**Hexaware Technologies**

Case Study

PayXpert, The Payroll Management System

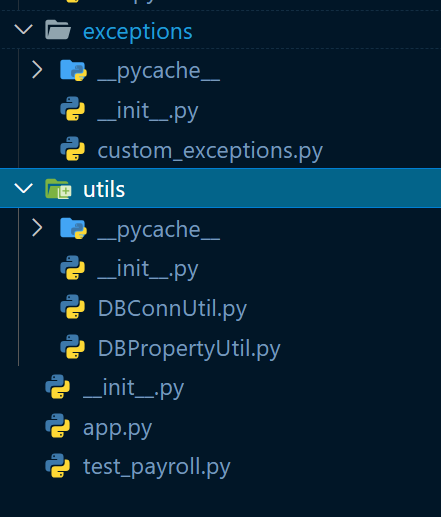
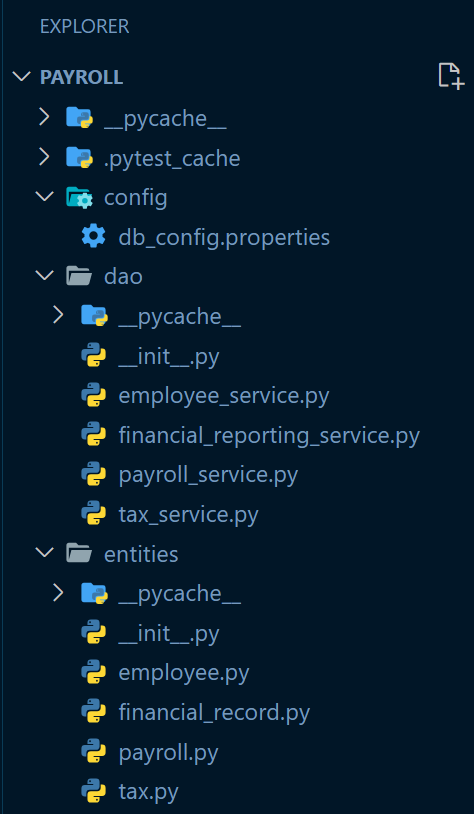
**1. Introduction**

The **PayXpert Payroll Management System** is designed to automate the processes of employee management, payroll processing, tax calculation, and financial reporting. This application is built using Python for the backend logic and Microsoft SQL Server (MSSQL) for database management. The goal is to provide a seamless interface for managing employee data, calculating payroll, and generating financial reports, ultimately increasing efficiency and reducing errors in manual processes.

**2. Project Structure**

The project follows a structured directory format to maintain code organization and readability. The primary packages include:

* **entity**: Contains classes that represent the data model for the application, corresponding to the database tables.
* **dao**: Contains data access objects that handle interactions with the database, including SQL queries and data manipulation.
* **exception**: Includes custom exception classes to handle specific errors that may arise during the execution of the application.
* **util**: Houses utility classes that assist in database connections and configuration management.
* **main**: Contains the main module to demonstrate the application's functionalities through a user interface or command-line interaction.

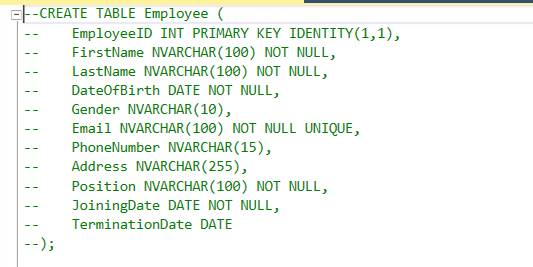


**3. Task Breakdown**

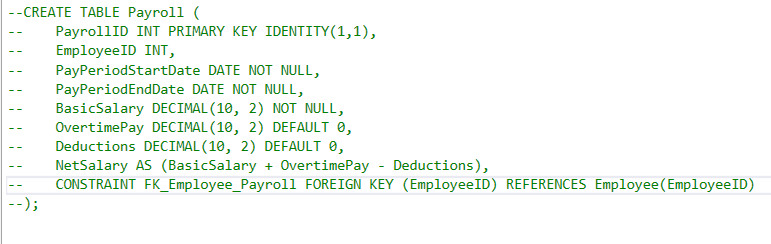
**Task 1: SQL Schema Design**

In this task, we designed the SQL schema for the application by creating tables in the MSSQL database. The schema consists of four main tables: **Employee**, **Payroll**, **Tax**, and **FinancialRecord**. Each table is defined with appropriate columns and data types that reflect the real-world entities they represent.

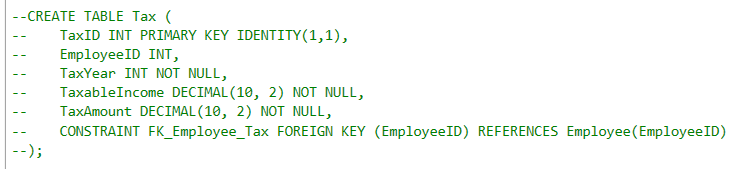
* **Employee Table**: Stores personal details of employees, including identifiers, names, contact information, position, and employment history.



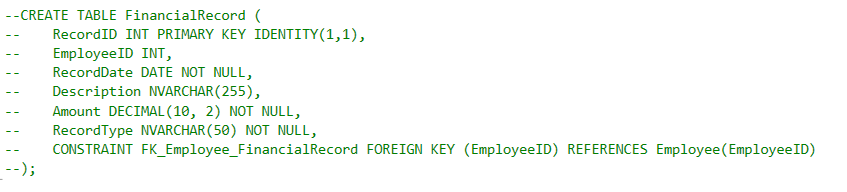
* **Payroll Table**: Captures payroll records for employees, detailing pay periods, salaries, overtime, deductions, and net salary.



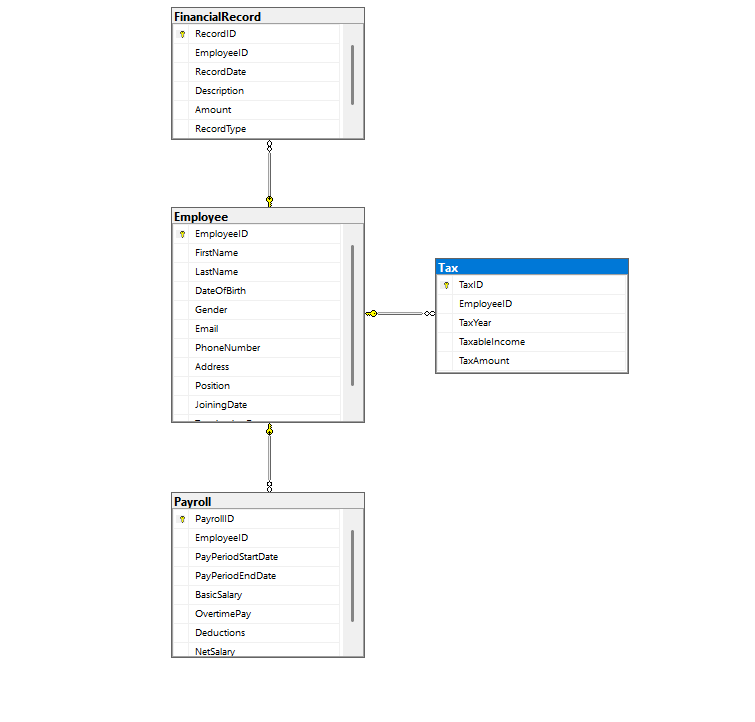
* **Tax Table**: Contains tax-related information, including taxable income and tax amounts for each employee by year.



* **FinancialRecord Table**: Records financial transactions related to employees, categorizing them as income, expenses, or tax payments.



This structured schema allows for efficient data retrieval and manipulation necessary for payroll processing and reporting.



**Task 2: Entity Classes (Model Layer)**

We created the model classes in the **entity** package to represent the database tables. Each class corresponds to a table in the database and encapsulates the related properties. For example, the Employee class includes attributes such as EmployeeID, FirstName, and Position.

* Each class has private variables, along with constructors (both default and parameterized) to initialize the objects.
* Getters and setters provide controlled access to these variables.
* The Employee class includes a method to calculate the age of the employee based on their date of birth.

