PayXpert

Name: Samiksha Sandeep Patil

Superset Id: 5273555

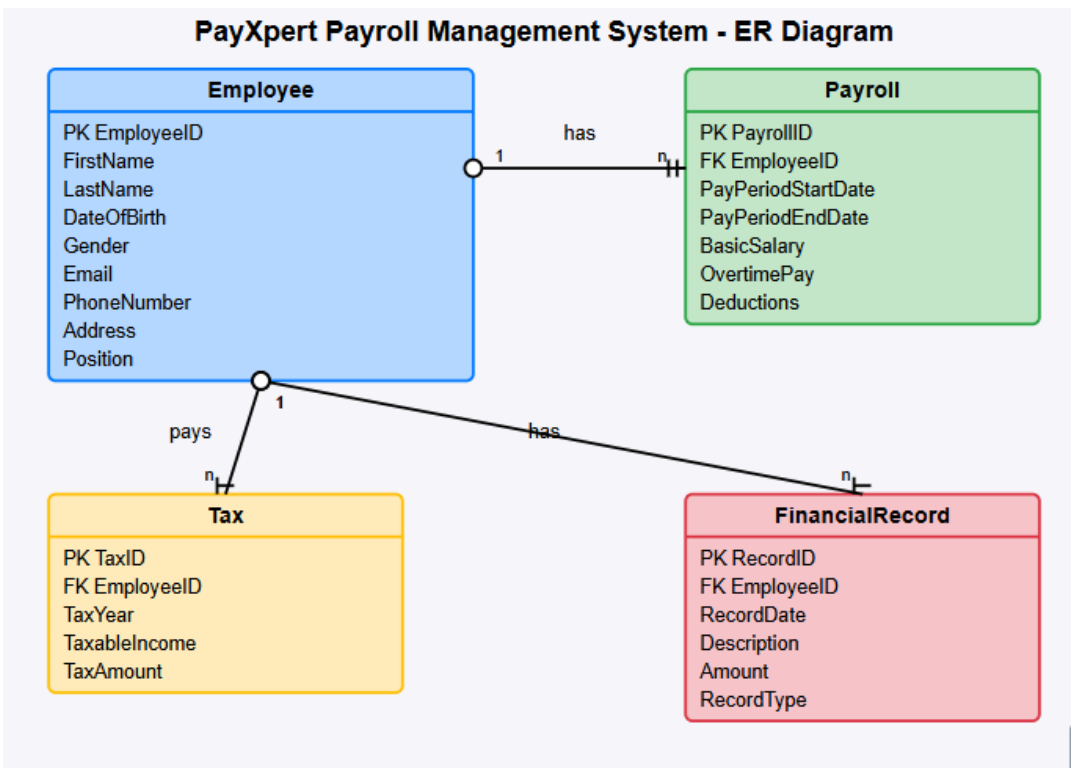
Github link: https://github.com/samikshapatil07/PayXpert

Planning Database

1. Understand the Required Tables

- Employee
- Payroll
- Tax
- FinancialRecord

2. ER Diagram



3. Choose Your Database System

Select a database management system i.e. MySQL.

4. Install Database Tools

Install the necessary database tools:

5. Create the Database

Create a new database specifically for the PayXpert system:

```
CREATE DATABASE PayXpert;
USE PayXpert;
```

6. Create the Employee Table

```
CREATE TABLE Employee (EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
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(20) NOT NULL,Address VARCHAR(200) NOT NULL,
Position VARCHAR(50) NOT NULL,JoiningDate DATE NOT NULL,
TerminationDate DATE NULL);
```

7. Create the Payroll Table

```
CREATE TABLE Payroll (
PayrollID INT PRIMARY KEY AUTO_INCREMENT,
EmployeeID INT NOT NULL,
PayPeriodStartDate DATE NOT NULL,
PayPeriodEndDate DATE NOT NULL,
BasicSalary DECIMAL(10, 2) NOT NULL,
OvertimePay DECIMAL(10, 2) NOT NULL,
Deductions DECIMAL(10, 2) NOT NULL,
NetSalary DECIMAL(10, 2) NOT NULL,
FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);
```

8. Create the Tax Table

```
CREATE TABLE Tax (
TaxID INT PRIMARY KEY AUTO_INCREMENT,
EmployeeID INT NOT NULL,
TaxYear INT NOT NULL,
TaxableIncome DECIMAL(10, 2) NOT NULL,
TaxAmount DECIMAL(10, 2) NOT NULL,
FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);
```

9. Create the FinancialRecord Table

```
CREATE TABLE FinancialRecord (
RecordID INT PRIMARY KEY AUTO_INCREMENT,
EmployeeID INT NOT NULL,
RecordDate DATE NOT NULL,
Description VARCHAR(200) NOT NULL,
Amount DECIMAL(10, 2) NOT NULL,
RecordType VARCHAR(50) NOT NULL,
FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);
```

10. Verify the Table Structure

```
DESCRIBE Employee;
DESCRIBE Payroll;
DESCRIBE Tax;
DESCRIBE FinancialRecord;
```

```
mysql> DESCRIBE Employee;
+-----------------+-------------+------+-----+---------+----------------+
| Field           | Type        | Null | Key | Default | Extra          |
+-----------------+-------------+------+-----+---------+----------------+
| EmployeeID      | int         | NO   | PRI | NULL    | auto_increment |
| FirstName       | varchar(50) | NO   |     | NULL    |                |
| LastName        | varchar(50) | NO   |     | NULL    |                |
| DateOfBirth     | date        | NO   |     | NULL    |                |
| Gender          | varchar(10) | NO   |     | NULL    |                |
| Email           | varchar(100)| NO   |     | NULL    |                |
| PhoneNumber     | varchar(20) | NO   |     | NULL    |                |
| Address         | varchar(200)| NO   |     | NULL    |                |
| Position        | varchar(50) | NO   |     | NULL    |                |
| JoiningDate     | date        | NO   |     | NULL    |                |
| TerminationDate | date        | YES  |     | NULL    |                |
+-----------------+-------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)

mysql> DESCRIBE Payroll;
+--------------------+---------------+------+-----+---------+----------------+
| Field              | Type          | Null | Key | Default | Extra          |
+--------------------+---------------+------+-----+---------+----------------+
| PayrollID          | int           | NO   | PRI | NULL    | auto_increment |
| EmployeeID         | int           | NO   | MUL | NULL    |                |
| PayPeriodStartDate | date          | NO   |     | NULL    |                |
| PayPeriodEndDate   | date          | NO   |     | NULL    |                |
| BasicSalary        | decimal(10,2) | NO   |     | NULL    |                |
| OvertimePay        | decimal(10,2) | NO   |     | NULL    |                |
| Deductions         | decimal(10,2) | NO   |     | NULL    |                |
| NetSalary          | decimal(10,2) | NO   |     | NULL    |                |
+--------------------+---------------+------+-----+---------+----------------+
```

```
mysql> DESCRIBE Tax;
+---------------+---------------+------+-----+---------+----------------+
| Field         | Type          | Null | Key | Default | Extra          |
+---------------+---------------+------+-----+---------+----------------+
| TaxID         | int           | NO   | PRI | NULL    | auto_increment |
| EmployeeID    | int           | NO   | MUL | NULL    |                |
| TaxYear       | int           | NO   |     | NULL    |                |
| TaxableIncome | decimal(10,2) | NO   |     | NULL    |                |
| TaxAmount     | decimal(10,2) | NO   |     | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)

mysql> DESCRIBE FinancialRecord;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| RecordID    | int           | NO   | PRI | NULL    | auto_increment |
| EmployeeID  | int           | NO   | MUL | NULL    |                |
| RecordDate  | date          | NO   |     | NULL    |                |
| Description | varchar(200)  | NO   |     | NULL    |                |
| Amount      | decimal(10,2) | NO   |     | NULL    |                |
| RecordType  | varchar(50)   | NO   |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)
```

