|  |
| --- |
| Student: Hoang Nhan Duyen Nguyen  Student ID: 1911876  Date: December 13, 2020 |

|  |
| --- |
|  |
| **Hi-Tech Order Management System** |
| Course Number: 420-DA3-AS  Course Title: Multi-tier Applications Development  Session: Autumn 2020  Group Number: 7234  Teacher: Quang Hoang Cao |

Table of Contents

[1. Project Description 1](#_Toc58860875)

[2. Project Design 1](#_Toc58860876)

[3. Project Implementation 15](#_Toc58860877)

[Example of a class in Business Layer 15](#_Toc58860878)

[Example of a class in Data Access Layer 15](#_Toc58860879)

[Windows Forms App: Hi-Tech Order Management System 16](#_Toc58860880)

[4. Project Testing 16](#_Toc58860881)

[5. Conclusion 16](#_Toc58860882)

# Project Description

Hi-Tech Order Management System is a Windows application built by using Microsoft Visual Studio 2019, C#, and SQL Server 2017 for Hi-Tech Distribution Inc. which supplies computer science books to colleges and universities in Quebec. The application allows users to add, update, manage… user, employee, customer, book, and orders information.

# 2. Project Design

**- Database Design**

**Table Name:** Categories (related to table Books)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***CategoryId*** | number | PK |
| CategoryName | string |  |

**Table Name:** Publishers (related to table Books)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***PublisherId*** | number | PK |
| PublisherName | string |  |
| WebAddress | string |  |

**Table Name:** Authors (related to table AuthorsBooks)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***AuthorId*** | number | PK |
| FirstName | string |  |
| LastName | string |  |
| Email | string |  |

**Table Name:** AuthorsBooks (related to table Books and table Authors)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***AuthorId*** | number | PK FK |
| ***ISBN*** | string | PK FK |
| YearPublished | string |  |
| Edition | string |  |

**Table Name:** Statues (related to table Books, Customers and Orders)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***Id*** | number | PK |
| Description | string |  |

**Table Name:** Books (related to Categories, Publishers, Statues, OrderLines, AuthorsBooks)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***ISBN*** | string | PK |
| BookTitle | string |  |
| UnitPrice | number |  |
| QOH | number |  |
| PublisherId | number | FK |
| CategoryId | number | FK |
| Status | number | FK |

**Table Name:** OrderLines (related to table Books and Orders)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***OrderId*** | number | PK FK |
| ***ISBN*** | string | PK FK |
| Quantity Ordered | number |  |

**Table Name:** Orders (related to table Customers, Employees, Statues, and OrderLines)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***OrderId*** | number | PK |
| CustomerId | number | FK |
| EmployeeId | number | FK |
| OrderType | string |  |
| OrderDate | date |  |
| RequiredDate | date |  |
| ShippingDate | date |  |
| OrderStatus | number |  |

**Table Name:** Customers (related to table Statues and Orders)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***CustomerId*** | number | PK |
| CustomerName | string |  |
| StreetName | string |  |
| Province | string |  |
| City | string |  |
| PostalCode | string |  |
| PhoneNumber | string |  |
| ContactName | string |  |
| ContactEmail | string |  |
| CreditLimit | number |  |
| Status | string | FK |

**Table Name:** Employees (related to table Jobs, UserAccounts and Orders)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***EmployeeId*** | number | PK |
| LastName | string |  |
| FirstName | string |  |
| PhoneNumber | string |  |
| Email | string |  |
| JobId | number | FK |

**Table Name:** UserAccounts (related to table Employees)

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***UserId*** | number | PK |
| Password | string |  |
| *EmployeeId* | number | FK |

**Table Name:** Jobs (related to table Employees)

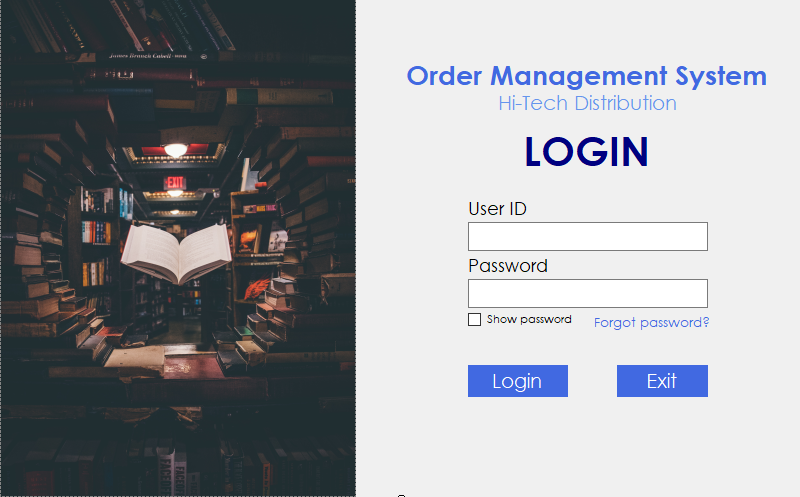
|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Design Note |
| ***JobId*** | number | PK |
| JobTitle | string |  |

