor.fetchall()
123             return [self._map_row_to_tax(row) for row in rows]
124         except mysql.connector.Error as err:
125             raise DatabaseConnectionException(f"Database error fetching taxes for employee ID {employee_id}: {err}")
126         except Exception as e:
127             raise DatabaseConnectionException(f"Unexpected error fetching taxes for employee ID {employee_id}: {e}")
128         finally:
129             self._close_cursor_connection(connection, cursor)
130
131     def get_taxes_for_year(self, tax_year):
132         if not isinstance(tax_year, int) or tax_year <= 1900:
133             raise InvalidInputException("Tax year must be a valid year (e.g., greater than 1900).")
134         connection, cursor = None, None
135         try:
136             connection, cursor = self._get_cursor()
137             cursor.execute("SELECT * FROM Tax WHERE TaxYear = %s ORDER BY EmployeeID", (tax_year,))
138             rows = cursor.fetchall()
139             return [self._map_row_to_tax(row) for row in rows]
140         except mysql.connector.Error as err:
141             raise DatabaseConnectionException(f"Database error fetching taxes for year {tax_year}: {err}")
142         except Exception as e:
143             raise DatabaseConnectionException(f"Unexpected error fetching taxes for year {tax_year}: {e}")
144         finally:
145             self._close_cursor_connection(connection, cursor)
146
```

## Financial Report Module Logic

- Aggregates total salary expenses, tax deductions, and net expenditures.
- Provides summarized reports for management decision-making.
- Ensures financial reports only include active and valid payroll records.



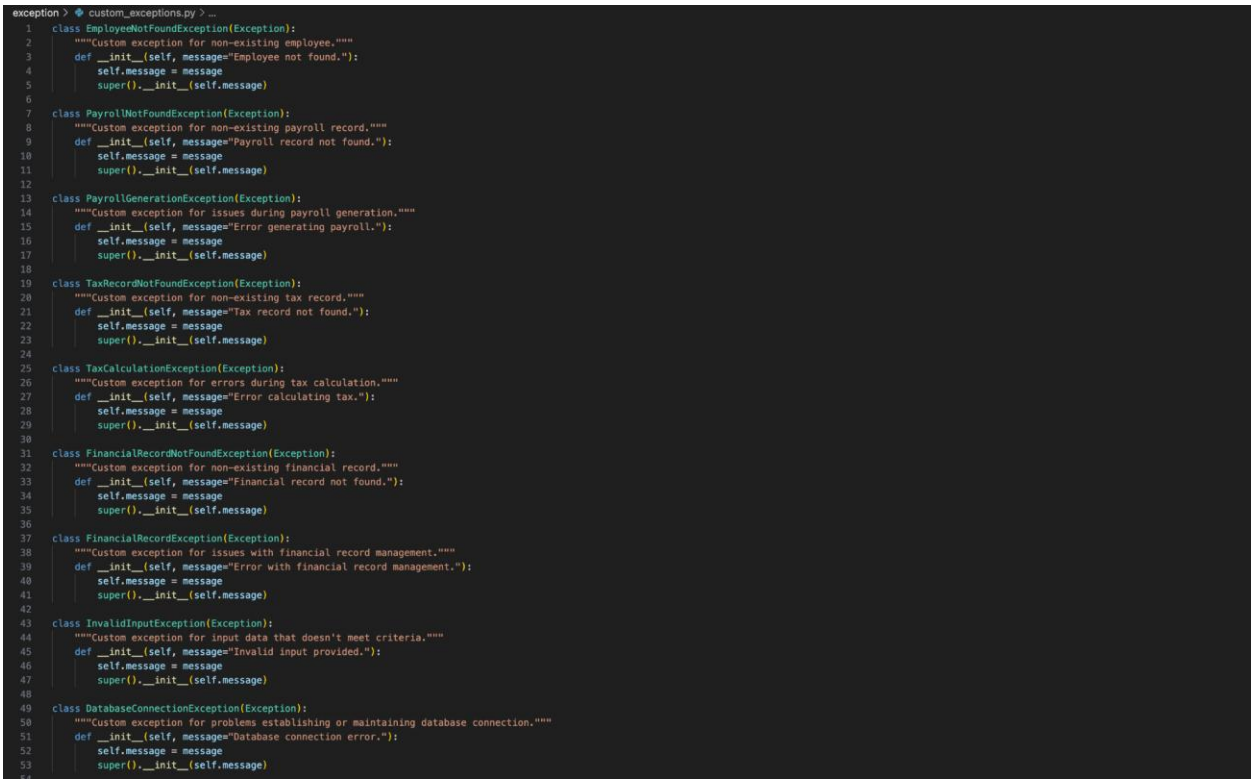
```
dao > financial_record_service.py > ...
from datetime import date
from dao.ifinancial_record_service import IFinancialRecordService
from entity.financial_record import FinancialRecord
from util.db_conn_util import DBConnUtil
from exception.custom_exceptions import FinancialRecordException, EmployeeNotFoundException, DatabaseConnectionException, InvalidInputException, FinancialRecordNotFoundException
from dao.employee_service import EmployeeService
import mysql.connector