11. Insert data

```
--Insert an employee
INSERT INTO Employee (FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber,
 Address, Position, JoiningDate)
VALUES ('Samiksha', 'Patil', '2003-10-20', 'Female', 'samikshapatil419@gmail.com',
 '123-456-7890', 'Kolhapur', 'Developer', '2025-01-01');
```

```
INSERT INTO Employee (FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber,
Address, Position, JoiningDate)
VALUES ('John', 'Doe', '1990-01-15', 'Male', 'john.doe@example.com',
'123-456-7890', '123 Main St', 'Developer', '2023-01-01');
```

```
mysql> INSERT INTO Employee (FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber, Address, Position, JoiningDate)
    -> VALUES ('Samiksha', 'Patil', '2003-10-20', 'Female', 'samikshapatil419@gmail.com', '123-456-7890', 'Kolhapur', 'Developer', '2025-01-01');
Query OK, 1 row affected (0.99 sec)
```

```
- Test that you can retrieve the employee
SELECT * FROM Employee WHERE FirstName = 'Samiksha';
```

```
mysql> SELECT * FROM Employee WHERE FirstName = 'Samiksha';
+------------+-----------+----------+-------------+--------+---------------------------+-------------+----------+----------+-------------+-----------------+
| EmployeeID | FirstName | LastName | DateOfBirth | Gender | Email                     | PhoneNumber | Address  | Position | JoiningDate | TerminationDate |
+------------+-----------+----------+-------------+--------+---------------------------+-------------+----------+----------+-------------+-----------------+
|          2 | Samiksha  | Patil    | 2003-10-20  | Female | samikshapatil419@gmail.com | 123-456-7890 | Kolhapur | Developer | 2025-01-01  | NULL            |
+------------+-----------+----------+-------------+--------+---------------------------+-------------+----------+----------+-------------+-----------------+
1 row in set (0.00 sec)
```

12. To view the data from Employee table:

```
SELECT * FROM Employee;
```

```
mysql> SELECT * FROM Employee;
+------------+-----------+----------+-------------+--------+---------------------------+-------------+-------------+----------+-------------+-----------------+
| EmployeeID | FirstName | LastName | DateOfBirth | Gender | Email                     | PhoneNumber | Address     | Position | JoiningDate | TerminationDate |
+------------+-----------+----------+-------------+--------+---------------------------+-------------+-------------+----------+-------------+-----------------+
|          1 | John      | Doe      | 1990-01-15  | Male   | john.doe@example.com      | 123-456-7890 | 123 Main St | Developer | 2023-01-01  | NULL            |
|          2 | Samiksha  | Patil    | 2003-10-20  | Female | samikshapatil419@gmail.com | 123-456-7890 | Kolhapur    | Developer | 2025-01-01  | NULL            |
+------------+-----------+----------+-------------+--------+---------------------------+-------------+-------------+----------+-------------+-----------------+
2 rows in set (0.00 sec)
```

Java:

Github Link: https://github.com/samikshapatil07/PayXpert

```
PayXpert/
│
├── entity/
│   ├── Employee.java
│   ├── Payroll.java
│   ├── Tax.java
│   └── FinancialRecord.java
│
├── dao/
│   ├── EmployeeService.java
│   ├── PayrollService.java
│   ├── TaxService.java
│   └── FinancialRecordService.java
│   ├── IEmployeeService.java
│   ├── IPayrollService.java
│   ├── ITaxService.java
│   └── IFinancialRecordService.java
│   ├── DAOEmployee.java
│   ├── DAOPayroll.java
│   ├── DAOTaxManagement.java
│   └── DAOFinancialRecord.java
│
├── exception/
│   ├── DatabaseConnectionException.java
│   ├── EmployeeNotFoundException.java
│   ├── FinancialRecordException.java
│   └── InvalidInputException.java
│   ├── PayrollGenerationException .java
│   └── TaxCalculationException .java
│
├── util/
│   ├── DBConnUtil.java
│   └── DBPropertyUtil.java
│
├── services/
│   ├── ValidationService.java
│   └── ReportGenerator.java
│
├── test/
│   └── PayrollServiceTest.java
│
├── main/
    └── MainModule .java
```

Entity Package:

Employee.java

> "This `Employee` class defines all the personal and job-related information of an employee using private variables. It provides constructors for easy object creation and getter/setter methods for encapsulation. The `calculateAge` method helps determine an employee's age using date of birth. `toString()` is overridden to display employee details in a user-friendly way."

```
package entity;

import java.time.LocalDate;
import java.time.Period;

public class Employee {
    private int employeeID;
    private String firstName;
    private String lastName;
    private LocalDate dateOfBirth;
    private String gender;
    private String email;
    private String phoneNumber;
    private String address;
    private String position;
    private LocalDate joiningDate;
    private LocalDate terminationDate;

 // Default constructor
    public Employee() {}
    // Parameterized constructor
    public Employee(int employeeID, String firstName, String lastName, LocalDate
    dateOfBirth,
                String gender, String email, String phoneNumber, String address,
                String position, LocalDate joiningDate, LocalDate terminationDate) {
        this.employeeID = employeeID;
        this.employeeID = employeeID;
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
        this.gender = gender;
        this.email = email;
        this.phoneNumber = phoneNumber;
        this.address = address;
        this.position = position;
        this.joiningDate = joiningDate;
        this.terminationDate = terminationDate;
}
    //all getter setter
    //for employeeID:
        public int getEmployeeID() {
            return employeeID;
        }
        public void setEmployeeID(int employeeID) {
            this.employeeID = employeeID;
        }
```

```
    //for firstName:
      public String getFirstName() {
         return firstName;
      }
      public void setFirstName(String firstName) {
         this.firstName = firstName;
      }
    //for lastName
      public String getLastName() {
         return lastName;
      }
            public void setLastName(String lastName) {
         this.lastName = lastName;
      }
    //for dateOfBirth
      public LocalDate getDateOfBirth() {
         return dateOfBirth;
      }
            public void setDateOfBirth(LocalDate dateOfBirth) {
         this.dateOfBirth = dateOfBirth;
      }
    //for gender
      public String getGender() {
         return gender;
      }
            public void setGender(String gender) {
         this.gender = gender;
      }
    //for email
      public String getEmail() {
         return email;
      }
            public void setEmail(String email) {
         this.email = email;
      }
    //for phoneNumber
      public String getPhoneNumber() {
         return phoneNumber;
      }
            public void setPhoneNumber(String phoneNumber) {
         this.phoneNumber = phoneNumber;
      }
    //for address
      public String getAddress() {
         return address;
      }
            public void setAddress(String address) {
         this.address = address;
      }
    //for position
      public String getPosition() {
         return position;
      }
            public void setPosition(String position) {
         this.position = position;
      }
    //for joiningDate
      public LocalDate getJoiningDate() {
         return joiningDate;
      }
```

```
        public void setJoiningDate(LocalDate joiningDate) {
        this.joiningDate = joiningDate;
    }
  //for terminationDate
    public LocalDate getTerminationDate() {
        return terminationDate;
    }
    public void setTerminationDate(LocalDate terminationDate) {
        this.terminationDate = terminationDate;
    }
// Calculate Age method
public int calculateAge() {
    if (dateOfBirth == null) {
        return 0;
    }
    return Period.between(dateOfBirth, LocalDate.now()).getYears();
}
@Override
public String toString() {
    return "Employee [employeeID=" + employeeID + ", firstName=" + firstName + ",
    lastName=" + lastName +
          ", dateOfBirth=" + dateOfBirth + ", gender=" + gender + ", email=" +
           email +
          ", phoneNumber=" + phoneNumber + ", position=" + position + "]";
   }
}
```

FinancialRecord.java

> "This `FinancialRecord` class models a financial transaction for an employee. It includes identifiers, the transaction date, description, amount, and type. The class uses constructors to initialize objects and follows encapsulation by using private fields with public getter and setter methods."

```
package entity;
import java.time.LocalDate;
public class FinancialRecord {
    private int recordID;
    private int employeeID;
    private LocalDate recordDate;
    private String description;
    private double amount;
    private String recordType;

    // Default constructor
    public FinancialRecord() {}

    // Parameterized constructor
    public FinancialRecord(int recordID, int employeeID, LocalDate recordDate,
                           String description, double amount, String recordType) {
        this.recordID = recordID;
        this.employeeID = employeeID;
        this.recordDate = recordDate;
        this.description = description;
        this.amount = amount;
        this.recordType = recordType;
    }
```

```
// Getters and Setters
//for recordID
public int getrecordID() {
    return recordID;
}
public void setrecordID(int recordID) {
    this.recordID = recordID;
}
//for employeeID
public int getEmployeeID() {
    return employeeID;
}
public void setEmployeeID(int employeeID) {
    this.employeeID = employeeID;
}
//for recordDate:
public LocalDate getrecordDate() {
    return recordDate;
}
public void setrecordDate(LocalDate recordDate) {
     this.recordDate = recordDate;
}
 //for description
 public String getdescription() {
        return description;
    }
  public void setdescription(String description) {
        this.description = description;
    }
 //for amount
  public double getamount() {
      return amount;
  }
          public void setamount(double amount) {
      this.amount = amount;
  }
 //for recordType
  public String getrecordType() {
          return recordType;
          }
          public void setrecordType(String recordType) {
                this.recordType = recordType;
            }
}
```