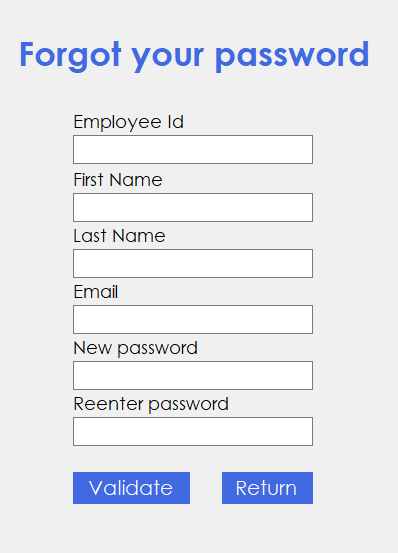
**- Entity Relationship Diagram**



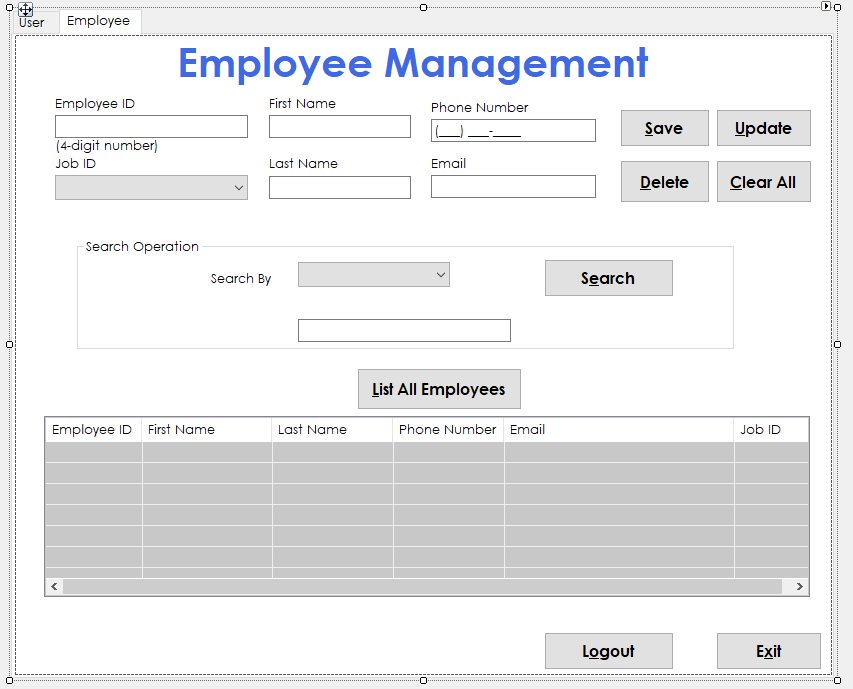
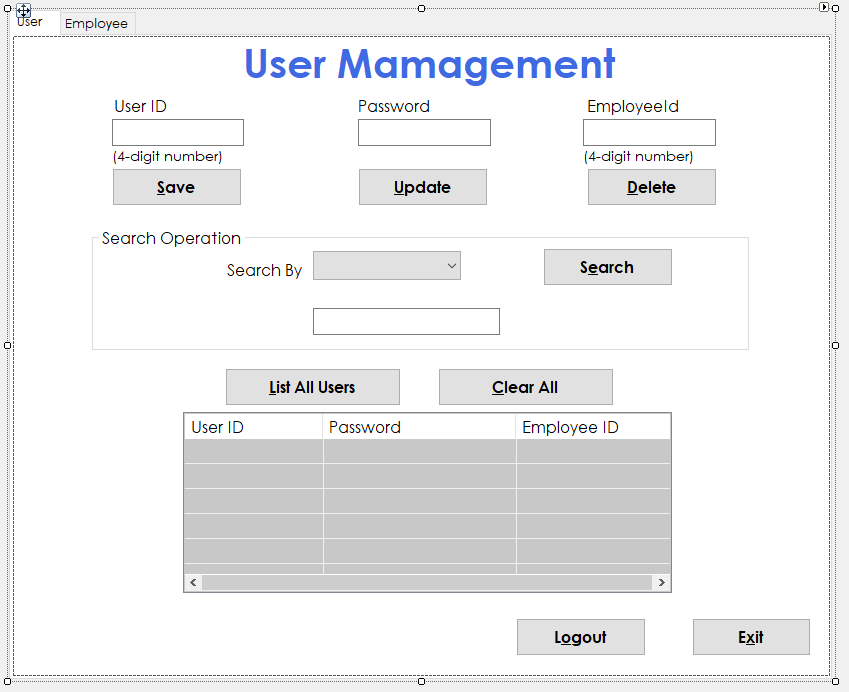
**- Design of GUI Classes**