class FinancialRecordService(IFinancialRecordService):
    def __init__(self, db_file_name="db.properties"):
        self.db_file_name = db_file_name
        self.employee_service = EmployeeService(db_file_name)

    def _get_cursor(self):
        connection = DBConnUtil.get_connection(self.db_file_name)
        if not connection:
            raise DatabaseConnectionException("Failed to get database connection for FinancialRecordService.")
        return connection, connection.cursor(dictionary=True)

    def _close_cursor_connection(self, connection, cursor):
        if cursor:
            cursor.close()

    def _map_row_to_financial_record(self, row):
        if row:
            return FinancialRecord(
                record_id=row.get("RecordID"),
                employee_id=row.get("EmployeeID"),
                record_date=row.get("RecordDate"),
                description=row.get("Description"),
                amount=float(row.get("Amount", 0.0)) if row.get("Amount") is not None else 0.0,
                record_type=row.get("RecordType")
            )
        return None

    def add_financial_record(self, employee_id: int, record_date_obj: date, description: str, amount: float, record_type: str):
        if not all([isinstance(employee_id, int), employee_id > 0,
                    isinstance(record_date_obj, date),
                    isinstance(description, str) and description.strip(),
                    isinstance(amount, (int, float)),
                    isinstance(record_type, str) and record_type.strip()]):
            raise InvalidInputException("Invalid input for adding financial record. Check types and values.")

        valid_record_types = ["income", "expense", "tax payment", "bonus", "reimbursement", "deduction"]
        if record_type.lower() not in valid_record_types:
            raise InvalidInputException(f"Invalid record type: '{record_type}'. Must be one of {valid_record_types}.")

        connection, cursor = None, None
        try:
            self.employee_service.get_employee_by_id(employee_id)

            connection, cursor = self._get_cursor()
            sql = """
            INSERT INTO FinancialRecord (EmployeeID, RecordDate, Description, Amount, RecordType)
            VALUES (%s, %s, %s, %s, %s)
            """
            params = (employee_id, record_date_obj, description, amount, record_type)
            cursor.execute(sql, params)
            connection.commit()

            last_id = cursor.lastrowid
            if not last_id:
                cursor.execute("SELECT LAST_INSERT_ID()")
                last_id = cursor.fetchone()['LAST_INSERT_ID()']

            if not last_id:
                raise FinancialRecordException("Failed to retrieve ID for newly added financial record.")

            return self.get_financial_record_by_id(last_id)

```

dao > financial_record_service.py > ...