Payroll.java

> **Stores payroll details per employee per month.**
> **Used by**
> `PayrollService` **for generating, retrieving, and processing payroll records.**

```
package entity;
import java.time.LocalDate;
public class Payroll {
        private int payrollID;
        private int employeeID;
        private LocalDate payPeriodStartDate;
```

```
        private LocalDate payPeriodEndDate;
        private double basicSalary;
        private double overtimePay;
        private double deductions;
        private double netSalary;
        // Default constructor
public Payroll() {}
    // Parameterized constructor
        public Payroll(int payrollID, int employeeID, LocalDate payPeriodStartDate,
                    LocalDate payPeriodEndDate, double basicSalary, double overtimePay,
                    double deductions, double netSalary) {
            this.payrollID = payrollID;
            this.employeeID = employeeID;
            this.payPeriodStartDate = payPeriodStartDate;
            this.payPeriodEndDate = payPeriodEndDate;
            this.basicSalary = basicSalary;
            this.overtimePay = overtimePay;
            this.deductions = deductions;
            this.netSalary = calculateNetSalary();
        }
        private double calculateNetSalary() {
            return basicSalary + overtimePay - deductions;
        }

        // Getters and Setter
        //payrollID
        public int getPayrollID() {
            return payrollID;
        }
        public void setPayrollID(int payrollID) {
            this.payrollID = payrollID;
        }
        //employeeID

        public int getemployeeID() {
            return employeeID;
        }
        public void setemployeeID(int employeeID) {
            this.employeeID = employeeID;
        }
        //payPeriodStartDate
        public LocalDate getpayPeriodStartDate() {
            return payPeriodStartDate;
        }
        public void setpayPeriodStartDate(LocalDate payPeriodStartDate) {
            this.payPeriodStartDate = payPeriodStartDate;
        }
        //payPeriodEndDate
        public LocalDate getpayPeriodEndDate() {
            return payPeriodEndDate;
        }
        public void setpayPeriodEndDate(LocalDate payPeriodEndDate) {
            this.payPeriodEndDate = payPeriodEndDate;
        }
        //basicSalary
        public double getbasicSalary() {
            return basicSalary;
        }
        public void setbasicSalary(double basicSalary) {
            this.basicSalary = basicSalary;
```

```
    }
    //overtimePay
    public double getovertimePay() {
        return overtimePay;
    }
    public void setovertimePay(double d) {
        this.overtimePay = d;
    }
    //deductions
    public double getdeductions() {
        return deductions;
    }
    public void setdeductions(double d) {
        this.deductions = d;
    }
    //netSalary
    public double getnetSalary() {
        return netSalary;
    }
    public void setnetSalary(double d) {
        this.netSalary = d;
    }

    @Override
    public String toString() {
        return "Payroll [payrollID=" + payrollID + ", employeeID=" + employeeID +
                ", payPeriodStartDate=" + payPeriodStartDate + ", payPeriodEndDate="
                 + payPeriodEndDate +
                ", basicSalary=" + basicSalary + ", overtimePay=" + overtimePay +
                ", deductions=" + deductions + ", netSalary=" + netSalary + "]";
    }
}
```

Tax.java

> This class is used by `TaxService` or `DAOTaxManagement` to store and manage tax data. It's useful for generating tax summaries, year-end reports, and verifying deductions

---

```
package entity;

public class Tax {
    private int taxID;
    private int employeeID;
    private int taxYear;
    private double taxableIncome;
    private double taxAmount;

    // Default constructor
    public Tax() {}

    // Parameterized constructor
    public Tax(int taxID, int employeeID, int taxYear, double taxableIncome,
    double taxAmount) {
        this.taxID = taxID;
        this.employeeID = employeeID;
        this.taxYear = taxYear;
        this.taxableIncome = taxableIncome;
        this.taxAmount = taxAmount;
```

```
    }

    // Getters and Setters
    public int getTaxID() {
        return taxID;
    }
    public void setTaxID(int taxID) {
        this.taxID = taxID;
    }

    public int getEmployeeID() {
        return employeeID;
    }
    public void setEmployeeID(int employeeID) {
        this.employeeID = employeeID;
    }

    public int getTaxYear() {
        return taxYear;
    }
    public void setTaxYear(int taxYear) {
        this.taxYear = taxYear;
    }

    public double getTaxableIncome() {
        return taxableIncome;
    }
    public void setTaxableIncome(double taxableIncome) {
        this.taxableIncome = taxableIncome;
    }

    public double getTaxAmount() {
        return taxAmount;
    }
    public void setTaxAmount(double taxAmount) {
        this.taxAmount = taxAmount;
    }
}
```

dao Package:

> The `dao` package in the PayXpert project contains several classes and interfaces that manage data access and business logic related to employees, financial records, payroll, and tax management

Data Access Objects (DAO) Classes:

1. **DAOEmployee.java**: Handles database operations related to employee information, such as adding, updating, deleting, and retrieving employee records.
2. **DAOFinancialRecord.java**: Manages database interactions for financial records, including creating, updating, deleting, and fetching financial data.
3. **DAOPayroll.java**: Responsible for database operations concerning payroll processing, such as inserting, updating, deleting, and retrieving payroll records.
4. **DAOTaxManagement.java**: Handles database tasks related to tax information, including adding, updating, deleting, and fetching tax records.

DAOEmployee.java

```
package dao;

import entity.Employee;

import java.time.LocalDate;
import java.util.Scanner;

public class DAOEmployee {
    private static Scanner scanner = new Scanner(System.in);
    private static EmployeeService employeeService = new EmployeeService();

    public static void manageEmployees() {
        try {
            System.out.println("\n==== Employee Management ====");
            System.out.println("1. Add Employee");
            System.out.println("2. View Employee");
            System.out.println("3. Update Employee");
            System.out.println("4. Delete Employee");
            System.out.println("0. Back");
            System.out.print("Enter choice: ");

            int choice = Integer.parseInt(scanner.nextLine());

            switch (choice) {
                case 1:
                    System.out.print("Enter EmployeeID: ");
                    int employeeID = Integer.parseInt(scanner.nextLine());

                    System.out.print("Enter FirstName: ");
                    String firstName = scanner.nextLine();

                    System.out.print("Enter LastName: ");
                    String lastName = scanner.nextLine();

                    System.out.print("Enter DateOfBirth (yyyy-MM-dd): ");
                    LocalDate dateOfBirth = LocalDate.parse(scanner.nextLine());

                    System.out.print("Enter Gender: ");
                    String gender = scanner.nextLine();

                    System.out.print("Enter Email: ");
                    String email = scanner.nextLine();

                    System.out.print("Enter PhoneNumber: ");
                    String phoneNumber = scanner.nextLine();

                    System.out.print("Enter Address: ");
                    String address = scanner.nextLine();

                    System.out.print("Enter Position: ");
                    String position = scanner.nextLine();

                    System.out.print("Enter JoiningDate (yyyy-MM-dd): ");
                    LocalDate joiningDate = LocalDate.parse(scanner.nextLine());

                    System.out.print("Enter TerminationDate (yyyy-MM-dd): ");
                    String terminationInput = scanner.nextLine();
                    LocalDate terminationDate = null;
                    if (!terminationInput.isBlank()) {
```

```
            terminationDate = LocalDate.parse(terminationInput);
        }

        Employee newEmployee = new Employee();
        newEmployee.setEmployeeID(employeeID);
        newEmployee.setFirstName(firstName);
        newEmployee.setLastName(lastName);
        newEmployee.setDateOfBirth(dateOfBirth);
        newEmployee.setGender(gender);
        newEmployee.setEmail(email);
        newEmployee.setPhoneNumber(phoneNumber);
        newEmployee.setAddress(address);
        newEmployee.setPosition(position);
        newEmployee.setJoiningDate(joiningDate);
        newEmployee.setTerminationDate(terminationDate);

        employeeService.addEmployee(newEmployee);
        System.out.println("Employee added successfully!");
        break;

    case 2:
        System.out.print("Enter employee ID: ");
        int empId = Integer.parseInt(scanner.nextLine());
        Employee emp = employeeService.getEmployeeById(empId);
        System.out.println("Employee details: " + emp);
        break;

    case 3:
        System.out.print("Enter employee ID to update: ");
        int updateId = Integer.parseInt(scanner.nextLine());
        Employee existingEmp = employeeService.getEmployeeById(updateId);
        if (existingEmp == null) {
            System.out.println("Employee not found.");
            break;
        }

System.out.print("Enter new FirstName (" + existingEmp.getFirstName() + "):
 ");
String newFirstName = scanner.nextLine();
if (!newFirstName.isBlank()) existingEmp.setFirstName(newFirstName);

System.out.print("Enter new LastName (" + existingEmp.getLastName() + "): ");
String newLastName = scanner.nextLine();
if (!newLastName.isBlank()) existingEmp.setLastName(newLastName);

System.out.print("Enter new Email (" + existingEmp.getEmail() + "): ");
String newEmail = scanner.nextLine();
if (!newEmail.isBlank()) existingEmp.setEmail(newEmail);

System.out.print("Enter new PhoneNumber (" + existingEmp.getPhoneNumber() +
"):
");
String newPhone = scanner.nextLine();
if (!newPhone.isBlank()) existingEmp.setPhoneNumber(newPhone);

System.out.print("Enter new Address (" + existingEmp.getAddress() + "): ");
String newAddress = scanner.nextLine();
if (!newAddress.isBlank()) existingEmp.setAddress(newAddress);

        System.out.print("Enter new Position (" + existingEmp.getPosition()
```

```
            + "):
            ");
            String newPosition = scanner.nextLine();
            if (!newPosition.isBlank()) existingEmp.setPosition(newPosition);

            employeeService.updateEmployee(existingEmp);
            System.out.println("Employee updated successfully!");
            break;

        case 4:
            System.out.print("Enter employee ID to delete: ");
            int deleteId = Integer.parseInt(scanner.nextLine());
            employeeService.removeEmployee(deleteId);
            System.out.println("Employee deleted successfully!");
            break;
        case 0:
            return;

        default:
                System.out.println("Invalid choice");
            }
        } catch (Exception e) {
            System.out.println("Error in employee management: " + e.getMessage());
        }
    }
}
```