*Login/ Forgot password forms*



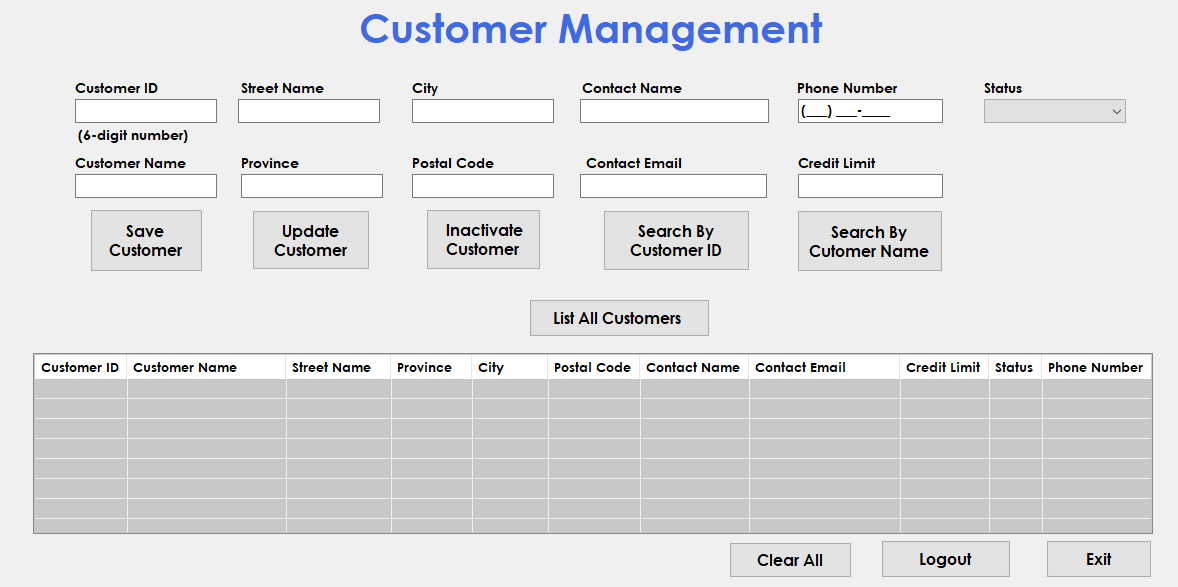


*Module 1: MIS Manager – User/Employee Management (Database Programming with*

*ADO.NET in Connected Mode)***

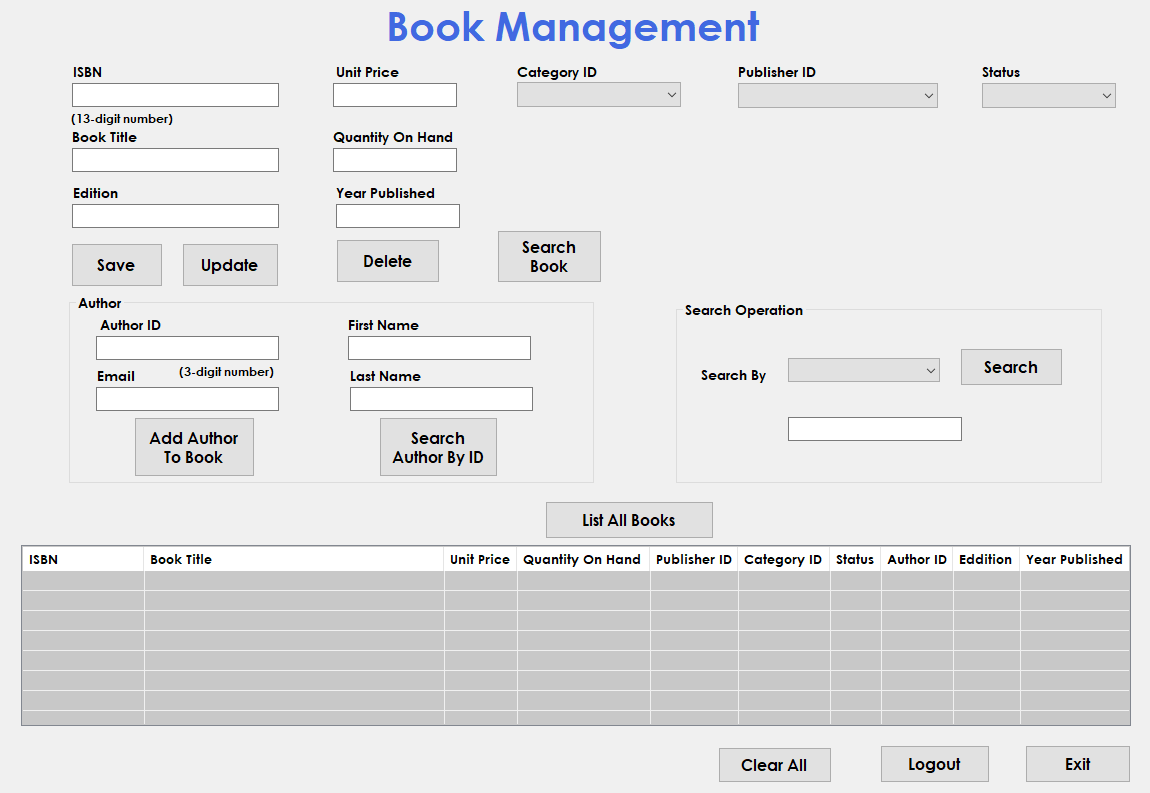
*Module 2: Sales Manager – Customer Management (Database Programming with*