```
9   class FinancialRecordService(IFinancialRecordService):
36      def add_financial_record(self, employee_id: int, record_date_obj: date, description: str, amount: float, record_type: str):
71          except EmployeeNotFoundException:
72              raise FinancialRecordException(f"Cannot add financial record: Employee with ID {employee_id} not found.")
73          except mysql.connector.Error as err:
74              if connection: connection.rollback()
75              raise FinancialRecordException(f"Database error adding financial record for EmpID {employee_id}: {err}")
76          except Exception as e:
77              if connection: connection.rollback()
78              if isinstance(e, FinancialRecordException) or isinstance(e, InvalidInputException): raise
79              raise FinancialRecordException(f"Unexpected error adding financial record for EmpID {employee_id}: {e}")
80          finally:
81              self._close_cursor_connection(connection, cursor)
82
83      def get_financial_record_by_id(self, record_id):
84          if not isinstance(record_id, int) or record_id <= 0:
85              raise InvalidInputException("Record ID must be a positive integer.")
86          connection, cursor = None, None
87          try:
88              connection, cursor = self._get_cursor()
89              cursor.execute("SELECT * FROM FinancialRecord WHERE RecordID = %s", (record_id,))
90              row = cursor.fetchone()
91              if not row:
92                  raise FinancialRecordNotFoundException(f"Financial record with ID {record_id} not found.")
93              return self._map_row_to_financial_record(row)
94          except FinancialRecordNotFoundException:
95              raise
96          except mysql.connector.Error as err:
97              raise DatabaseConnectionException(f"Database error fetching financial record by ID {record_id}: {err}")
98          except Exception as e:
99              raise DatabaseConnectionException(f"Unexpected error fetching financial record by ID {record_id}: {e}")
100         finally:
101             self._close_cursor_connection(connection, cursor)
102
103     def get_financial_records_for_employee(self, employee_id):
104         if not isinstance(employee_id, int) or employee_id <= 0:
105             raise InvalidInputException("Employee ID must be a positive integer.")
106         connection, cursor = None, None
107         try:
108             connection, cursor = self._get_cursor()
109             cursor.execute("SELECT * FROM FinancialRecord WHERE EmployeeID = %s ORDER BY RecordDate DESC", (employee_id,))
110             rows = cursor.fetchall()
111             return [self._map_row_to_financial_record(row) for row in rows]
112         except mysql.connector.Error as err:
113             raise DatabaseConnectionException(f"Database error fetching financial records for employee ID {employee_id}: {err}")
114         except Exception as e:
115             raise DatabaseConnectionException(f"Unexpected error fetching financial records for employee ID {employee_id}: {e}")
116         finally:
117             self._close_cursor_connection(connection, cursor)
118
119     def get_financial_records_for_date(self, record_date_obj: date):
120         if not isinstance(record_date_obj, date):
121             raise InvalidInputException("Record date must be a valid date object.")
122         connection, cursor = None, None
123         try:
124             connection, cursor = self._get_cursor()
125             cursor.execute("SELECT * FROM FinancialRecord WHERE RecordDate = %s ORDER BY EmployeeID", (record_date_obj,))
126             rows = cursor.fetchall()
127             return [self._map_row_to_financial_record(row) for row in rows]
128         except mysql.connector.Error as err:
129             raise DatabaseConnectionException(f"Database error fetching financial records for date {record_date_obj}: {err}")
130         except Exception as e:
131             raise DatabaseConnectionException(f"Unexpected error fetching financial records for date {record_date_obj}: {e}")
132         finally:
133             self._close_cursor_connection(connection, cursor)
```

## Data Integrity and Exception Handling

- Prevents insertion of invalid or incomplete data through input validation.
- Catches database connectivity failures and handles gracefully.
- Provides user-friendly error messages for invalid operations.
- Modular code structure ensures separation of logic, data handling, and presentation.



```python
class EmployeeNotFoundException(Exception):
    """Custom exception for non-existing employee."""
    def __init__(self, message="Employee not found."):
        self.message = message
        super().__init__(self.message)

class PayrollNotFoundException(Exception):
    """Custom exception for non-existing payroll record."""
    def __init__(self, message="Payroll record not found."):
        self.message = message
        super().__init__(self.message)

class PayrollGenerationException(Exception):
    """Custom exception for issues during payroll generation."""
    def __init__(self, message="Error generating payroll."):
        self.message = message
        super().__init__(self.message)

class TaxRecordNotFoundException(Exception):
    """Custom exception for non-existing tax record."""
    def __init__(self, message="Tax record not found."):
        self.message = message
        super().__init__(self.message)

class TaxCalculationException(Exception):
    """Custom exception for errors during tax calculation."""
    def __init__(self, message="Error calculating tax."):
        self.message = message
        super().__init__(self.message)