DAOFinancialRecord.java

```
package dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import util.DBConnUtil;

public class DAOFinancialRecord {

    public static void generateFinancialReport(int empId) {

    try (Connection conn = DBConnUtil.getConnection()) {

        double totalIncome = 0;
        double totalDeductions = 0;
        double totalTaxPaid = 0;

        // Fetch financial records
        String recordQuery = "SELECT * FROM FinancialRecord WHERE EmployeeID = ?";
        try (PreparedStatement recStmt = conn.prepareStatement(recordQuery)) {
            recStmt.setInt(1, empId);
            ResultSet recRs = recStmt.executeQuery();

            while (recRs.next()) {
                double amount = recRs.getDouble("Amount");
                String type = recRs.getString("RecordType");

                if ("income".equalsIgnoreCase(type)) {
                    totalIncome += amount;
                } else if ("expense".equalsIgnoreCase(type)) {
```

```
            totalDeductions += amount;
        }
    }
}

// Fetch tax records
String taxQuery = "SELECT SUM(TaxAmount) AS TotalTax FROM Tax
WHERE EmployeeID = ?";
try (PreparedStatement taxStmt = conn.prepareStatement(taxQuery)) {
    taxStmt.setInt(1, empId);
    ResultSet taxRs = taxStmt.executeQuery();
    if (taxRs.next()) {
        totalTaxPaid = taxRs.getDouble("TotalTax");
    }
}

// Fetch employee name
String empName = "Unknown";
String nameQuery = "SELECT FirstName FROM Employee WHERE EmployeeID = ?";
        try (PreparedStatement nameStmt = conn.prepareStatement(nameQuery)) {
            nameStmt.setInt(1, empId);
            ResultSet nameRs = nameStmt.executeQuery();
            if (nameRs.next()) {
                empName = nameRs.getString("FirstName");
            }
        }

        // Output
        System.out.println("\n--- Financial Report ---");
        System.out.println("Employee: " + empName + " (ID: " + empId + ")");
        System.out.printf("  Total Income    : ₹%.2f%n", totalIncome);
        System.out.printf("  Total Deductions: ₹%.2f%n", totalDeductions);
        System.out.printf("  Net Income      : ₹%.2f%n",
        (totalIncome - totalDeductions));
        System.out.printf("  Total Tax Paid  : ₹%.2f%n", totalTaxPaid);

    } catch (SQLException e) {
        System.out.println("Error generating financial report: " + e.getMessage());
    }
  }
}
```

DAOPayroll.java

```
package dao;

import java.time.LocalDate;
import java.util.Scanner;

public class DAOPayroll {
    private static final Scanner scanner = new Scanner(System.in);

    public static void managePayroll() {
        try {
            System.out.println("\n==== Payroll Management ====");
            System.out.println("1. Generate Payroll");
            System.out.println("2. View Payroll");
            System.out.println("0. Back");
            System.out.print("Enter choice: ");
```

```
            int choice = Integer.parseInt(scanner.nextLine());

            switch (choice) {
                case 1:
                    generatePayroll();
                    break;
                case 2:
                    viewPayroll();
                    break;
                case 0:
                    return;
                default:
                    System.out.println("Invalid choice");
            }
        } catch (Exception e) {
            System.out.println("Error in payroll management: " + e.getMessage());
        }
    }

    private static void generatePayroll() {
        try {
            System.out.print("Enter employee ID: ");
            int empId = Integer.parseInt(scanner.nextLine());
            System.out.print("Enter JoiningDate (yyyy-MM-dd): ");
            LocalDate startDate = LocalDate.parse(scanner.nextLine());
            System.out.print("Enter TerminationDate (yyyy-MM-dd): ");
            LocalDate endDate = LocalDate.parse(scanner.nextLine());

            PayrollService payrollService = new PayrollService();
            payrollService.generatePayroll(empId, startDate, endDate); // instance call

            System.out.println("Payroll generated successfully!");
        } catch (Exception e) {
            System.out.println("Failed to generate payroll: " + e.getMessage());
        }
    }


    private static void viewPayroll() {
        try {
            System.out.print("Enter Employee ID: ");
            int empId = Integer.parseInt(scanner.nextLine());
            PayrollService payrollService = new PayrollService();
            payrollService.getPayrollsForEmployee(empId);
        } catch (Exception e) {
            System.out.println("Error retrieving payroll: " + e.getMessage());
        }
    }
}
```

DAOTaxManagement.java

```
package dao;

import entity.Tax;
import exception.EmployeeNotFoundException;
import exception.TaxCalculationException;

import java.util.Scanner;
public class DAOTaxManagement {
```

```
    private static final TaxService taxService = new TaxService();
    private static final Scanner scanner = new Scanner(System.in);

    public static void manageTax() {
        System.out.println("\n--- Tax Calculation ---");
        try {
            System.out.print("Enter Employee ID: ");
            int empId = Integer.parseInt(scanner.nextLine());

            System.out.print("Enter Tax Year (e.g., 2024): ");
            int taxYear = Integer.parseInt(scanner.nextLine());

            Tax tax = taxService.calculateTax(empId, taxYear);

            System.out.println("\nTax Record Created Successfully:");
            printTax(tax);

        } catch (TaxCalculationException | EmployeeNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void printTax(Tax tax) {
        System.out.println("----------------------------------------");
        System.out.println("Tax ID        : " + tax.getTaxID());
        System.out.println("Employee ID   : " + tax.getEmployeeID());
        System.out.println("Tax Year      : " + tax.getTaxYear());
        System.out.printf("Taxable Income : ₹%.2f%n", tax.getTaxableIncome());
        System.out.printf("Tax Amount    : ₹%.2f%n", tax.getTaxAmount());
        System.out.println("----------------------------------------");
    }
}
```

Service Interfaces:

1. **IEmployeeService.java**: Defines the contract for employee-related services, specifying methods for operations like adding, updating, deleting, and retrieving employee information.
2. **IFinancialRecordService.java**: Outlines the methods for managing financial records, including creating, updating, deleting, and fetching financial data.
3. **IPayrollService.java**: Specifies the operations for payroll services, such as processing payroll, updating payroll records, and retrieving payroll information.
4. **ITaxService.java**: Defines the methods for handling tax-related services, including calculating taxes, updating tax information, and retrieving tax records.