*ADO.NET in Disconnected Mode)*

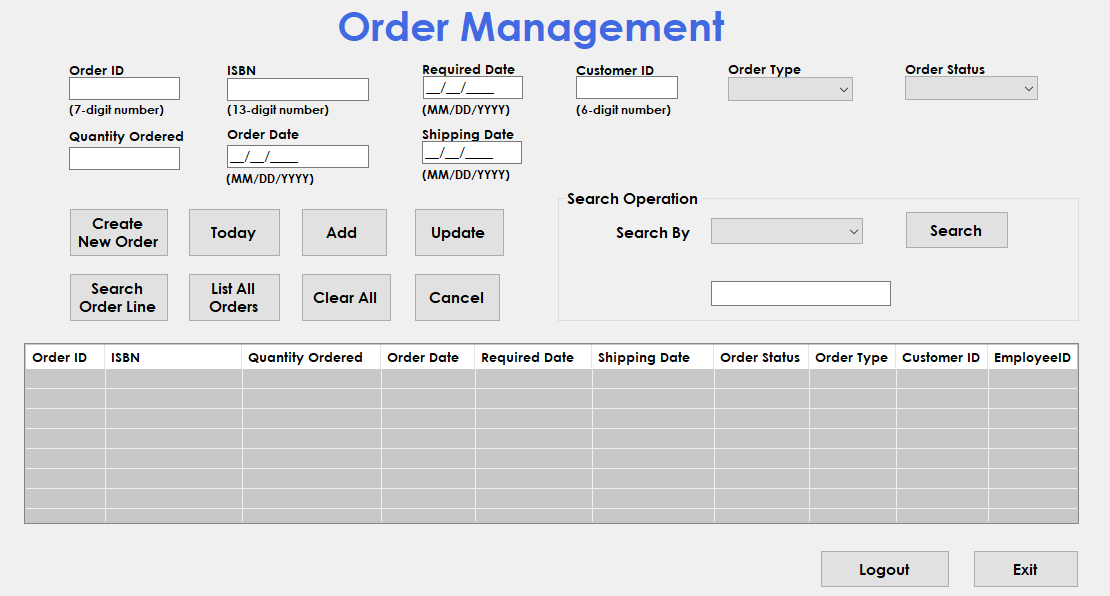


*Module 3: Inventory Controller – Book Management (Database Programming with*

*ADO.NET in Connected Mode)*

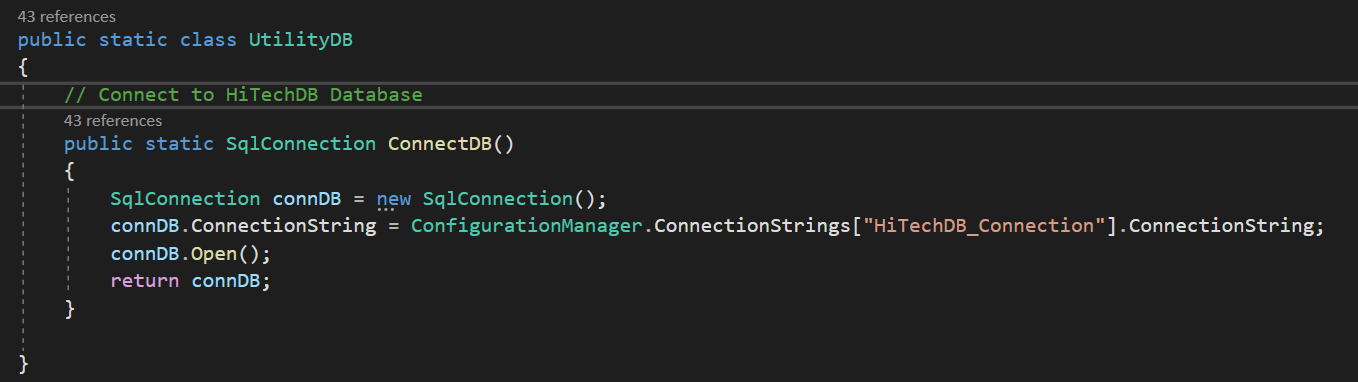


*Module 4: Order Clerks – Order Management (Entity Framework)*



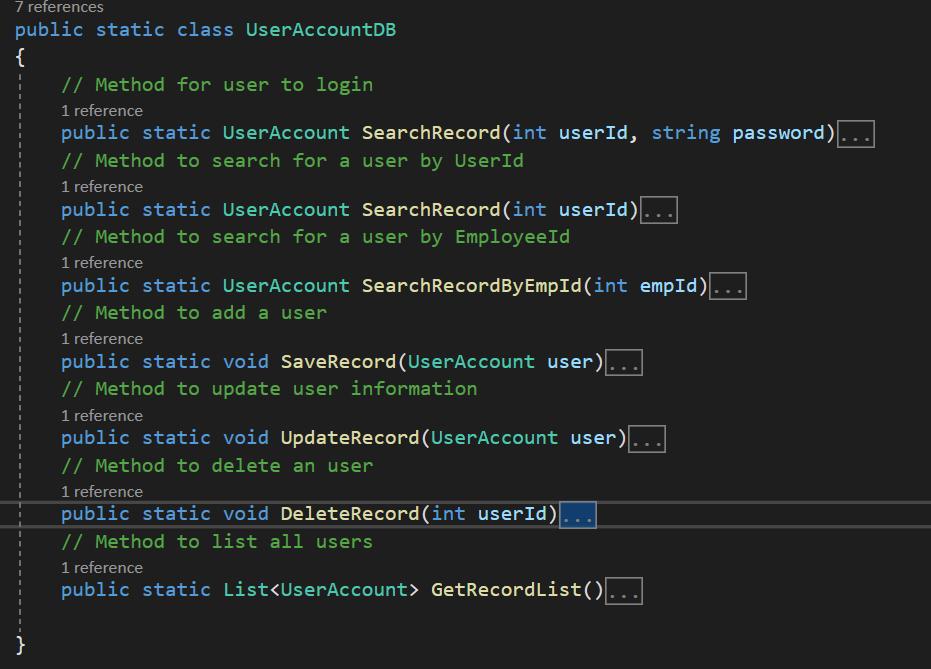
- Design of Data Access Classes (Hi-Tech Library)