class FinancialRecordNotFoundException(Exception):
    """Custom exception for non-existing financial record."""
    def __init__(self, message="Financial record not found."):
        self.message = message
        super().__init__(self.message)

class FinancialRecordException(Exception):
    """Custom exception for issues with financial record management."""
    def __init__(self, message="Error with financial record management."):
        self.message = message
        super().__init__(self.message)

class InvalidInputException(Exception):
    """Custom exception for input data that doesn't meet criteria."""
    def __init__(self, message="Invalid input provided."):
        self.message = message
        super().__init__(self.message)

class DatabaseConnectionException(Exception):
    """Custom exception for problems establishing or maintaining database connection."""
    def __init__(self, message="Database connection error."):
        self.message = message
        super().__init__(self.message)
```

## Unit Testing

Unit testing ensures the correctness, reliability, and robustness of the PayXpert Payroll Management System. The following test scenarios outline essential test cases applied during development:

**Test Case: CalculateGrossSalaryForEmployee**

**Objective:** Verify the system correctly calculates an employee's gross salary.

**Test Steps:**

- Provide sample inputs for basic salary and overtime pay.
- Validate that the gross salary equals the sum of basic salary and overtime pay.
  **Expected Result:** Gross Salary = Basic Salary + Overtime Pay

```python
def test_get_payrolls_for_employee(self):
    print(f"\nRunning test_get_payrolls_for_employee for employee ID: {self.test_employee_id}...")
    p1 = self.payroll_service.generate_payroll(self.test_employee_id, date(2024,4,1), date(2024,4,30), 5100,100,50)
    p2 = self.payroll_service.generate_payroll(self.test_employee_id, date(2024,5,1), date(2024,5,31), 5200,150,60)


    payrolls = self.payroll_service.get_payrolls_for_employee(self.test_employee_id)
    self.assertTrue(len(payrolls) >= 2)

    payroll_ids = [p.get_payroll_id() for p in payrolls]
    self.assertIn(p1.get_payroll_id(), payroll_ids)
    self.assertIn(p2.get_payroll_id(), payroll_ids)
    print("test_get_payrolls_for_employee PASSED.")
```

**Test Case: CalculateNetSalaryAfterDeductions**

**Objective:** Ensure the system accurately calculates net salary after applying deductions (taxes, insurance, etc.).

**Test Steps:**

- Provide inputs for basic salary, overtime pay, and applicable deductions.
- Validate that net salary is correctly computed.
  **Expected Result:**
  Net Salary = Basic Salary + Overtime Pay - Deductions - Tax

```
def test_get_taxes_for_employee(self):
    print(f"\nRunning test_get_taxes_for_employee for EID: {self.test_employee_id}...")
    tax1 = self.tax_service.calculate_tax(self.test_employee_id, self.current_year -1, 60000)
    tax2 = self.tax_service.calculate_tax(self.test_employee_id, self.current_year, 65000)

    taxes = self.tax_service.get_taxes_for_employee(self.test_employee_id)
    self.assertTrue(len(taxes) >= 2)
    tax_ids_retrieved = [t.get_tax_id() for t in taxes]
    self.assertIn(tax1.get_tax_id(), tax_ids_retrieved)
    self.assertIn(tax2.get_tax_id(), tax_ids_retrieved)
    print("test_get_taxes_for_employee PASSED.")
```

**Test Case: VerifyTaxCalculationForHighIncomeEmployee**

**Objective:** Test tax calculation logic for high-income employees to ensure proper deduction according to tax brackets.

**Test Steps:**

- Simulate a high-income employee with salary exceeding the tax threshold.
- Validate the tax deduction is accurate as per defined tax rates.
  **Expected Result:** Tax calculated based on appropriate tax bracket rules.