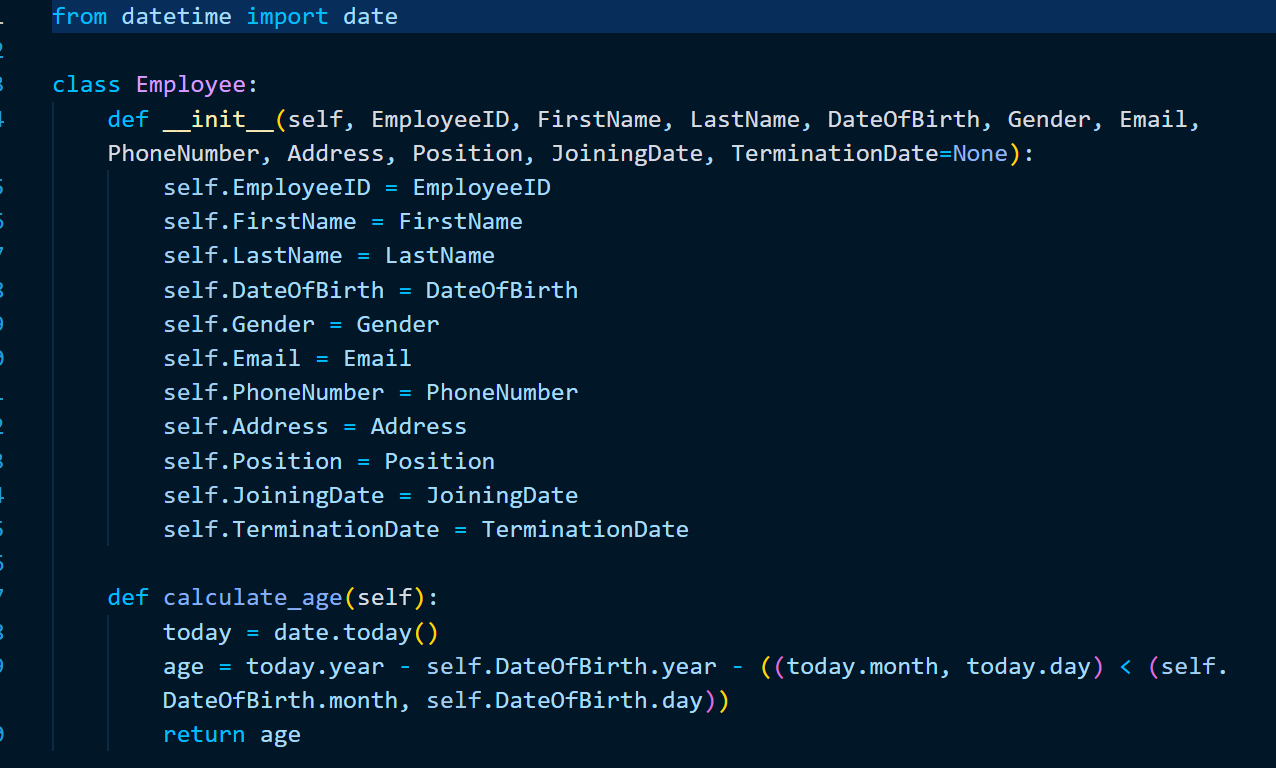
This object-oriented approach promotes encapsulation and reusability of code, making it easier to manage employee-related data.

Classes:

* **Employee:**

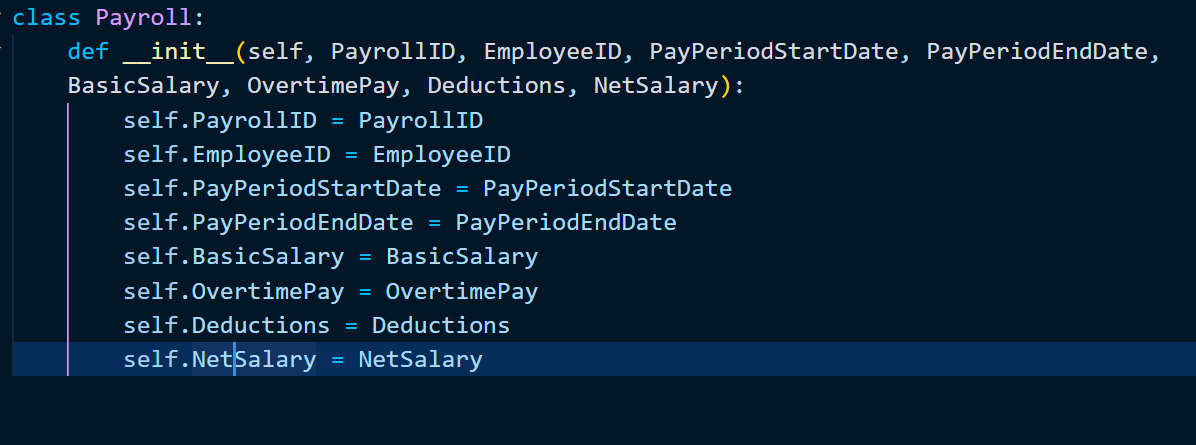
Properties: EmployeeID, FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber, Address, Position, JoiningDate, TerminationDate

Methods: CalculateAge()



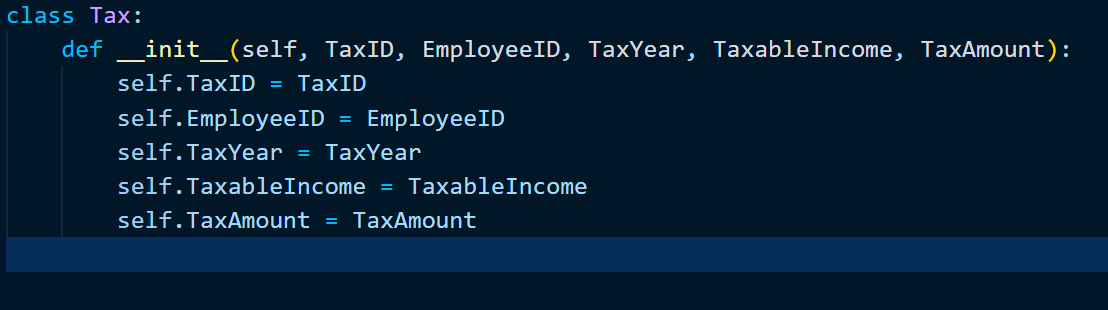
* **Payroll:**

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



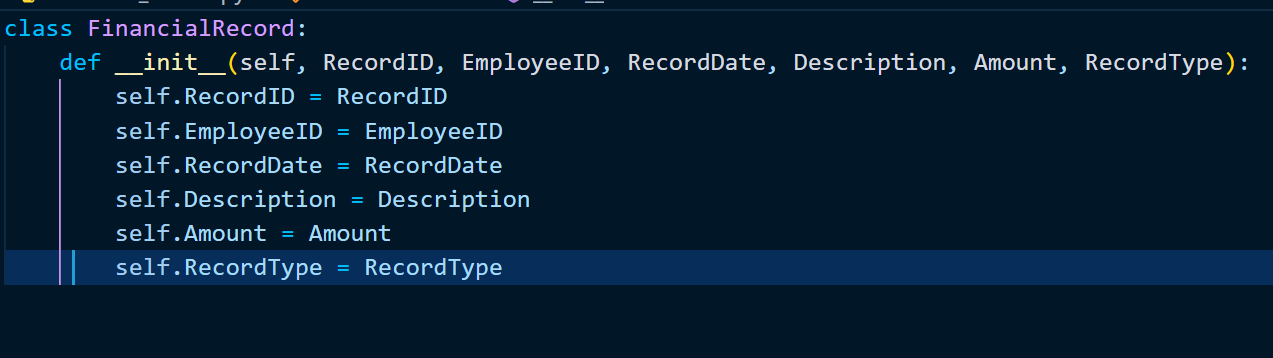
* **Tax:**

Properties: TaxID, EmployeeID, TaxYear, TaxableIncome, TaxAmount



* **FinancialRecord:**

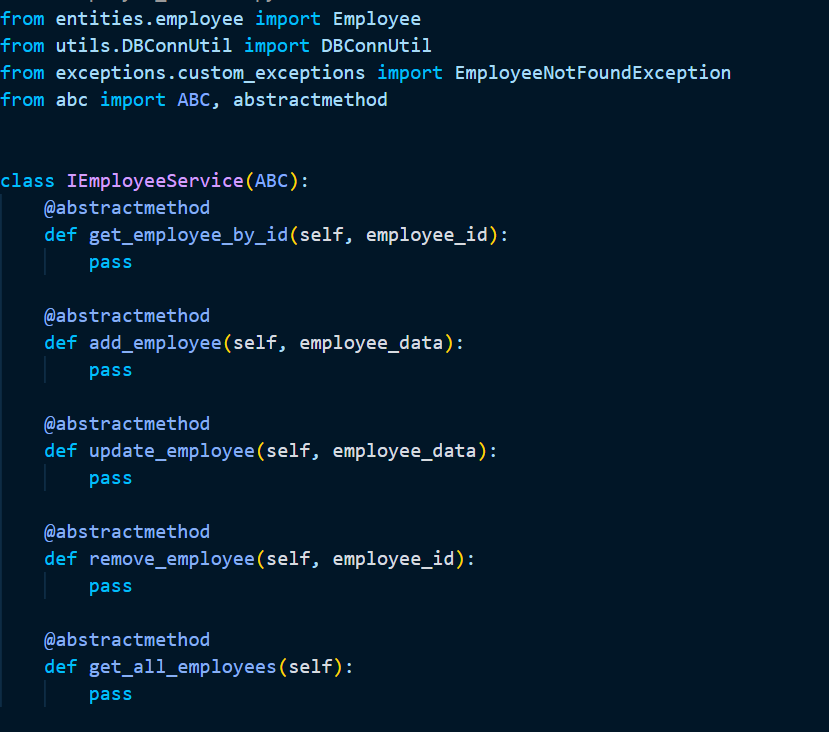
Properties: RecordID, EmployeeID, RecordDate, Description, Amount, RecordType