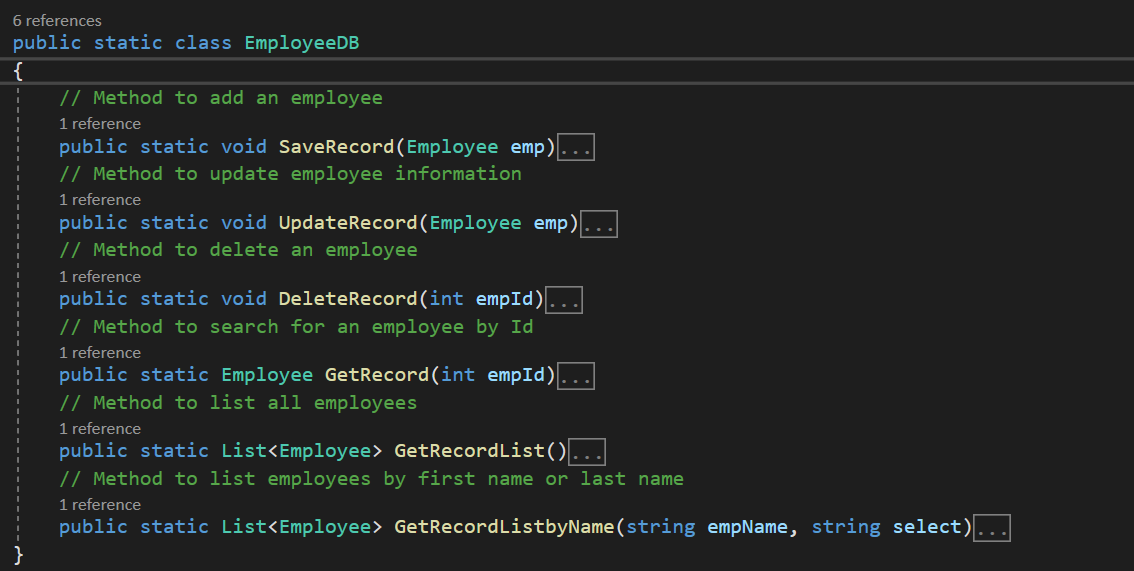
**UtilityDB.cs**

**

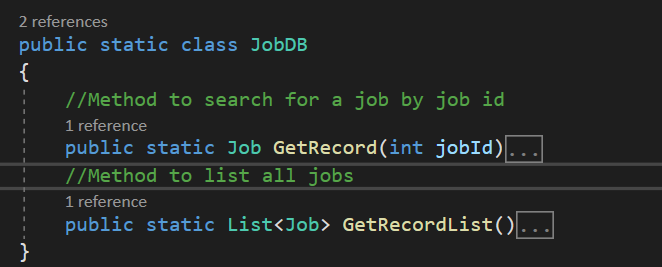
*Module 1: MIS Manager – User/Employee Management*