```
def test_get_taxes_for_employee(self):
    print(f"\nRunning test_get_taxes_for_employee for EID: {self.test_employee_id}...")
    tax1 = self.tax_service.calculate_tax(self.test_employee_id, self.current_year -1, 60000)
    tax2 = self.tax_service.calculate_tax(self.test_employee_id, self.current_year, 65000)

    taxes = self.tax_service.get_taxes_for_employee(self.test_employee_id)
    self.assertTrue(len(taxes) >= 2)
    tax_ids_retrieved = [t.get_tax_id() for t in taxes]
    self.assertIn(tax1.get_tax_id(), tax_ids_retrieved)
    self.assertIn(tax2.get_tax_id(), tax_ids_retrieved)
    print("test_get_taxes_for_employee PASSED.")
```

**Test Case: ProcessPayrollForMultipleEmployees**

**Objective:** Test payroll processing for multiple employees to ensure system handles batch operations.

**Test Steps:**

- Provide payroll inputs for several employees.
- Run payroll processing module.
- Validate correct payroll records generated for each employee.
  **Expected Result:** Payroll processed accurately for all valid employees.

```
def test_get_payrolls_for_employee(self):
    print(f"\nRunning test_get_payrolls_for_employee for employee ID: {self.test_employee_id}...")
    p1 = self.payroll_service.generate_payroll(self.test_employee_id, date(2024,4,1), date(2024,4,30), 5100,100,50)
    p2 = self.payroll_service.generate_payroll(self.test_employee_id, date(2024,5,1), date(2024,5,31), 5200,150,60)


    payrolls = self.payroll_service.get_payrolls_for_employee(self.test_employee_id)
    self.assertTrue(len(payrolls) >= 2)

    payroll_ids = [p.get_payroll_id() for p in payrolls]
    self.assertIn(p1.get_payroll_id(), payroll_ids)
    self.assertIn(p2.get_payroll_id(), payroll_ids)
    print("test_get_payrolls_for_employee PASSED.")
```

**Test Case: VerifyErrorHandlingForInvalidEmployeeData**
**Objective:** Ensure system handles invalid employee data gracefully with proper error messages.
**Test Steps:**

- Enter incomplete or invalid employee details (e.g., missing name, invalid date).
- Attempt to add employee record.
  Expected Result: System rejects invalid input with clear error messages, preventing data corruption.

```python
def test_add_employee_invalid_input_type(self):
    print("\nRunning test_add_employee_invalid_input_type (Unit Test)...")
    with self.assertRaises(InvalidInputException):
        self.employee_service.add_employee("not an employee object")
    print("test_add_employee_invalid_input_type (Unit Test) PASSED.")
```

# Output Screenshots

- Employee Add/Update/View with age calculation.

```
iamesaq@Esaqs-MacBook-Pro Project-feature-initial-dev-and-testing-setup % python3 -m main.main_module
PayXpert System Initialized Successfully.

====== PayXpert Main Menu ======
1. Employee Management
2. Payroll Management
3. Tax Management
4. Financial Record Management
5. Reporting
0. Exit
Enter your choice: 1

--- Employee Management ---
1. Add New Employee
2. Get Employee by ID
3. Get All Employees
4. Update Employee
5. Remove Employee
0. Back to Main Menu
Enter choice: 1

--- Add New Employee ---
First Name: Esaq
Last Name: A
Date of Birth (YYYY-MM-DD): 2003-03-09
Gender (Male, Female, Other): Male
Email: esaq@gmail.com
Phone Number: 8667638590
Address: Chennai
Position: Developer
Joining Date (YYYY-MM-DD): 2025-07-01
Termination Date (optional, leave blank if N/A) (YYYY-MM-DD):

Employee added successfully with ID: 5
Employee ID: 5
Name: Esaq A
DOB: 2003-03-09
Gender: male
Email: esaq@gmail.com
Phone: 8667638590
Address: Chennai
Position: Developer
Joining Date: 2025-07-01
Termination Date: N/A
--- Employee Management ---
1. Add New Employee
2. Get Employee by ID
3. Get All Employees
4. Update Employee
5. Remove Employee
0. Back to Main Menu
Enter choice: 2

--- Get Employee by ID ---
Enter Employee ID: 2
Employee ID: 2
Name: Bob Johnson
DOB: 1985-11-20
Gender: Male
Email: bob.johnson@example.com
Phone: 555-0102
Address: 456 Oak Street
Position: Project Manager
Joining Date: 2018-03-15
Termination Date: N/A
Age: 39
```