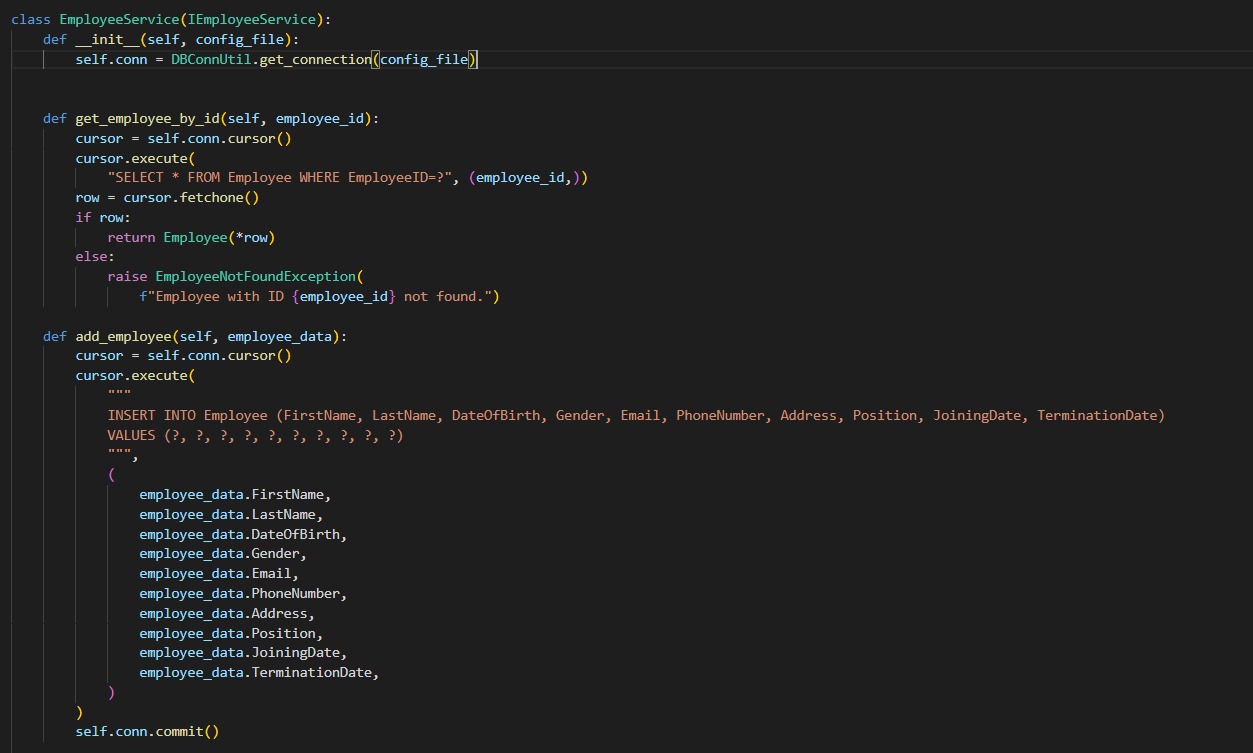
* **EmployeeService** (implements IEmployeeService):

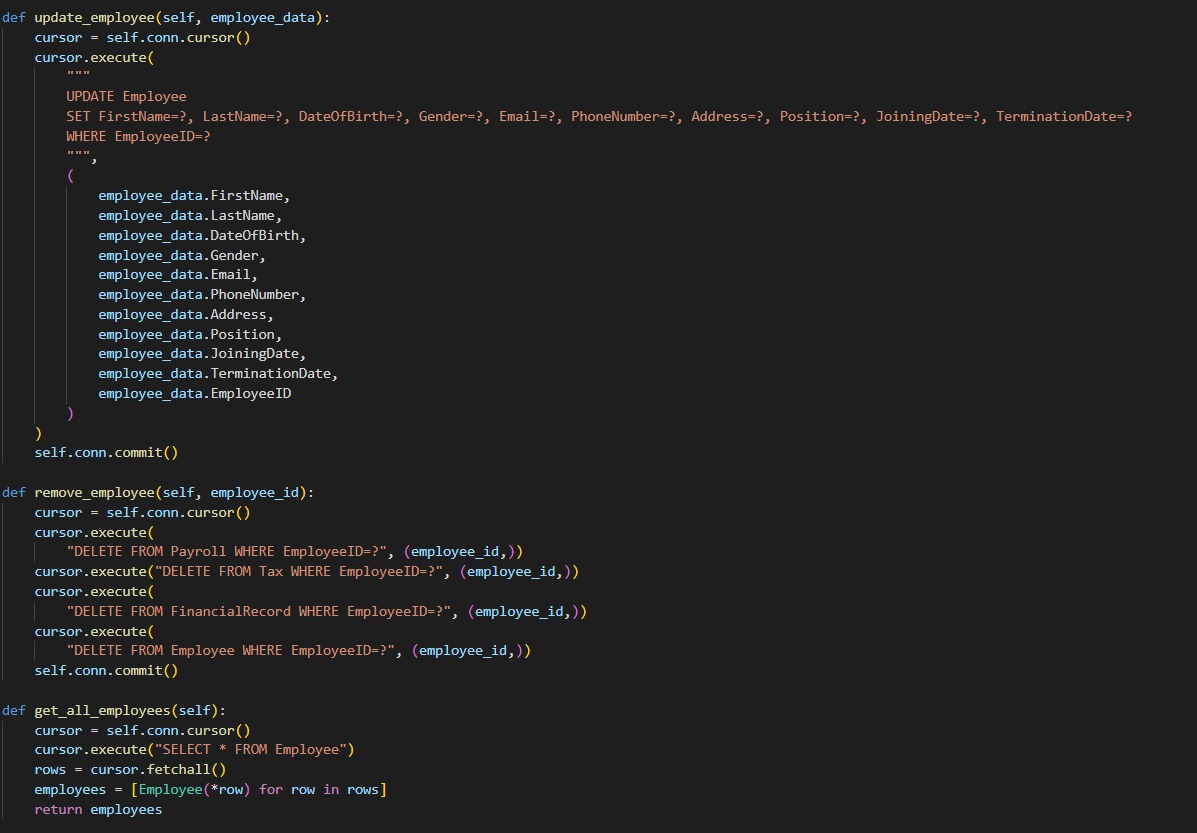
Methods: GetEmployeeById, GetAllEmployees, AddEmployee, UpdateEmployee, RemoveEmployee