**UserAccountDB.cs**



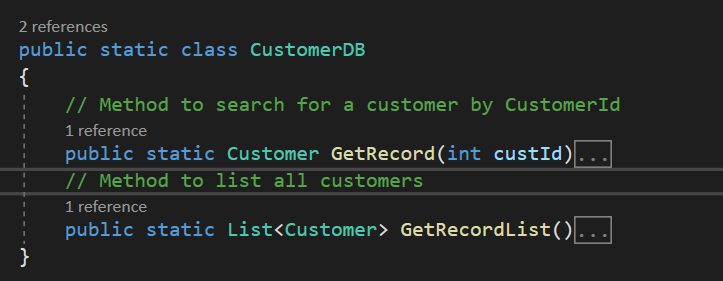
**EmployeeDB.cs**

**JobDB.cs**

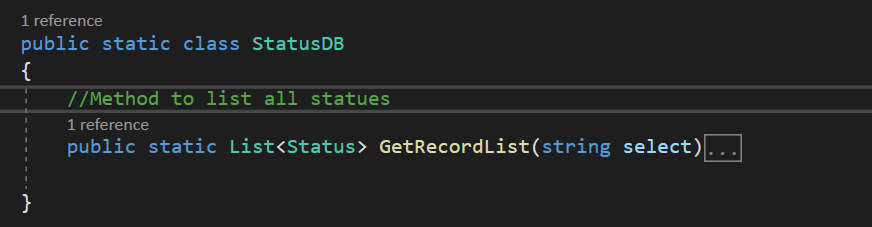


*Module 2: Sales Manager – Customer Management*

**CustomerDB.cs**

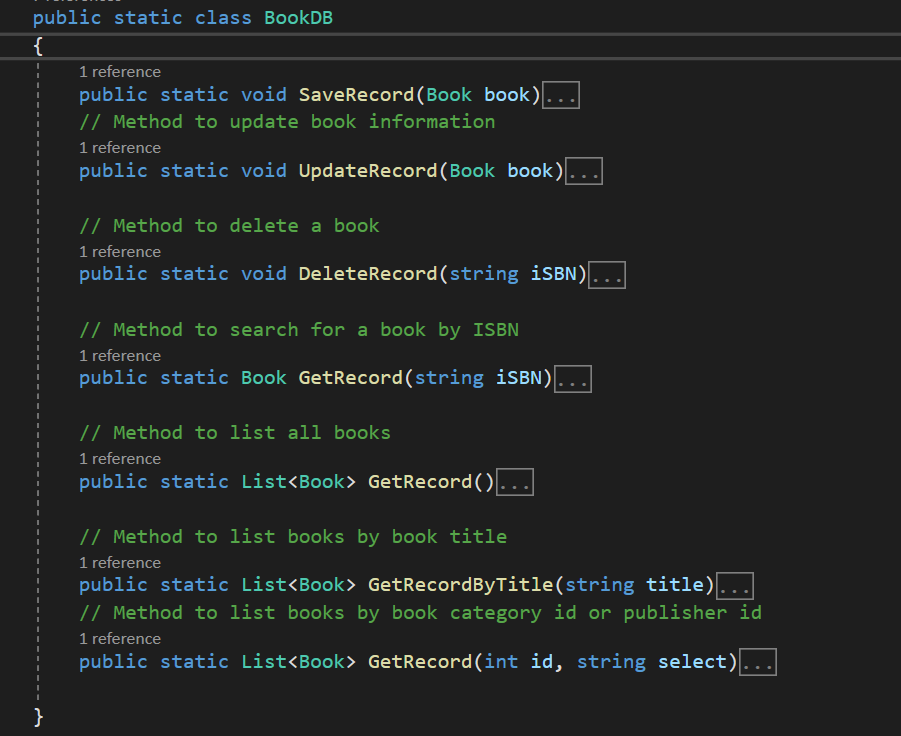


**StatusDB.cs**

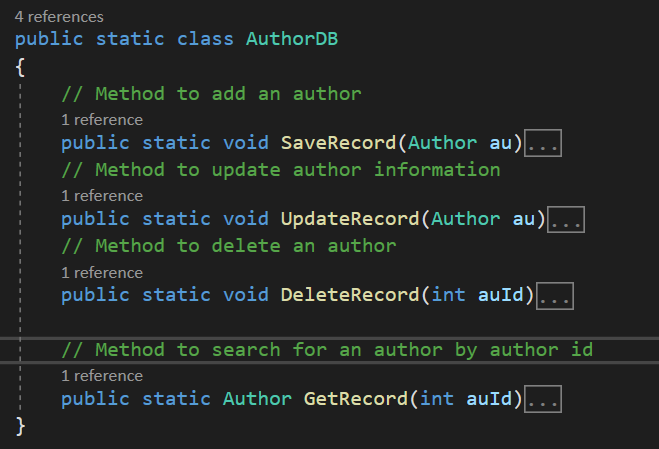
**

*Module 3: Inventory Controller – Book Management*

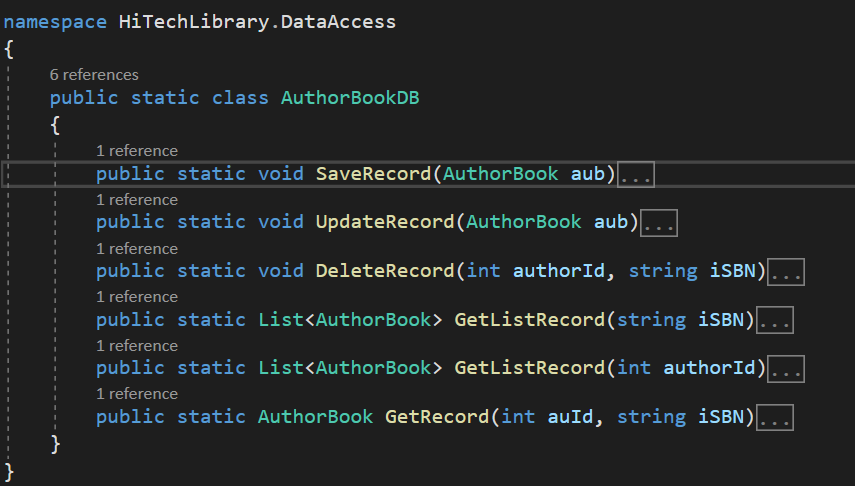
**BookDB.cs**



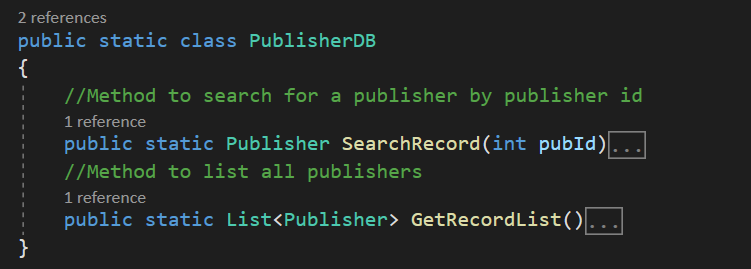
**AuthorDB.cs**

****

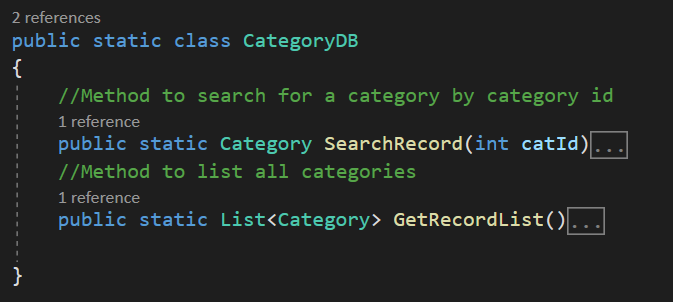
**AuthorBookDB.cs**

****

**PublisherDB.cs**

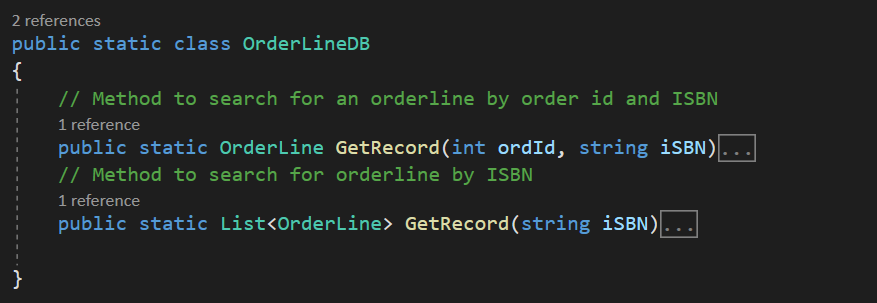
**

**CategoryDB.cs**

**

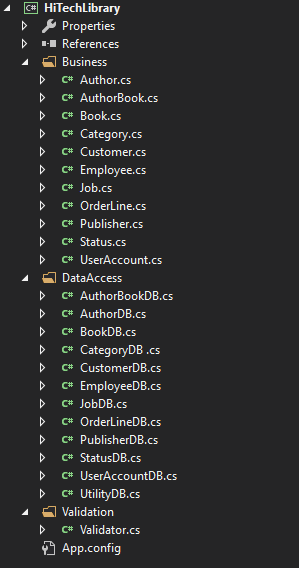
*Module 4: Order Clerks – Order Management*

**OrderLineDB.cs**



# 3. Project Implementation

Library: HiTechLibrary