```
--- Employee Management ---
1. Add New Employee
2. Get Employee by ID
3. Get All Employees
4. Update Employee
5. Remove Employee
0. Back to Main Menu
Enter choice: 3

--- All Employees ---
Employee ID: 1
Name: Alice Smith
DOB: 1990-05-15
Gender: Female
Email: alice.smith@example.com
Phone: 555-0101
Address: 123 Willow Lane
Position: Software Engineer
Joining Date: 2020-08-01
Termination Date: N/A
Age: 35
--------------------
Employee ID: 2
Name: Bob Johnson
DOB: 1985-11-20
Gender: Male
Email: bob.johnson@example.com
Phone: 555-0102
Address: 456 Oak Street
Position: Project Manager
Joining Date: 2018-03-15
Termination Date: N/A
Age: 39
--------------------
Employee ID: 3
Name: Carol Williams
DOB: 1995-02-10
Gender: Female
Email: carol.williams@example.com
Phone: 555-0103
Address: 789 Pine Avenue
Position: Data Analyst
Joining Date: 2022-01-10
Termination Date: N/A
Age: 30
--------------------
Employee ID: 5
Name: Esaq A
DOB: 2003-03-09
Gender: male
Email: esaq@gmail.com
Phone: 8667638590
Address: Chennai
Position: Developer
Joining Date: 2025-07-01
Termination Date: N/A
Age: 22
--- Employee Management ---
1. Add New Employee
2. Get Employee by ID
3. Get All Employees
4. Update Employee
5. Remove Employee
0. Back to Main Menu
Enter choice: 4

--- Update Employee ---
Enter Employee ID to update: 5
Existing data:
Employee ID: 5
Name: Esaq A
DOB: 2003-03-09
Gender: male
Email: esaq@gmail.com
Phone: 8667638590
Address: Chennai
Position: Developer
Joining Date: 2025-07-01
Termination Date: N/A
Enter new data (leave blank to keep current value):
First Name [Esaq]:
Last Name [A]:
Date of Birth [2003-03-09] (YYYY-MM-DD):
Gender [male]:
Email [esaq@gmail.com]: mynameisesaq@gmail.com
Phone Number [8667638590]: 8667637590
Address [Chennai]:
Position [Developer]:
Joining Date [2025-07-01] (YYYY-MM-DD):
Termination Date [N/A] (YYYY-MM-DD, or empty to clear):

Employee updated successfully:
Employee ID: 5
Name: Esaq A
DOB: 2003-03-09
Gender: male
Email: mynameisesaq@gmail.com
Phone: 8667637590
Address: Chennai
Position: Developer
Joining Date: 2025-07-01
Termination Date: N/A
```

```
--- Employee Management ---
1. Add New Employee
2. Get Employee by ID
3. Get All Employees
4. Update Employee
5. Remove Employee
0. Back to Main Menu
Enter choice: 5

--- Remove Employee ---
Enter Employee ID to remove: 5
Are you sure you want to remove Esaq A (ID: 5)? (yes/no): yes
Employee ID 5 removed successfully.

--- Employee Management ---
1. Add New Employee
2. Get Employee by ID
3. Get All Employees
4. Update Employee
5. Remove Employee
0. Back to Main Menu
Enter choice: 2

--- Get Employee by ID ---
Enter Employee ID: 5
Error: Employee with ID 5 not found.
```

- Payroll Add/View with total record count.

```
--- Payroll Management ---
1. Generate Payroll for Employee
2. Get Payroll by ID
3. Get Payrolls for Employee
4. Get Payrolls for Period
0. Back to Main Menu
Enter choice: 1

--- Generate Payroll ---
Enter Employee ID: 3
Pay Period Start Date (YYYY-MM-DD): 2025-07-01
Pay Period End Date (YYYY-MM-DD): 2025-07-31
Basic Salary: 25000
Overtime Pay (optional, default 0): 0
Deductions (optional, default 0): 0

Payroll generated successfully:
Payroll ID: 5
Employee ID: 3
Pay Period: 2025-07-01 to 2025-07-31
Basic Salary: 25000.0
Overtime Pay: 0.0
Deductions: 0.0
Net Salary: 25000.0
```