* **IEmployeeService:**



* **EmployeeService:**

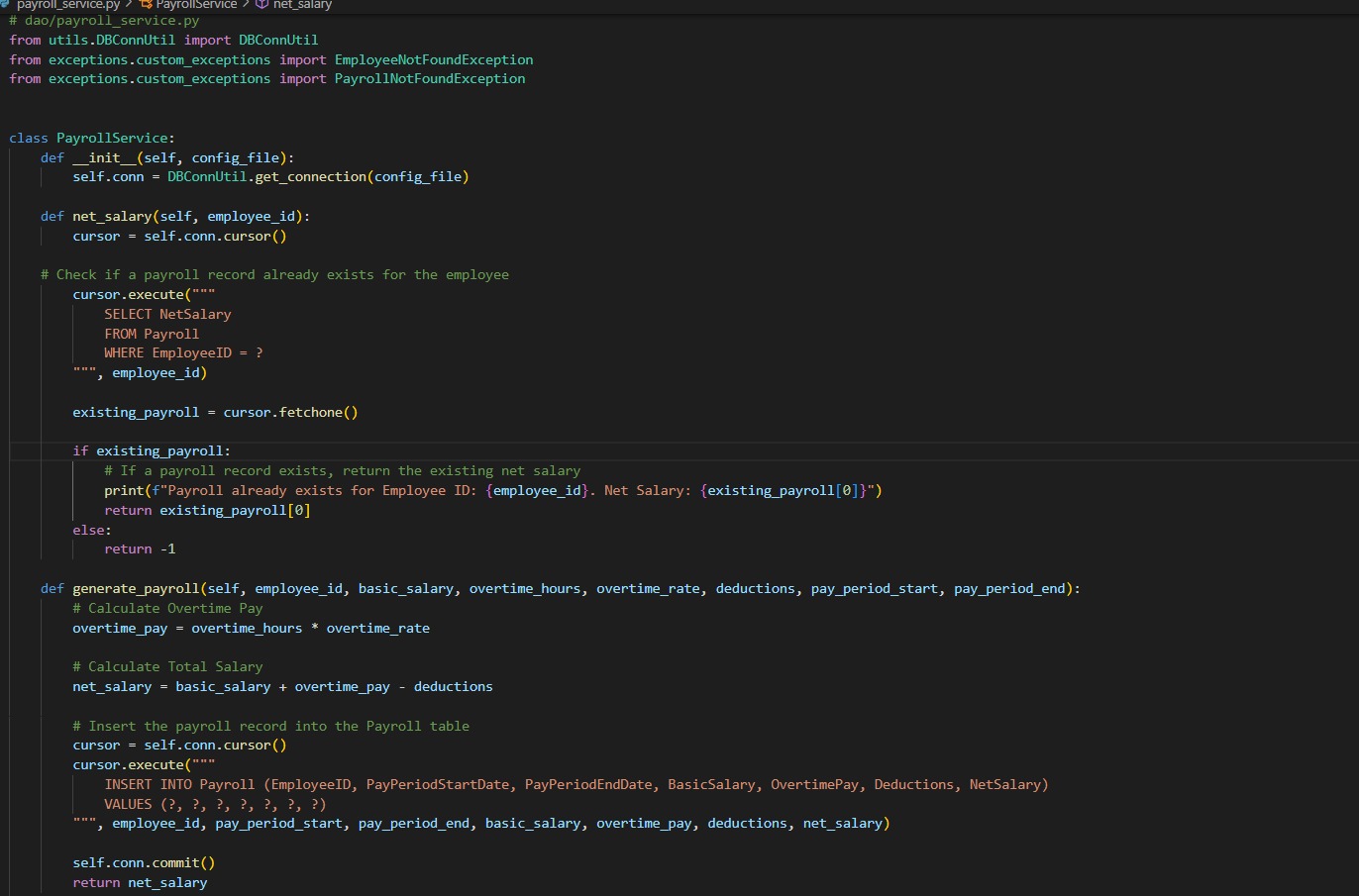


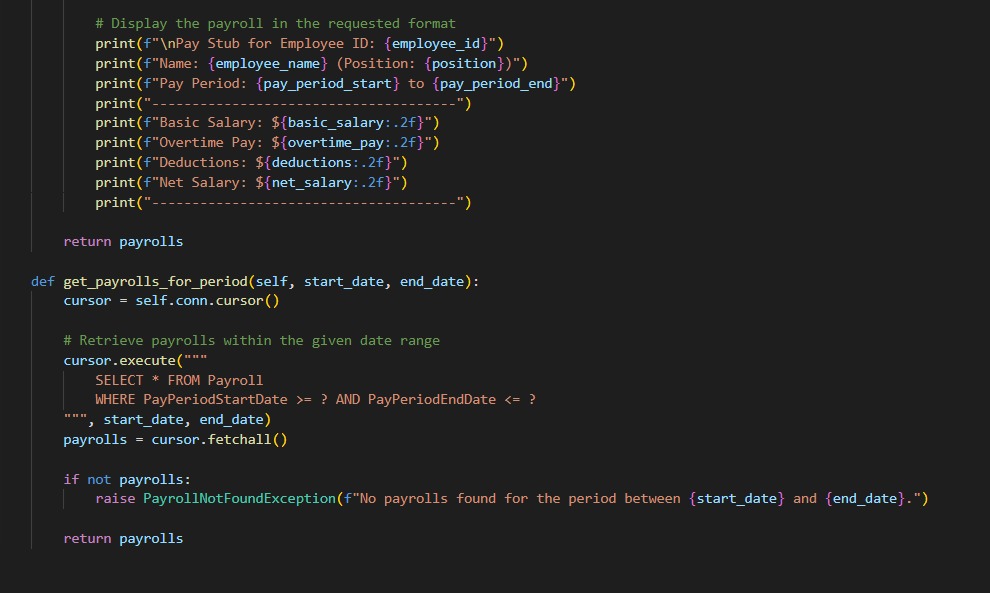


* **PayrollService** (implements IPayrollService):

Methods: GeneratePayroll, GetPayrollById, GetPayrollsForEmployee, GetPayrollsForPeriod

* **PayrollService**

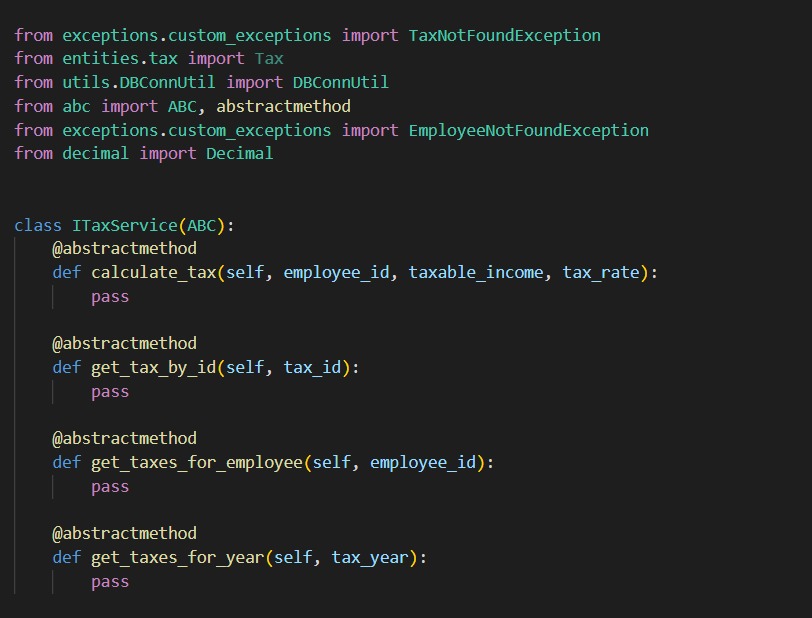


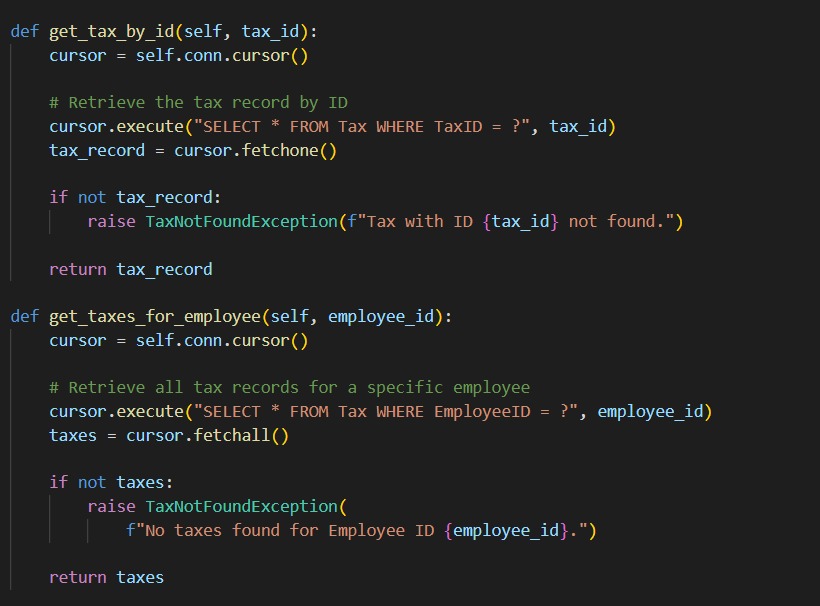


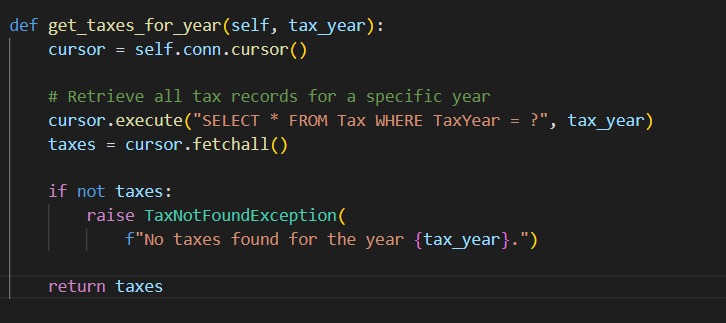
* **TaxService** (implements ITaxService):