IEmployeeService.java:

```
package dao;

import entity.Employee;
import exception.EmployeeNotFoundException;
import exception.InvalidInputException;

import java.util.List;

public interface IEmployeeService {
    Employee getEmployeeById(int employeeId) throws EmployeeNotFoundException;
    List<Employee> getAllEmployees();
    boolean addEmployee(Employee employee) throws InvalidInputException;
    boolean updateEmployee(Employee employee) throws EmployeeNotFoundException,
```

```
    InvalidInputException;
    boolean removeEmployee(int employeeId) throws EmployeeNotFoundException;
}
```

IFinancialRecordService.java

```
package dao;

import entity.FinancialRecord;
import exception.EmployeeNotFoundException;
import exception.FinancialRecordException;

import java.time.LocalDate;
import java.util.List;

public interface IFinancialRecordService {
    boolean addFinancialRecord(int employeeId, String description, double amount,
    String recordType) throws FinancialRecordException, EmployeeNotFoundException;
    FinancialRecord getFinancialRecordById(int recordId) throws
    FinancialRecordException;
    List<FinancialRecord> getFinancialRecordsForEmployee(int employeeId) throws
    EmployeeNotFoundException;
    List<FinancialRecord> getFinancialRecordsForDate(LocalDate recordDate);
}
```

IPayrollService.java

```
package dao;

import entity.Employee;
import entity.Payroll;
import exception.EmployeeNotFoundException;
import exception.PayrollGenerationException;

import java.time.LocalDate;

public interface IPayrollService {
    Payroll generatePayroll(int employeeId, LocalDate startDate, LocalDate endDate)
    throws PayrollGenerationException, EmployeeNotFoundException;
    Payroll getPayrollById(int payrollId) throws PayrollGenerationException;
    void getPayrollsForEmployee(int employeeId) throws EmployeeNotFoundException;
    void getPayrollsForPeriod(LocalDate startDate, LocalDate endDate);
    Employee getEmployeeById(int employeeId);
}
```

ITaxService.java

```
package dao;

import entity.Tax;
import exception.EmployeeNotFoundException;
import exception.TaxCalculationException;

import java.util.List;

public interface ITaxService {
    Tax calculateTax(int employeeId, int taxYear) throws TaxCalculationException,
    EmployeeNotFoundException;
    Tax getTaxById(int taxId) throws TaxCalculationException;
    List<Tax> getTaxesForEmployee(int employeeId) throws EmployeeNotFoundException;
```

```
    List<Tax> getTaxesForYear(int taxYear);
}
```

Service Implementation Classes:

1. **EmployeeService.java**: Implements the `IEmployeeService` interface, providing concrete methods for managing employee data by utilizing `DAOEmployee`.
2. **FinancialRecordService.java**: Implements the `IFinancialRecordService` interface, offering concrete methods for handling financial records through `DAOFinancialRecord`.
3. **PayrollService.java**: Implements the `IPayrollService` interface, delivering concrete methods for processing payroll using `DAOPayroll`.
4. **TaxService.java**: Implements the `ITaxService` interface, providing concrete methods for managing tax information via `DAOTaxManagement`.

EmployeeService.java

```
package dao;

import entity.Employee;
import exception.DatabaseConnectionException;
import exception.EmployeeNotFoundException;
import exception.InvalidInputException;
import util.DBConnUtil;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeService implements IEmployeeService {

    private Connection getConnection() throws DatabaseConnectionException {
        return DBConnUtil.getConnection();
    }

    @Override
    public Employee getEmployeeById(int employeeId) throws EmployeeNotFoundException {
        String sql = "SELECT * FROM Employee WHERE EmployeeID = ?";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            Employee employee = new Employee();
            employee.setEmployeeID(rs.getInt("EmployeeID"));
            employee.setFirstName(rs.getString("FirstName"));
            employee.setLastName(rs.getString("LastName"));
            employee.setDateOfBirth(rs.getDate("DateOfBirth").toLocalDate());
            employee.setGender(rs.getString("Gender"));
            employee.setEmail(rs.getString("Email"));
            employee.setPhoneNumber(rs.getString("PhoneNumber"));
            employee.setAddress(rs.getString("Address"));
            employee.setPosition(rs.getString("Position"));
            employee.setJoiningDate(rs.getDate("JoiningDate").toLocalDate());
```

```
            Date terminationDate = rs.getDate("TerminationDate");
            if (terminationDate != null) {
                employee.setTerminationDate(terminationDate.toLocalDate());
            }

            return employee;
        } else {
            throw new EmployeeNotFoundException("Employee with ID " +
            employeeId + " not found");
        }

    } catch (SQLException | DatabaseConnectionException e) {
        throw new EmployeeNotFoundException("Error retrieving employee: " +
         e.getMessage());
    }
}

@Override
public List<Employee> getAllEmployees() {
List<Employee> employees = new ArrayList<>();
String sql = "SELECT * FROM Employee";

try (Connection conn = getConnection();
     Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {

    while (rs.next()) {
        Employee employee = new Employee();
        employee.setEmployeeID(rs.getInt("EmployeeID"));
        employee.setFirstName(rs.getString("FirstName"));
        employee.setLastName(rs.getString("LastName"));
        employee.setDateOfBirth(rs.getDate("DateOfBirth").toLocalDate());
        employee.setGender(rs.getString("Gender"));
        employee.setEmail(rs.getString("Email"));
        employee.setPhoneNumber(rs.getString("PhoneNumber"));
        employee.setAddress(rs.getString("Address"));
        employee.setPosition(rs.getString("Position"));
        employee.setJoiningDate(rs.getDate("JoiningDate").toLocalDate());

        Date terminationDate = rs.getDate("TerminationDate");
        if (terminationDate != null) {
            employee.setTerminationDate(terminationDate.toLocalDate());
        }

        employees.add(employee);
    }

} catch (SQLException | DatabaseConnectionException e) {
    System.err.println("Error retrieving all employees: " + e.getMessage());
}

return employees;
}

@Override
public boolean addEmployee(Employee employee) throws InvalidInputException {
    if (employee == null) {
        throw new InvalidInputException("Employee cannot be null");
    }
```

```
        if (employee.getFirstName() == null || employee.getFirstName().trim().
        isEmpty()) {
            throw new InvalidInputException("First name cannot be empty");
        }

        String sql = "INSERT INTO Employee (FirstName, LastName, DateOfBirth, Gender,
        Email, " +
                    "PhoneNumber, Address, Position, JoiningDate,
                     TerminationDate) " +
                    "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        try (Connection conn = getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, employee.getFirstName());
            stmt.setString(2, employee.getLastName());
            stmt.setDate(3, Date.valueOf(employee.getDateOfBirth()));
            stmt.setString(4, employee.getGender());
            stmt.setString(5, employee.getEmail());
            stmt.setString(6, employee.getPhoneNumber());
            stmt.setString(7, employee.getAddress());
            stmt.setString(8, employee.getPosition());
            stmt.setDate(9, Date.valueOf(employee.getJoiningDate()));

            if (employee.getTerminationDate() != null) {
                stmt.setDate(10, Date.valueOf(employee.getTerminationDate()));
            } else {
                stmt.setNull(10, Types.DATE);
            }

            int rowsAffected = stmt.executeUpdate();
            return rowsAffected > 0;

        } catch (SQLException | DatabaseConnectionException e) {
            throw new InvalidInputException("Error adding employee: "
            + e.getMessage());
        }
    }

    @Override
    public boolean updateEmployee(Employee employee) throws
    EmployeeNotFoundException,
     InvalidInputException {
        if (employee == null) {
            throw new InvalidInputException("Employee cannot be null");
        }

        // Check if employee exists
        try {
            getEmployeeById(employee.getEmployeeID());
        } catch (EmployeeNotFoundException e) {
            throw e; // Re-throw if employee doesn't exist
        }

        String sql = "UPDATE Employee SET FirstName = ?, LastName = ?,
        DateOfBirth = ?, " +
                    "Gender = ?, Email = ?, PhoneNumber = ?, Address = ?,
                     Position = ?, " +
                    "JoiningDate = ?, TerminationDate = ? WHERE EmployeeID = ?";
```

```
        try (Connection conn = getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, employee.getFirstName());
            stmt.setString(2, employee.getLastName());
            stmt.setDate(3, Date.valueOf(employee.getDateOfBirth()));
            stmt.setString(4, employee.getGender());
            stmt.setString(5, employee.getEmail());
            stmt.setString(6, employee.getPhoneNumber());
            stmt.setString(7, employee.getAddress());
            stmt.setString(8, employee.getPosition());
            stmt.setDate(9, Date.valueOf(employee.getJoiningDate()));

            if (employee.getTerminationDate() != null) {
                stmt.setDate(10, Date.valueOf(employee.getTerminationDate()));
            } else {
                stmt.setNull(10, Types.DATE);
            }
            stmt.setInt(11, employee.getEmployeeID());

            int rowsAffected = stmt.executeUpdate();
            return rowsAffected > 0;

        } catch (SQLException | DatabaseConnectionException e) {
            throw new InvalidInputException("Error updating employee: " +
             e.getMessage());
        }
    }

    @Override
    public boolean removeEmployee(int employeeId) throws EmployeeNotFoundException {
        // Check if employee exists
        try {
            getEmployeeById(employeeId);
        } catch (EmployeeNotFoundException e) {
            throw e; // Re-throw if employee doesn't exist
        }

        String sql = "DELETE FROM Employee WHERE EmployeeID = ?";

        try (Connection conn = getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, employeeId);

            int rowsAffected = stmt.executeUpdate();
            return rowsAffected > 0;

        } catch (SQLException | DatabaseConnectionException e) {
            throw new EmployeeNotFoundException("Error removing employee: " +
            e.getMessage());
        }
    }
}
```