```
--- Payroll Management ---
1. Generate Payroll for Employee
2. Get Payroll by ID
3. Get Payrolls for Employee
4. Get Payrolls for Period
0. Back to Main Menu
Enter choice: 2

--- Get Payroll by ID ---
Enter Payroll ID: 2
Payroll ID: 2
Employee ID: 1
Pay Period: 2024-04-01 to 2024-04-30
Basic Salary: 6000.0
Overtime Pay: 250.0
Deductions: 700.0
Net Salary: 5550.0
```

```
--- Payroll Management ---
1. Generate Payroll for Employee
2. Get Payroll by ID
3. Get Payrolls for Employee
4. Get Payrolls for Period
0. Back to Main Menu
Enter choice: 3

--- Get Payrolls for Employee ---
Enter Employee ID: 3
Payroll ID: 5
Employee ID: 3
Pay Period: 2025-07-01 to 2025-07-31
Basic Salary: 25000.0
Overtime Pay: 0.0
Deductions: 0.0
Net Salary: 25000.0
--------------

--- Payroll Management ---
1. Generate Payroll for Employee
2. Get Payroll by ID
3. Get Payrolls for Employee
4. Get Payrolls for Period
0. Back to Main Menu
Enter choice: 4

--- Get Payrolls for Period ---
Search Period Start Date (YYYY-MM-DD): 2024-03-01
Search Period End Date (YYYY-MM-DD): 2024-03-31
Payroll ID: 4
Employee ID: 1
Pay Period: 2020-09-09 to 2025-07-01
Basic Salary: 10000.0
Overtime Pay: 0.0
Deductions: 0.0
Net Salary: 10000.0
--------------
Payroll ID: 1
Employee ID: 1
Pay Period: 2024-03-01 to 2024-03-31
Basic Salary: 6000.0
Overtime Pay: 500.0
Deductions: 700.0
Net Salary: 5800.0
--------------
Payroll ID: 3
Employee ID: 2
Pay Period: 2024-03-01 to 2024-03-31
Basic Salary: 7500.0
Overtime Pay: 0.0
Deductions: 1000.0
Net Salary: 6500.0
--------------
```

- Tax data entry and deduction display.

```
--- Tax Management ---
1. Calculate/Record Tax for Employee
2. Get Tax Record by ID
3. Get Taxes for Employee
4. Get Taxes for Year
0. Back to Main Menu
Enter choice: 1

--- Calculate/Record Tax ---
Enter Employee ID: 3
Tax Year (e.g., 2023): 2024
Taxable Income for the year: 250000
Tax Rate (e.g., 0.15 for 15%, default 0.15): 0.15

Tax calculated and recorded successfully:
Tax ID: 3
Employee ID: 3
Tax Year: 2024
Taxable Income: 250000.0
Tax Amount: 37500.0
```

```
--- Tax Management ---
1. Calculate/Record Tax for Employee
2. Get Tax Record by ID
3. Get Taxes for Employee
4. Get Taxes for Year
0. Back to Main Menu
Enter choice: 2

--- Get Tax Record by ID ---
Enter Tax ID: 1
Tax ID: 1
Employee ID: 1
Tax Year: 2023
Taxable Income: 72000.0
Tax Amount: 10800.0
```

```
--- Tax Management ---
1. Calculate/Record Tax for Employee
2. Get Tax Record by ID
3. Get Taxes for Employee
4. Get Taxes for Year
0. Back to Main Menu
Enter choice: 3

--- Get Taxes for Employee ---
Enter Employee ID: 1
Tax ID: 1
Employee ID: 1
Tax Year: 2023
Taxable Income: 72000.0
Tax Amount: 10800.0
---------------

--- Tax Management ---
1. Calculate/Record Tax for Employee
2. Get Tax Record by ID
3. Get Taxes for Employee
4. Get Taxes for Year
0. Back to Main Menu
Enter choice: 4

--- Get Taxes for Year ---
Enter Tax Year: 2024
Tax ID: 3
Employee ID: 3
Tax Year: 2024
Taxable Income: 250000.0
Tax Amount: 37500.0
---------------
```

- Financial Reports showing salary and tax summaries.