Methods: CalculateTax, GetTaxById, GetTaxesForEmployee, GetTaxesForYear

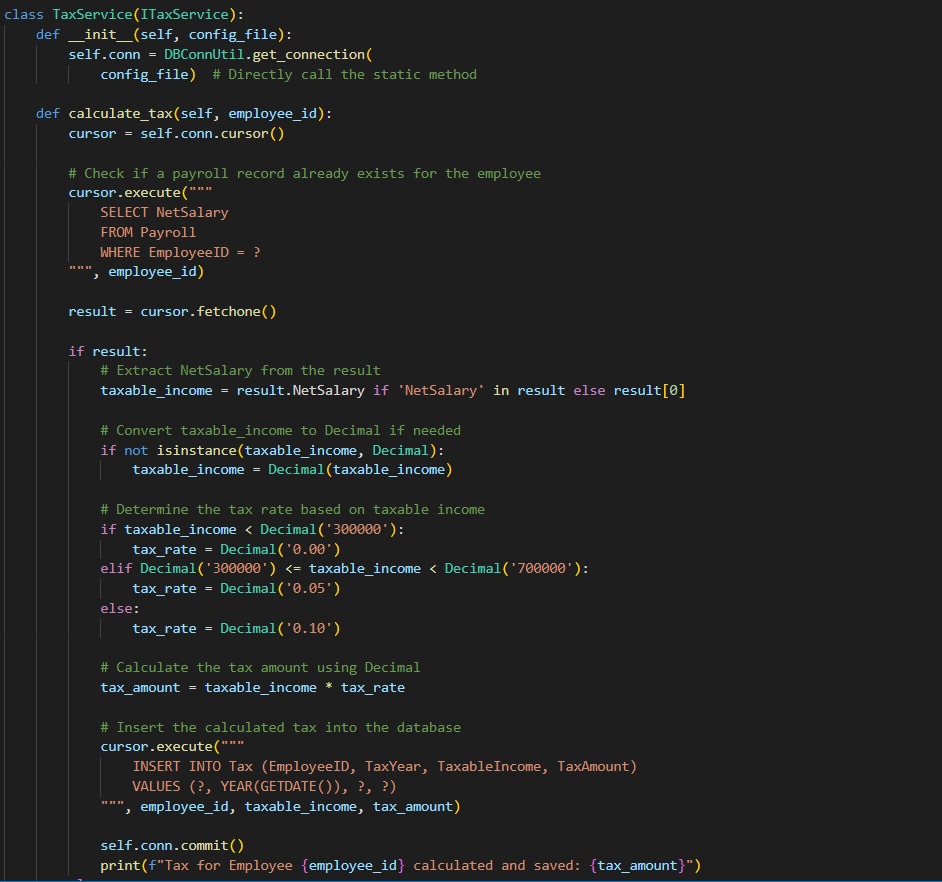
* **ITaxService:**



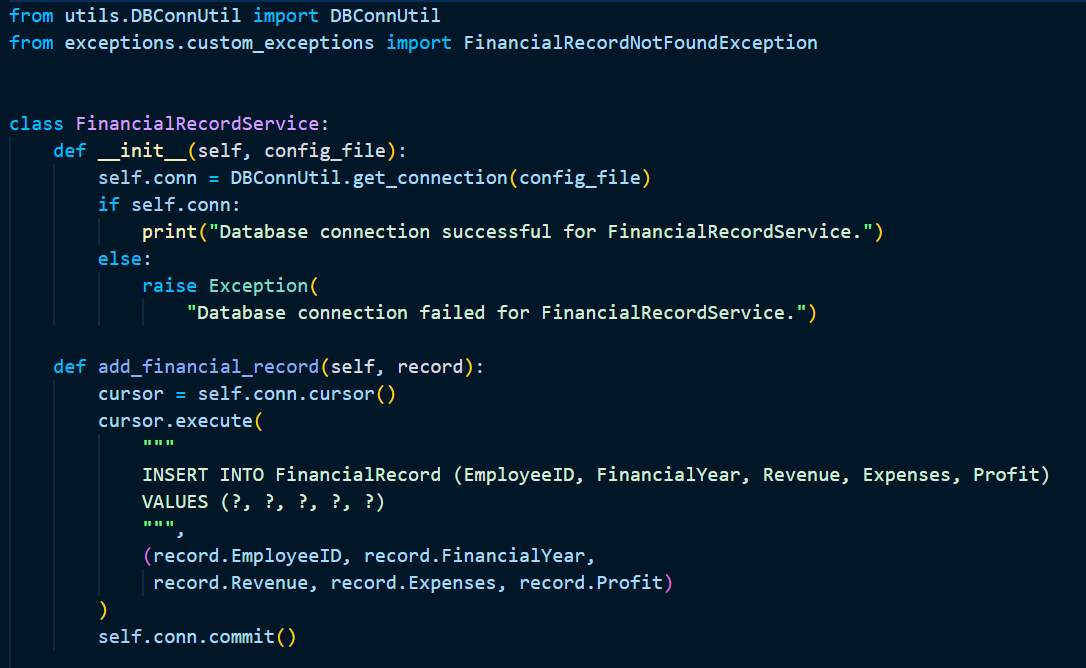


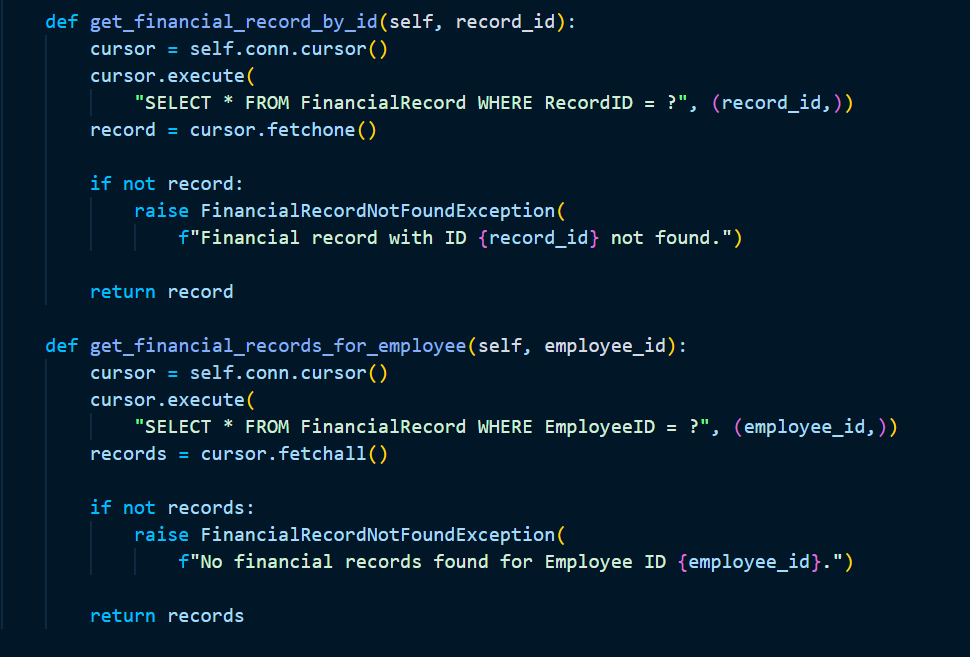


* **TaxService:**



* **FinancialRecordService** (implements IFinancialRecordService): Methods: AddFinancialRecord, GetFinancialRecordById, GetFinancialRecordsForEmployee, GetFinancialRecordsForDate



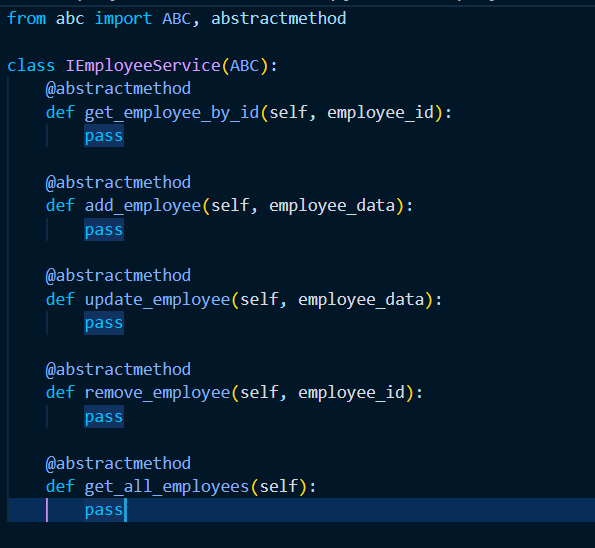


**Task 3: Service Interfaces and Implementations**

In this task, we defined service interfaces in the **dao** package to encapsulate the business logic for interacting with employee and payroll data. The interfaces, such as IEmployeeService and IPayrollService, outline the essential methods for CRUD operations and payroll calculations.

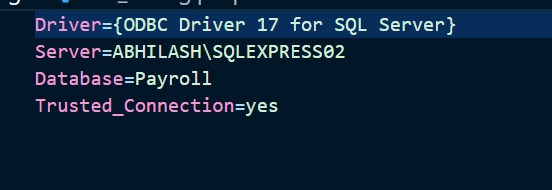
* Each interface specifies methods like GetEmployeeById, AddEmployee, and GeneratePayroll.
* We then created implementation classes for these interfaces that contain the actual logic for interacting with the MSSQL database using Python's pyodbc library.

This design ensures a clear separation of concerns, allowing us to manage database interactions independently of the business logic, which enhances maintainability and testability.

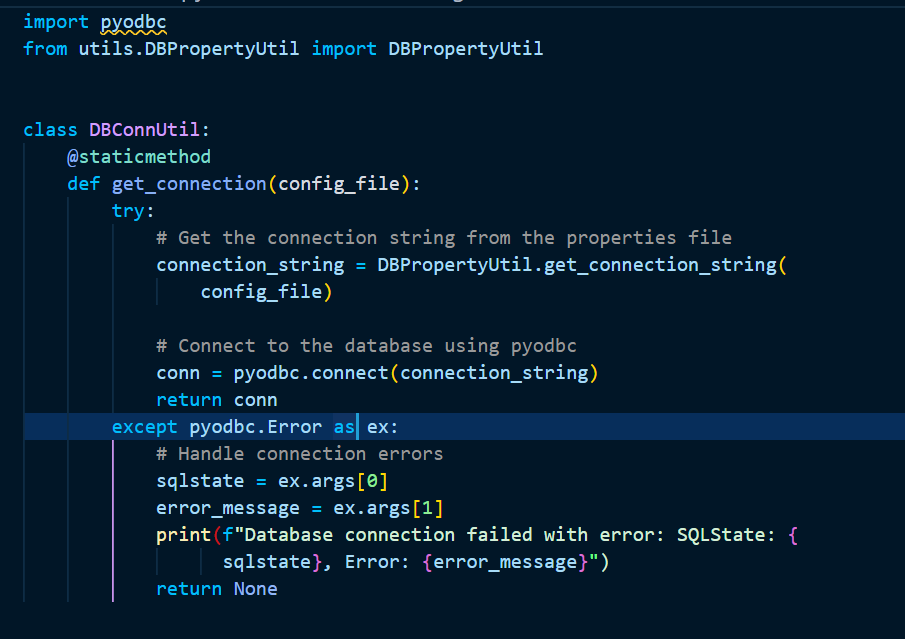


**Task 4: Database Connectivity**We established a connection to the MSSQL database using a **configuration file** that stores the connection details, including the server name, database name, and authentication credentials.

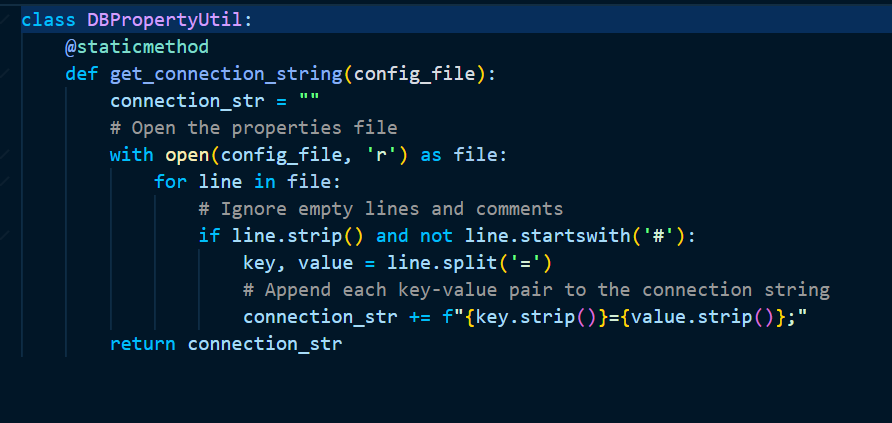
* The connection is managed using the **pyodbc** library in Python.
* A utility class, **DBConnUtil,** is created to handle the database connection logic, facilitating the establishment and closure of connections.
* The **DBPropertyUtil** class is utilized to read the configuration settings from the config file, streamlining the connection process.  
  This design allows the application to perform essential data operations efficiently, such as fetching employee records, inserting payroll data, and updating tax information.
* **configuration file**