FinancialRecordService.java:

```
package dao;
```

```
import entity.FinancialRecord;
import exception.DatabaseConnectionException;
import exception.EmployeeNotFoundException;
import exception.FinancialRecordException;
import util.DBConnUtil;

//import java.io.IOException;
import java.sql.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class FinancialRecordService implements IFinancialRecordService {

    private final IEmployeeService employeeService;

    public FinancialRecordService() {
        this.employeeService = new EmployeeService();
    }

private Connection getConnection() throws DatabaseConnectionException {
    return DBConnUtil.getConnection();
}

@Override
public boolean addFinancialRecord(int employeeId, String description,
double amount, String recordType)
        throws FinancialRecordException, EmployeeNotFoundException {

    // Check if employee exists
    employeeService.getEmployeeById(employeeId);

    String sql = "INSERT INTO FinancialRecord (EmployeeID, RecordDate,
    Description, Amount, RecordType) " +
                "VALUES (?, ?, ?, ?, ?)";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);
        stmt.setDate(2, Date.valueOf(LocalDate.now()));
        stmt.setString(3, description);
        stmt.setDouble(4, amount);
        stmt.setString(5, recordType);

        int rowsAffected = stmt.executeUpdate();
        return rowsAffected > 0;

    } catch (SQLException | DatabaseConnectionException e) {
        throw new FinancialRecordException("Error adding financial record: "
        + e.getMessage());
    }
}

@Override
public FinancialRecord getFinancialRecordById(int recordId) throws
FinancialRecordException {
    String sql = "SELECT * FROM FinancialRecord WHERE RecordID = ?";

    try (Connection conn = getConnection();
```

```
 PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, recordId);
ResultSet rs = stmt.executeQuery();

if (rs.next()) {
    FinancialRecord record = new FinancialRecord();
    record.setrecordID(rs.getInt("RecordID"));
    record.setEmployeeID(rs.getInt("EmployeeID"));
    record.setrecordDate(rs.getDate("RecordDate").toLocalDate());
    record.setdescription(rs.getString("Description"));
    record.setamount(rs.getDouble("Amount"));
    record.setrecordType(rs.getString("RecordType"));

    return record;
} else {
    throw new FinancialRecordException("Financial record with ID "
     + recordId + " not found");
}

    } catch (SQLException | DatabaseConnectionException e) {
        throw new FinancialRecordException("Error retrieving financial record: "
        + e.getMessage());
    }
}

@Override
public List<FinancialRecord> getFinancialRecordsForEmployee(int employeeId)
throws EmployeeNotFoundException {
    // Check if employee exists
    employeeService.getEmployeeById(employeeId);

    List<FinancialRecord> records = new ArrayList<>();
    String sql = "SELECT * FROM FinancialRecord WHERE EmployeeID = ?";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            FinancialRecord record = new FinancialRecord();
            record.setrecordID(rs.getInt("RecordID"));
            record.setEmployeeID(rs.getInt("EmployeeID"));
            record.setrecordDate(rs.getDate("RecordDate").toLocalDate());
            record.setdescription(rs.getString("Description"));
            record.setamount(rs.getDouble("Amount"));
            record.setrecordType(rs.getString("RecordType"));

            records.add(record);
        }

        return records;

    } catch (SQLException | DatabaseConnectionException e) {
        throw new EmployeeNotFoundException("Error retrieving financial
        records for employee: " + e.getMessage());
    }
}
```

```
@Override
public List<FinancialRecord> getFinancialRecordsForDate(LocalDate recordDate) {
    List<FinancialRecord> records = new ArrayList<>();
    String sql = "SELECT * FROM FinancialRecord WHERE RecordDate = ?";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setDate(1, Date.valueOf(recordDate));
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            FinancialRecord record = new FinancialRecord();
            record.setrecordID(rs.getInt("RecordID"));
            record.setEmployeeID(rs.getInt("EmployeeID"));
            record.setrecordDate(rs.getDate("RecordDate").toLocalDate());
            record.setdescription(rs.getString("Description"));
            record.setamount(rs.getDouble("Amount"));
            record.setrecordType(rs.getString("RecordType"));

            records.add(record);
        }

    } catch (SQLException | DatabaseConnectionException e) {
        System.err.println("Error retrieving financial records for date: "
        + e.getMessage());
    }

    return records;
  }
}
```

PayrollService.java:

```
package dao;

import entity.Employee;
import entity.Payroll;
import exception.DatabaseConnectionException;
import exception.EmployeeNotFoundException;
import exception.PayrollGenerationException;
import util.DBConnUtil;

import java.sql.*;
import java.time.LocalDate;
//import java.util.ArrayList;
//import java.util.List;

public class PayrollService implements IPayrollService {

    private final IEmployeeService employeeService;

    public PayrollService() {
        this.employeeService = new EmployeeService();
    }

    private Connection getConnection() throws DatabaseConnectionException {
        return DBConnUtil.getConnection();
    }
```

```
@Override
public Payroll generatePayroll(int employeeId, LocalDate startDate,
LocalDate endDate)
        throws PayrollGenerationException, EmployeeNotFoundException {

    employeeService.getEmployeeById(employeeId);

    double basicSalary = 5000.0;
    double overtimePay = 1000.0;
    double deductions = 1500.0;
    double netSalary = basicSalary + overtimePay - deductions;

    Payroll payroll = new Payroll();
    payroll.setemployeeID(employeeId);
    payroll.setpayPeriodStartDate(startDate);
    payroll.setpayPeriodEndDate(endDate);
    payroll.setbasicSalary(basicSalary);
    payroll.setovertimePay(overtimePay);
    payroll.setdeductions(deductions);
    payroll.setnetSalary(netSalary);

    String sql = "INSERT INTO Payroll (EmployeeID, PayPeriodStartDate,
    PayPeriodEndDate, " +
            "BasicSalary, OvertimePay, Deductions, NetSalary) " +
            "VALUES (?, ?, ?, ?, ?, ?, ?)";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql,
          Statement.RETURN_GENERATED_KEYS)) {

        stmt.setInt(1, payroll.getemployeeID());
        stmt.setDate(2, Date.valueOf(payroll.getpayPeriodStartDate()));
        stmt.setDate(3, Date.valueOf(payroll.getpayPeriodEndDate()));
        stmt.setDouble(4, payroll.getbasicSalary());
        stmt.setDouble(5, payroll.getovertimePay());
        stmt.setDouble(6, payroll.getdeductions());
        stmt.setDouble(7, payroll.getnetSalary());

        int rowsAffected = stmt.executeUpdate();

        if (rowsAffected > 0) {
            ResultSet rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                payroll.setPayrollID(rs.getInt(1));
            }
            return payroll;
        } else {
            throw new PayrollGenerationException
            ("Failed to generate payroll record");
        }

    } catch (SQLException | DatabaseConnectionException e) {
        throw new PayrollGenerationException("Error generating payroll: "
        + e.getMessage());
    }
}

@Override
public Payroll getPayrollById(int payrollId) throws PayrollGenerationException {
```

```
String sql = "SELECT * FROM Payroll WHERE PayrollID = ?";

try (Connection conn = getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {

    stmt.setInt(1, payrollId);
    ResultSet rs = stmt.executeQuery();

    if (rs.next()) {
        return extractPayrollFromResultSet(rs);
    } else {
        throw new PayrollGenerationException("Payroll with ID " +
        payrollId + " not found");
    }

} catch (SQLException | DatabaseConnectionException e) {
    throw new PayrollGenerationException("Error retrieving payroll: " +
     e.getMessage());
}
}

@Override
public void getPayrollsForEmployee(int employeeId) throws
EmployeeNotFoundException {
    employeeService.getEmployeeById(employeeId);

    String sql = "SELECT * FROM Payroll WHERE EmployeeID = ?";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);
        ResultSet rs = stmt.executeQuery();

        boolean found = false;
        while (rs.next()) {
            found = true;
            Payroll payroll = extractPayrollFromResultSet(rs);
            System.out.println(payroll);
        }

        if (!found) {
            System.out.println("No payroll records found for Employee ID: "
            + employeeId);
        }

    } catch (SQLException | DatabaseConnectionException e) {
        throw new EmployeeNotFoundException("Error retrieving
        payrolls for employee: " + e.getMessage());
    }
}


private Payroll extractPayrollFromResultSet(ResultSet rs) throws SQLException {
    Payroll payroll = new Payroll();
    payroll.setPayrollID(rs.getInt("PayrollID"));
    payroll.setemployeeID(rs.getInt("EmployeeID"));
    payroll.setpayPeriodStartDate(rs.getDate("PayPeriodStartDate").toLocalDate());
    payroll.setpayPeriodEndDate(rs.getDate("PayPeriodEndDate").toLocalDate());
    payroll.setbasicSalary(rs.getDouble("BasicSalary"));
```

```
    payroll.setovertimePay(rs.getDouble("OvertimePay"));
    payroll.setdeductions(rs.getDouble("Deductions"));
    payroll.setnetSalary(rs.getDouble("NetSalary"));
    return payroll;
}

public double calculateNetSalary(double basicSalary, double overtimePay,
double deductions) {
    return basicSalary + overtimePay - deductions;
}

@Override
public Employee getEmployeeById(int employeeId) {
    return null;
}

@Override
public void getPayrollsForPeriod(LocalDate startDate, LocalDate endDate) {
    // TODO Auto-generated method stub

    }
}
```