```
--- Financial Record Management ---
1. Add Financial Record
2. Get Financial Record by ID
3. Get Financial Records for Employee
4. Get Financial Records for Date
0. Back to Main Menu
Enter choice: 1

--- Add Financial Record ---
Enter Employee ID: 6
Record Date (YYYY-MM-DD): 2025-07-01
Description: Bonus
Amount (can be negative for debits if applicable by your logic): 250000
Record Type (e.g., income, expense, bonus, tax payment): bonus

Financial record added successfully:
Record ID: 5
Employee ID: 6
Record Date: 2025-07-01
Description: Bonus
Amount: 250000.0
Record Type: bonus
```

```
--- Financial Record Management ---
1. Add Financial Record
2. Get Financial Record by ID
3. Get Financial Records for Employee
4. Get Financial Records for Date
0. Back to Main Menu
Enter choice: 2

--- Get Financial Record by ID ---
Enter Record ID: 1
Record ID: 1
Employee ID: 1
Record Date: 2024-03-15
Description: Travel Expense Reimbursement
Amount: 150.0
Record Type: reimbursement
```

```
--- Financial Record Management ---
1. Add Financial Record
2. Get Financial Record by ID
3. Get Financial Records for Employee
4. Get Financial Records for Date
0. Back to Main Menu
Enter choice: 3

--- Get Financial Records for Employee ---
Enter Employee ID: 6
Record ID: 5
Employee ID: 6
Record Date: 2025-07-01
Description: Bonus
Amount: 250000.0
Record Type: bonus
--------------

--- Financial Record Management ---
1. Add Financial Record
2. Get Financial Record by ID
3. Get Financial Records for Employee
4. Get Financial Records for Date
0. Back to Main Menu
Enter choice: 4

--- Get Financial Records for Date ---
Enter Record Date (YYYY-MM-DD): 2025-07-01
Record ID: 5
Employee ID: 6
Record Date: 2025-07-01
Description: Bonus
Amount: 250000.0
Record Type: bonus
---------------
```

- Sample error handling (e.g., invalid input, connection issues).

```
====== PayXpert Main Menu ======
1. Employee Management
2. Payroll Management
3. Tax Management
4. Financial Record Management
5. Reporting
0. Exit
Enter your choice: 1

--- Employee Management ---
1. Add New Employee
2. Get Employee by ID
3. Get All Employees
4. Update Employee
5. Remove Employee
0. Back to Main Menu
Enter choice: 2

--- Get Employee by ID ---
Enter Employee ID: 19
Error: Employee with ID 19 not found.
```

```
--- Payroll Management ---
1. Generate Payroll for Employee
2. Get Payroll by ID
3. Get Payrolls for Employee
4. Get Payrolls for Period
0. Back to Main Menu
Enter choice: 1

--- Generate Payroll ---
Enter Employee ID: 9
Pay Period Start Date (YYYY-MM-DD): 22
Invalid date format. Please use YYYY-MM-DD.
Pay Period Start Date (YYYY-MM-DD): 2000-09-09
Pay Period End Date (YYYY-MM-DD): 2000-10-09
Basic Salary: abcd
Invalid input. Please enter a number.
Basic Salary: 20000
Overtime Pay (optional, default 0): a
Invalid input. Please enter a number.
Overtime Pay (optional, default 0): 0
Deductions (optional, default 0): 23
Error: Cannot generate payroll: Employee with ID 9 not found.
```

# Challenges Faced

- Handling date formats during input and database storage.
- Ensuring encapsulation with getters and setters.
- Database column mismatches and corrections.
- Implementing specific attribute update using interactive menu.
- Validating optional fields like Termination Date.
- Integrating tax deductions into salary calculations.

- Generating consolidated financial reports.

## Conclusion

The PayXpert system successfully demonstrates core concepts of OOP, database management, and exception handling. It offers scalable architecture that integrates Employee, Payroll, Tax, and Financial Reporting modules. The system emphasizes clean, modular code, efficient record management, automated financial calculations, and meaningful report generation.

## Future Enhancements

- GUI development for better user experience.
- Enhanced validations and security.
- Multi-user access and concurrency control.
- Role-based authentication.