* **DBConnUtil**



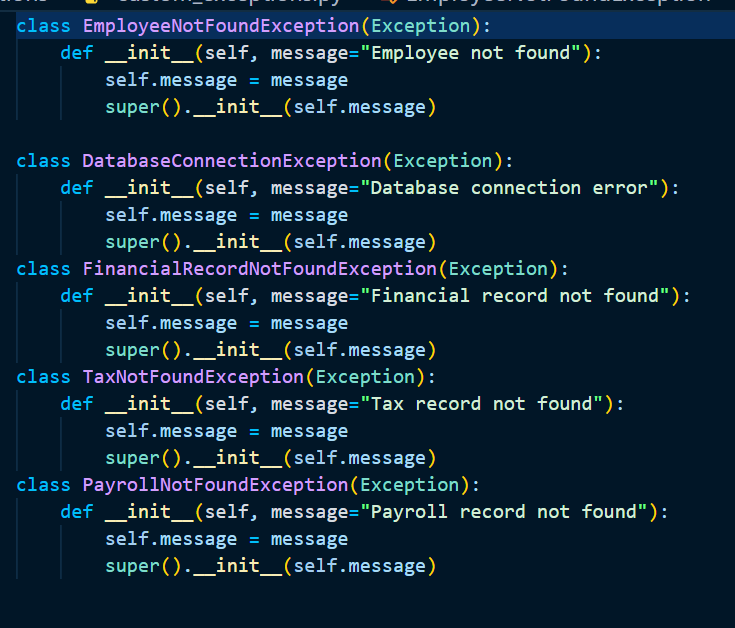
* **DBPropertyUtil**



**Task 5: Custom Exceptions**

We implemented custom exceptions to handle specific error scenarios in the application. These include:

* **EmployeeNotFoundException**: Thrown when an attempt is made to access an employee that does not exist in the database.
* **PayrollGenerationException**: Raised when there are issues with payroll calculations, such as missing employee data.
* **TaxCalculationException**: Triggered when there are errors in tax calculations for an employee.
* **FinancialRecordException**: Used for issues related to financial records, such as adding or retrieving records.
* **InvalidInputException**: Thrown when the provided input data does not meet the expected criteria.
* **DatabaseConnectionException**: Raised when there are problems establishing or maintaining a connection to the database.



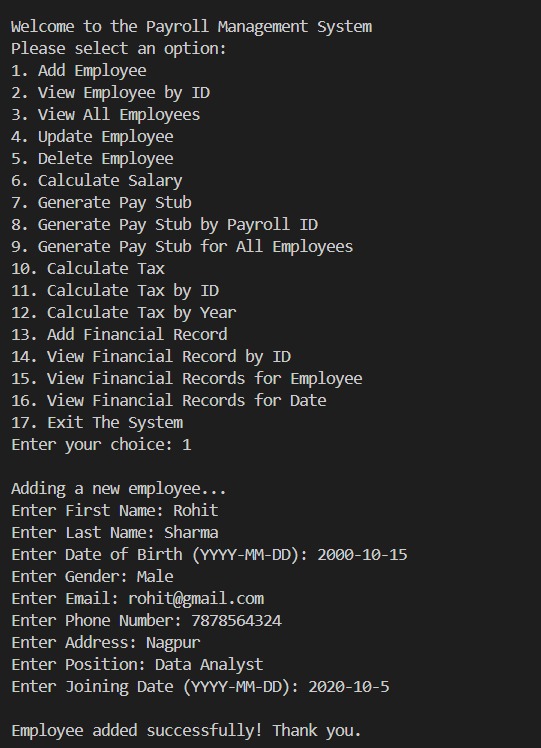
**Task 6: Functionality Demonstration**

In this task, we developed the main module of the application, which presents a menu-driven interface to users. Through this interface, users can perform various operations such as:

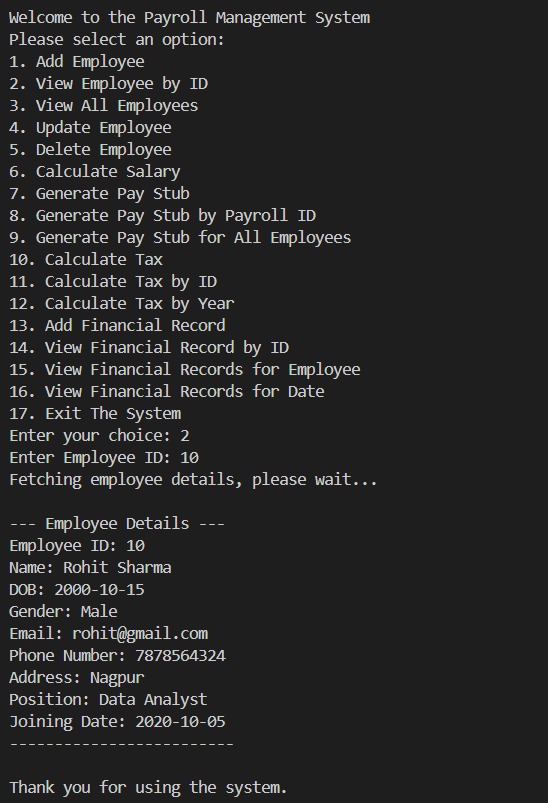
* Adding new employee records
* Generating payroll for a specific pay period
* Calculating taxes for an employee
* Viewing financial records

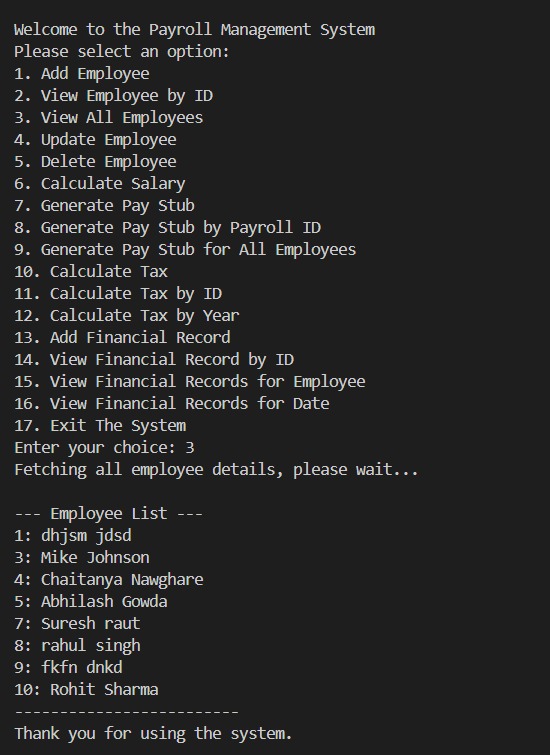
The menu-driven approach enhances user experience, allowing for easy navigation and interaction with the application's functionalities.

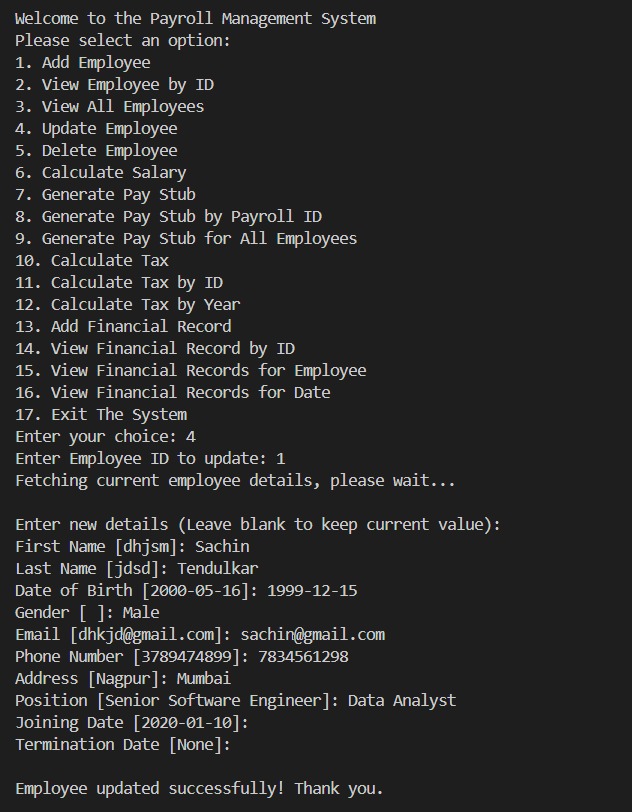
* **Adding new employee records**



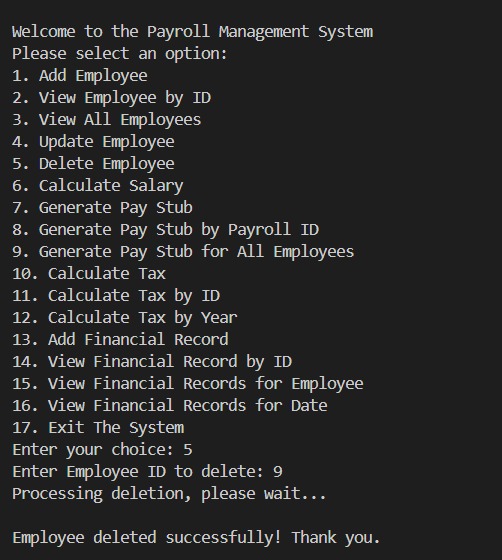
* **View Employee by ID**



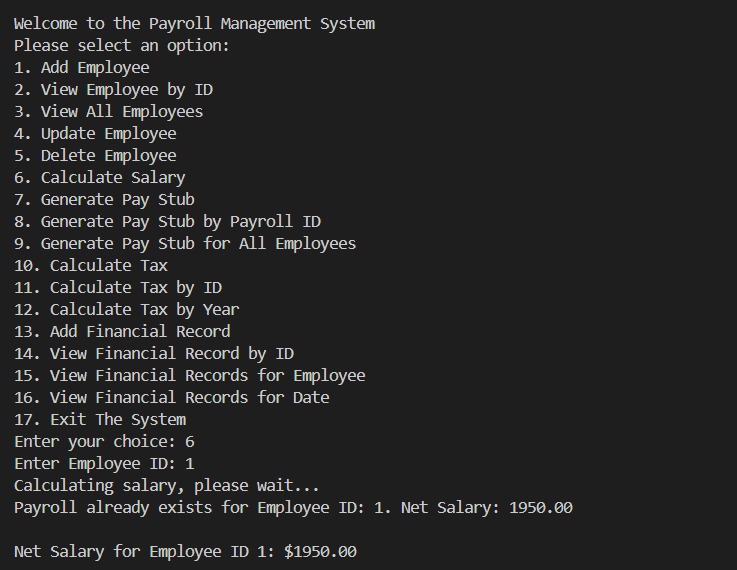
* **View all Employee**
* **Update Employee**