TaxService.java:

```
package dao;

import entity.Tax;
import exception.DatabaseConnectionException;
import exception.EmployeeNotFoundException;
import exception.TaxCalculationException;
import util.DBConnUtil;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class TaxService implements ITaxService {

    private final IEmployeeService employeeService;

public TaxService() {
    this.employeeService = new EmployeeService();
}

private Connection getConnection() throws DatabaseConnectionException {
    return DBConnUtil.getConnection();
}

@Override
public Tax calculateTax(int employeeId, int taxYear) throws
TaxCalculationException, EmployeeNotFoundException {
    employeeService.getEmployeeById(employeeId);

    String payrollQuery = "SELECT SUM(GrossSalary) AS TotalIncome,
     SUM(Deductions) AS TotalDeductions " +
     "FROM Payroll WHERE EmployeeID = ? AND YEAR(PayPeriodEndDate) = ?";

    double totalIncome = 0.0;
    double totalDeductions = 0.0;
```

```
try (Connection conn = getConnection();
     PreparedStatement stmt = conn.prepareStatement(payrollQuery)) {

    stmt.setInt(1, employeeId);
    stmt.setInt(2, taxYear);

    ResultSet rs = stmt.executeQuery();
    if (rs.next()) {
        totalIncome = rs.getDouble("TotalIncome");
        totalDeductions = rs.getDouble("TotalDeductions");
    }

    double taxableIncome = totalIncome - totalDeductions;
    double taxAmount = calculateTaxAmount(taxableIncome);

    String insertSql = "INSERT INTO Tax (EmployeeID, TaxYear,
  TaxableIncome, TaxAmount) VALUES (?, ?, ?, ?)";
    try (PreparedStatement insertStmt = conn.prepareStatement(insertSql,
  Statement.RETURN_GENERATED_KEYS)) {
        insertStmt.setInt(1, employeeId);
        insertStmt.setInt(2, taxYear);
        insertStmt.setDouble(3, taxableIncome);
        insertStmt.setDouble(4, taxAmount);

        int rowsAffected = insertStmt.executeUpdate();
        if (rowsAffected > 0) {
            ResultSet generatedKeys = insertStmt.getGeneratedKeys();
            if (generatedKeys.next()) {
                int taxId = generatedKeys.getInt(1);
                Tax tax = new Tax();
                tax.setTaxID(taxId);
                tax.setEmployeeID(employeeId);
                tax.setTaxYear(taxYear);
                tax.setTaxableIncome(taxableIncome);
                tax.setTaxAmount(taxAmount);
                return tax;
            }
        }
        throw new TaxCalculationException("Failed to insert tax record.");
    }
    } catch (SQLException | DatabaseConnectionException e) {
        throw new TaxCalculationException("Error calculating tax: " +
        e.getMessage());
    }
}

private double calculateTaxAmount(double taxableIncome) {
    if (taxableIncome <= 250000) {
        return 0;
    } else if (taxableIncome <= 500000) {
        return taxableIncome * 0.05;
    } else if (taxableIncome <= 1000000) {
        return taxableIncome * 0.20;
    } else {
        return taxableIncome * 0.30;
    }
}
```

```
@Override
public Tax getTaxById(int taxId) throws TaxCalculationException {
    String sql = "SELECT * FROM Tax WHERE TaxID = ?";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, taxId);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            Tax tax = new Tax();
            tax.setTaxID(rs.getInt("TaxID"));
            tax.setEmployeeID(rs.getInt("EmployeeID"));
            tax.setTaxYear(rs.getInt("TaxYear"));
            tax.setTaxableIncome(rs.getDouble("TaxableIncome"));
            tax.setTaxAmount(rs.getDouble("TaxAmount"));

            return tax;
        } else {
            throw new TaxCalculationException("Tax with ID " + taxId +
            " not found");
        }

    } catch (SQLException | DatabaseConnectionException e) {
        throw new TaxCalculationException("Error retrieving tax: " +
        e.getMessage());
    }
}

@Override
public List<Tax> getTaxesForEmployee(int employeeId) throws
 EmployeeNotFoundException {
    employeeService.getEmployeeById(employeeId);

    List<Tax> taxes = new ArrayList<>();
    String sql = "SELECT * FROM Tax WHERE EmployeeID = ?";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            Tax tax = new Tax();
            tax.setTaxID(rs.getInt("TaxID"));
            tax.setEmployeeID(rs.getInt("EmployeeID"));
            tax.setTaxYear(rs.getInt("TaxYear"));
            tax.setTaxableIncome(rs.getDouble("TaxableIncome"));
            tax.setTaxAmount(rs.getDouble("TaxAmount"));

            taxes.add(tax);
        }

        return taxes;

    } catch (SQLException | DatabaseConnectionException e) {
        throw new EmployeeNotFoundException
        ("Error retrieving taxes for employee: " + e.getMessage());
```

```
    }
}

@Override
public List<Tax> getTaxesForYear(int taxYear) {
    List<Tax> taxes = new ArrayList<>();
    String sql = "SELECT * FROM Tax WHERE TaxYear = ?";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, taxYear);
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            Tax tax = new Tax();
            tax.setTaxID(rs.getInt("TaxID"));
            tax.setEmployeeID(rs.getInt("EmployeeID"));
            tax.setTaxYear(rs.getInt("TaxYear"));
            tax.setTaxableIncome(rs.getDouble("TaxableIncome"));
            tax.setTaxAmount(rs.getDouble("TaxAmount"));

            taxes.add(tax);
        }

    } catch (SQLException | DatabaseConnectionException e) {
        System.err.println("Error retrieving taxes for year: "
        + e.getMessage());
    }

    return taxes;
  }
}
```

Exception Package

> The `exception` package in the PayXpert project contains custom exception classes that handle specific error scenarios within the application.

FinancialRecordException.java

FinancialRecordException.java: This class extends the `Exception` class and is used to signal issues related to financial records. It provides multiple constructors to allow for different ways of initializing the exception, including with a custom message or a cause.

```
package exception;

public class FinancialRecordException extends Exception {
    private static final long serialVersionUID = 1L;

    public FinancialRecordException() {
        super();
    }

    public FinancialRecordException(String message) {
        super(message);
    }

    public FinancialRecordException(String message, Throwable cause) {
```

```
        super(message, cause);
    }

    public FinancialRecordException(Throwable cause) {
        super(cause);
    }
}
```

PayrollGenerationException.java

PayrollGenerationException.java: Extends the `Exception` class and is thrown when errors occur during payroll processing. This exception helps in identifying and handling payroll-specific issues separately from other exceptions.

```
package exception;

public class PayrollGenerationException extends Exception {
 private static final long serialVersionUID = 1L;

 public PayrollGenerationException() {
    super();
 }

 public PayrollGenerationException(String message) {
    super(message);
 }

 public PayrollGenerationException(String message, Throwable cause) {
    super(message, cause);
 }

 public PayrollGenerationException(Throwable cause) {
    super(cause);
 }
}
```

TaxCalculationException.java

TaxCalculationException.java: This exception is raised when there are errors in tax calculation processes. By creating a specific exception for tax calculation errors, the application can provide more precise error handling and messaging related to tax operations.

```
// Tax calculation exception
package exception;

public class TaxCalculationException extends Exception {
private static final long serialVersionUID = 1L;

public TaxCalculationException() {
  super();
}

public TaxCalculationException(String message) {
  super(message);
}

public TaxCalculationException(String message, Throwable cause) {
  super(message, cause);
}
```

```
public TaxCalculationException(Throwable cause) {
  super(cause);
}
}
```

EmployeeNotFound.java

EmployeeNotFound.java: This exception is thrown when there are issues related to employee operations, such as adding, updating, or retrieving employee information. It helps in identifying and handling errors specific to employee management.

```
package exception;

public class EmployeeNotFoundException extends Exception {
    private static final long serialVersionUID = 1L;

    public EmployeeNotFoundException() {
        super();
    }

    public EmployeeNotFoundException(String message) {
        super(message);
    }

    public EmployeeNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }

    public EmployeeNotFoundException(Throwable cause) {
        super(cause);
    }
}
```

DatabaseConnectionException.java

DatabaseConnectionException.java: This exception is used to indicate problems that occur during database interactions. It serves as a general exception for various database-related errors, such as connection failures, query issues, or data integrity violations.

```
// Database connection exception
package exception;

public class DatabaseConnectionException extends Exception {
    private static final long serialVersionUID = 1L;

    public DatabaseConnectionException() {
        super("Failed to connect to database!");
    }

    public DatabaseConnectionException(String message) {
        super(message);
    }
}
```

InvalidInputException.java

InvalidInputException.java: This exception is raised when authentication processes fail, such as when user credentials are invalid or when there are issues during user login. It aids in managing errors related to user authentication and access control.

```
// Invalid input exception

package exception;

public class InvalidInputException extends Exception {
    private static final long serialVersionUID = 1L;

    public InvalidInputException() {
        super();
    }

    public InvalidInputException(String message) {
        super(message);
    }

    public InvalidInputException(String message, Throwable cause) {
        super(message, cause);
    }

    public InvalidInputException(Throwable cause) {
        super(cause);
    }
}
```

Util Package

> The `util` package in the PayXpert project contains utility classes that provide essential functionalities to support the application's operations

DBConnUtil.java

DBConnUtil.java: This class manages the database connection for the application. It provides methods to establish and close connections to the database, ensuring efficient and reliable communication between the application and the database..

```
package util;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DBConnUtil {

    private static final String fileName="db.properties";
    public static Connection getConnection() {
        Connection con=null;
        String connString=null;
        try {
        connString=DBPropertyUtil.getConnection(fileName);
        }catch (IOException e) {
            System.out.println("Connection String Creation Failed");
            e.printStackTrace();
        }
        if(connString!=null) {
            try {
                con=DriverManager.getConnection(connString);
```

```
        }catch (SQLException e) {
            System.out.println("Error While Establishing DBConnection.....");
            e.printStackTrace();
        }
      }
      return con;
   }
}
```

DBPropertyUtil.java

DBPropertyUtil.java: This class handles the retrieval and management of database configuration properties. It reads database-related settings, such as URL, username, and password, from configuration files or environment variables, facilitating flexible and secure database connectivity.

```
package util;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
public class DBPropertyUtil {
   //this method takes the file name which contains
   //user name, password, port number , protocol and database name as an argument
   //and returns a connection string
   public static String getConnection(String fileName)throws IOException {
      //fileName="db.properties"
      String connStr=null;
      Properties props=new Properties();
      FileInputStream fis=new FileInputStream(fileName);
      props.load(fis);
      String user=props.getProperty("user");
      String password=props.getProperty("password");
      String protocol=props.getProperty("protocol");
      String system=props.getProperty("system");
      String database=props.getProperty("database");
      String port=props.getProperty("port");
      connStr=protocol+"//"+system+":"+port+"/"+database+"?user="+user+"&password="
      +password;
      return  connStr;
   }
}
```

Test Package

> The `test` package in the PayXpert project contains unit tests to ensure the correctness and reliability of the application's components.

PayrollServiceTest

PayrollServiceTest.java: This class contains test cases for the `PayrollService` class, verifying that payroll processing functions as intended. It utilizes mock data and assertions to validate various payroll scenarios, ensuring that calculations and data handling are accurate.

```
package test;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

import entity.Employee;
```

```
import entity.Payroll;
import entity.Tax;
import exception.EmployeeNotFoundException;
import exception.InvalidInputException;
import dao.EmployeeService;
import dao.PayrollService;
import dao.TaxService;

import java.util.List;

public class PayrollServiceTest {

    private EmployeeService employeeService;

    @Before
    public void setup() {
        new PayrollService();
        new TaxService();
        employeeService = new EmployeeService();
    }

//---------- TestCase1: CalculateGrossSalaryForEmployee
    @Test
    public void testCase1_calculateGrossSalary() {
        double basic = 5000, overtime = 500, expected = 5500;
        Payroll p = new Payroll();
        p.setbasicSalary(basic);
        p.setovertimePay(overtime);
        double actual = p.getbasicSalary() + p.getovertimePay();
        assertEquals(expected, actual, 0.01);
    }

//--------- TestCase2: CalculateNetSalaryAfterDeductions
    @Test
    public void testCase2_calculateNetSalary() {
        double basic = 5000, overtime = 500, deductions = 1000, expected = 4500;
        Payroll p = new Payroll();
        p.setbasicSalary(basic);
        p.setovertimePay(overtime);
        p.setdeductions(deductions);
        p.setnetSalary((basic + overtime) - deductions);
        assertEquals(expected, p.getnetSalary(), 0.01);
    }

//---------- TestCase3: VerifyTaxCalculationForHighIncomeEmployee
    @Test
    public void testCase3_VerifyTaxCalculationForHighIncomeEmployee() {
        Tax tax = new Tax();
        tax.setEmployeeID(5);
        tax.setTaxableIncome(120000);
        tax.setTaxAmount(36000);

        assertEquals("Tax calculation for high income incorrect", 36000,
        tax.getTaxAmount(), 0.01);
    }


//---------- TestCase4: ProcessPayrollForMultipleEmployees
    @Test
    public void testCase4_processPayrollBatch() {
```

```
        Payroll p1 = new Payroll();
        p1.setnetSalary(4500);
        Payroll p2 = new Payroll();
        p2.setnetSalary(6200);
        Payroll p3 = new Payroll();
        p3.setnetSalary(3900);

        List<Payroll> payrolls = List.of(p1, p2, p3);
        assertEquals(3, payrolls.size());
        double total = payrolls.stream().mapToDouble(Payroll::getnetSalary).sum();
        assertEquals(14600.0, total, 0.01);
    }

//------- TestCase5: Verify_Error_Handling_For_Invalid_Employee_Data
    @Test(expected = EmployeeNotFoundException.class)
    public void testCase5_invalidEmployeeId() throws EmployeeNotFoundException {
        employeeService.getEmployeeById(9999);
    }
    @Test(expected = InvalidInputException.class)
    public void testCase5_invalidEmailFormat() throws InvalidInputException {
        Employee e = new Employee();
        e.setEmployeeID(100);
        e.setEmail("invalid-email");
        employeeService.addEmployee(e);
    }
}
```

Main Package

> The `main` package in the PayXpert project contains the primary class responsible for launching the application:

MainModule

MainModule.java: This class serves as the entry point for the PayXpert application. It initializes the necessary components and orchestrates the startup process, ensuring that all services and configurations are properly set up before the application begins handling operations.

```
package main;

import dao.DAOEmployee;
import dao.DAOPayroll;
import dao.DAOTaxManagement;
import dao.DAOFinancialRecord;
import java.util.Scanner;

public class MainModule {
    private static final Scanner scanner = new Scanner(System.in);


    public static void main(String[] args) {
        try {
            boolean exit = false;
            while (!exit) {
                System.out.println("\n==== PayXpert: Payroll Management System ====");
                System.out.println("1. Employee Management");
                System.out.println("2. Payroll Processing");
                System.out.println("3. Tax Management");
                System.out.println("4. Financial Records");
```

```
            System.out.println("0. Exit");
            System.out.print("Enter your choice: ");

            int choice = Integer.parseInt(scanner.nextLine());

            switch (choice) {
                case 1:
                    DAOEmployee.manageEmployees();
                    break;
                case 2:
                    DAOPayroll.managePayroll();
                    break;
                case 3:
                    DAOTaxManagement.manageTax();
                    break;
                case 4:
                    System.out.print("Enter Employee ID to view Financial Report: ");
                    int empId = scanner.nextInt();
                    DAOFinancialRecord.generateFinancialReport(empId);
                    break;
                case 0:
                    exit = true;
                    System.out.println("Thank you for using PayXpert");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
                }
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```