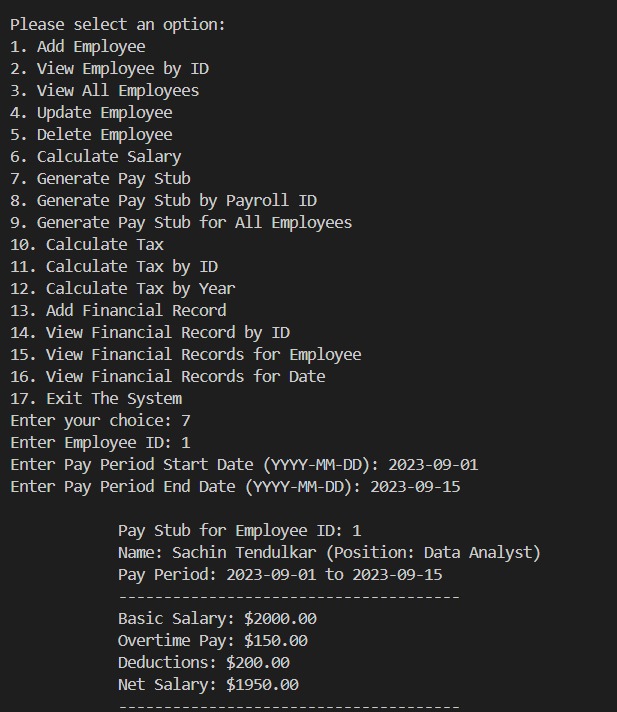
* **Delete Employee**



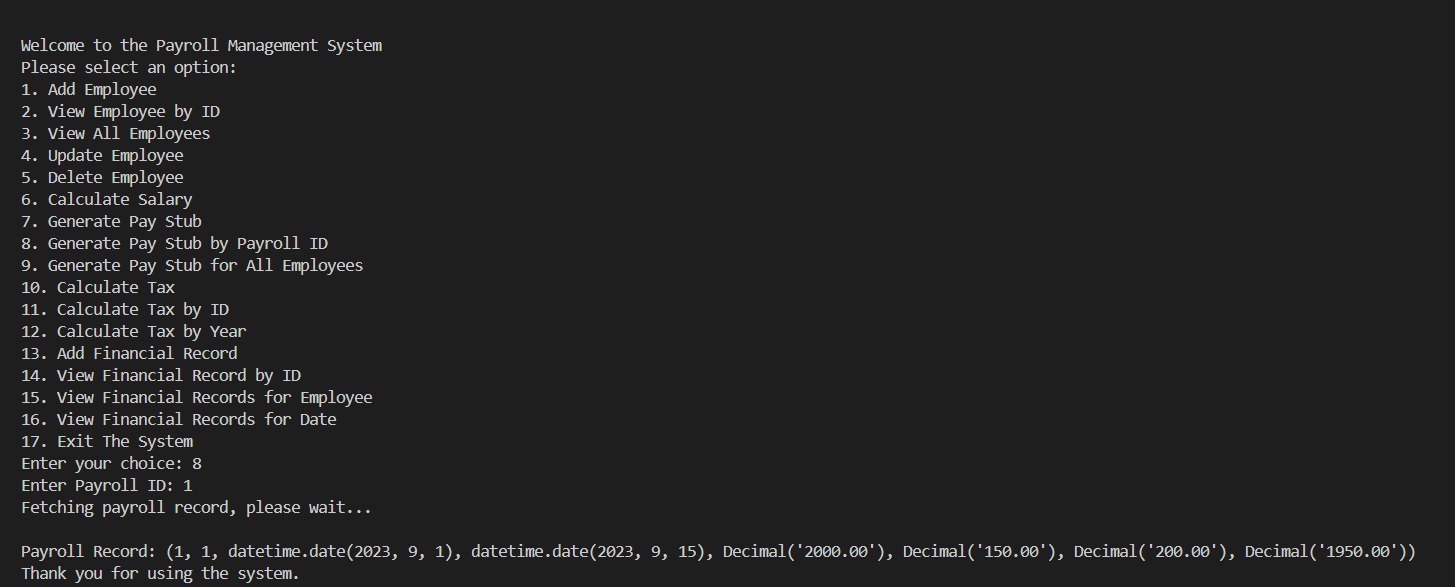
* **Calculate Salary**



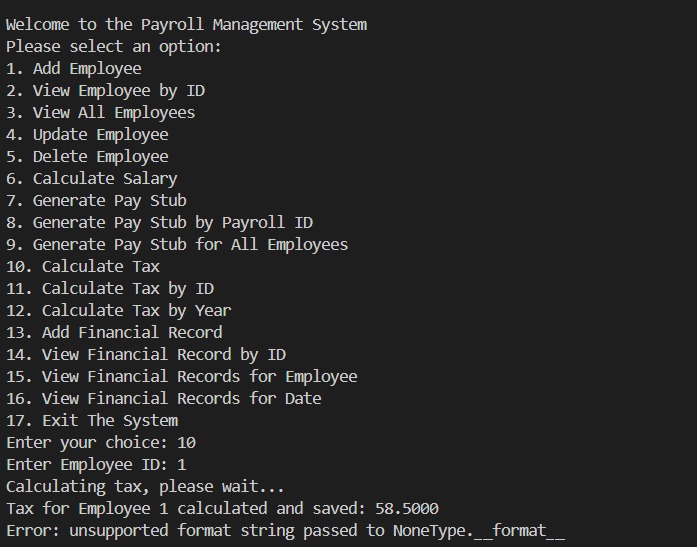
* **Generate Pay Stub**



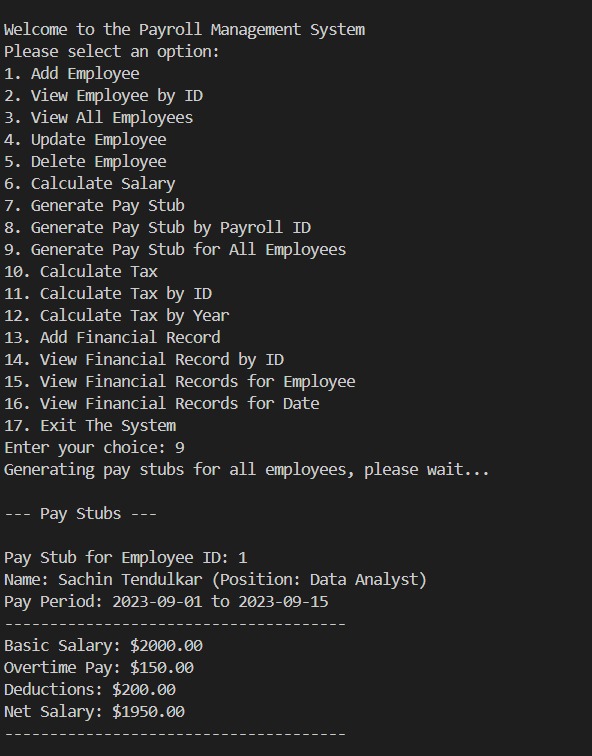
* **Generate Pay Stub by Payroll ID**

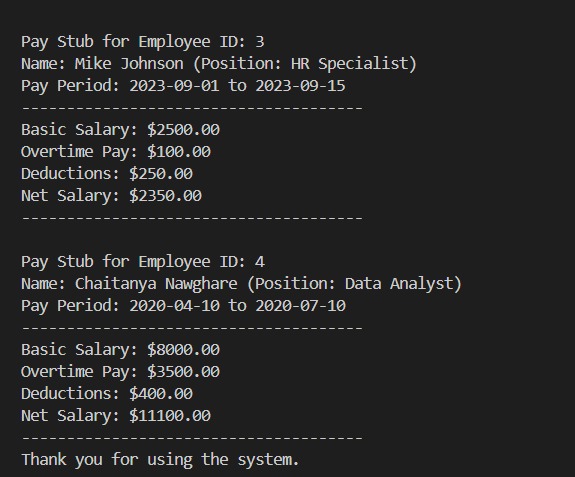


* **Calculating the Tax**

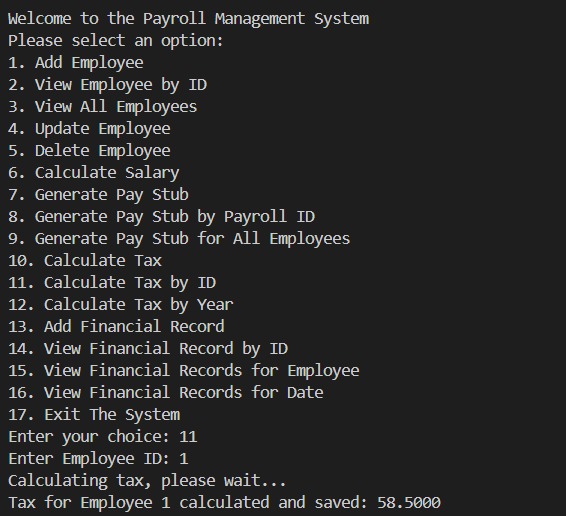


* **Generate Pay Stub for All Employees**

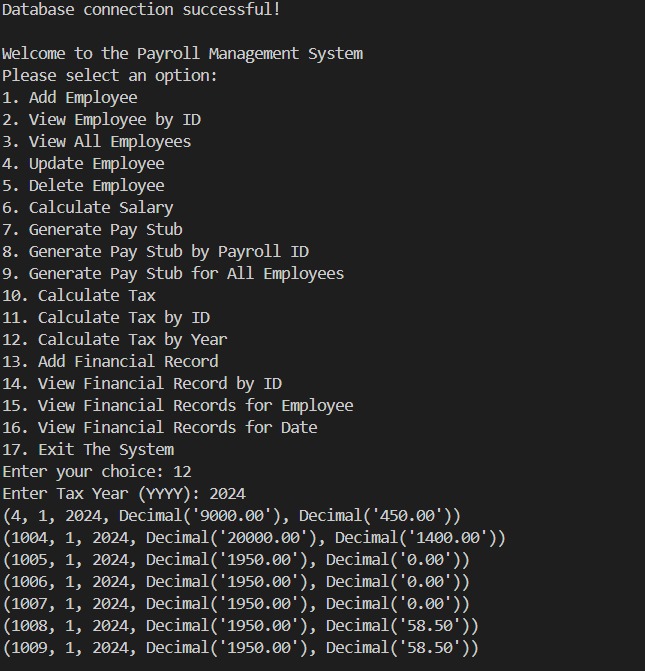




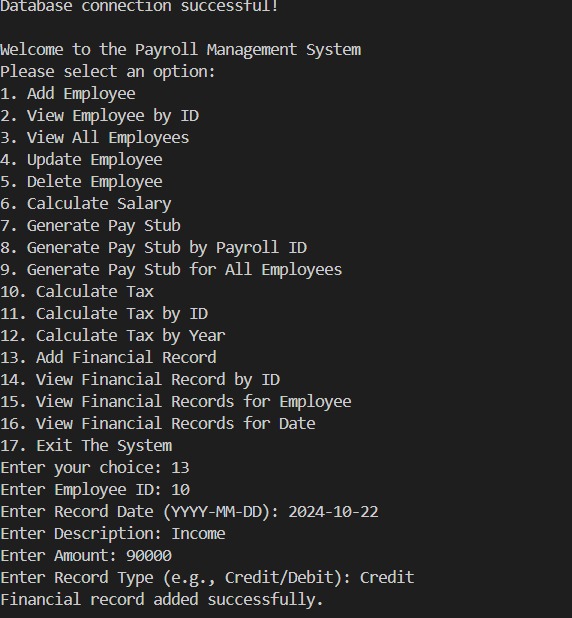
* **Calculate Tax by ID**



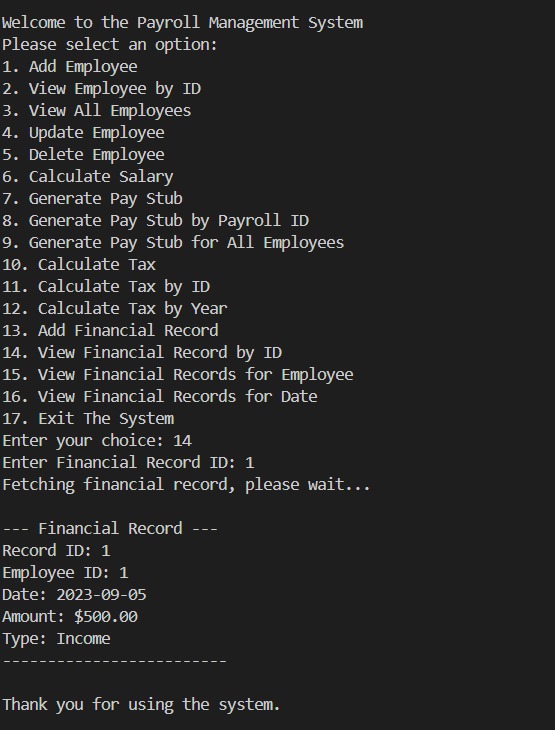
* **Calculate Tax by Year**



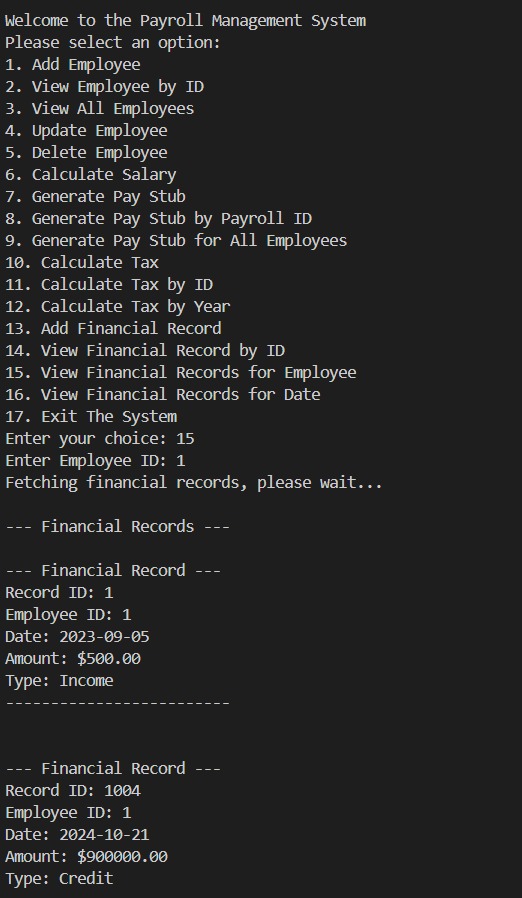
* **Add Financial Record**



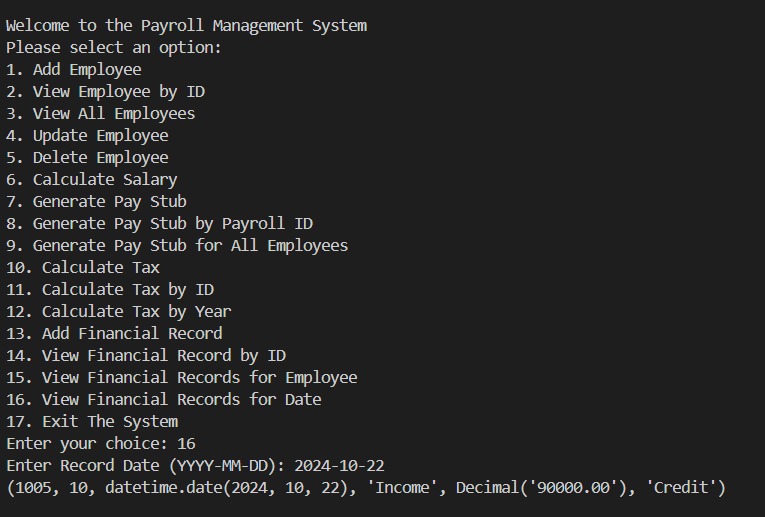
* **View Financial Record by ID**



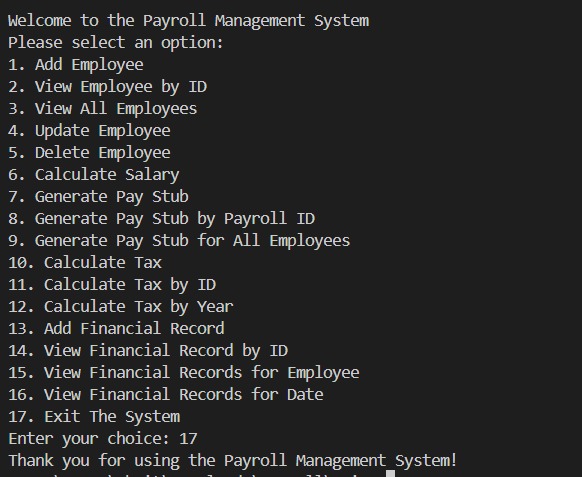
* **View Financial Records for Employee**



* **View Financial Records for Date**



* **Exit The System**



**Task 5: Unit Testing**

Create NUnit test cases for car rental System are essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of NUnit test cases for various components of the system:

* **Test Case**: CalculateGrossSalaryForEmployee

Objective: Verify that the system correctly calculates the gross salary for an employee

* **Test Case**: CalculateNetSalaryAfterDeductions

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

* **Test Case:** VerifyTaxCalculationForHighIncomeEmployee • Objective: Test the system's ability to calculate taxes for a high-income employee.
* **Test Case**: ProcessPayrollForMultipleEmployees •

Objective: Test the end-to-end payroll processing for a batch of employees.

* **Test Case:** VerifyErrorHandlingForInvalidEmployeeData

Objective: Ensure the system handles invalid input data gracefully