## Example of a class in Business Layer

*Employee.cs*

namespace HiTechLibrary.Business

{

public class Employee

{

private int employeeId;

private string firstName;

private string lastName;

private string phoneNumber;

private string email;

private int jobId;

public int EmployeeId { get => employeeId; set => employeeId = value; }

public string FirstName { get => firstName; set => firstName = value; }

public string LastName { get => lastName; set => lastName = value; }

public string PhoneNumber { get => phoneNumber; set => phoneNumber = value; }

public string Email { get => email; set => email = value; }

public int JobId { get => jobId; set => jobId = value; }

// default constructor (constructor has to be the class name)

public Employee()

{

employeeId = 0;

firstName = "";

lastName = "";

phoneNumber = "";

email = "";

}

//overloaded constructor ( parameterized constructor)

public Employee(int employeeId, string firstName, string lastName, string phoneNumber, string email, int jobId)

{ //this is the CLASS variable

this.employeeId = employeeId;

this.firstName = firstName;

this.lastName = lastName;

this.phoneNumber = phoneNumber;

this.email = email;

this.jobId = jobId;

}

public void SaveEmployee(Employee emp)

{

EmployeeDB.SaveRecord(emp);

}

public void UpdateEmployee(Employee emp)

{

EmployeeDB.UpdateRecord(emp);

}

public void DeleteEmployee(int empId)

{

EmployeeDB.DeleteRecord(empId);

}

public Employee SearchEmployee(int empId)

{

return EmployeeDB.GetRecord(empId);

}

public List<Employee> SearchEmployeeByName(string name, string select)

{

return EmployeeDB.GetRecordListbyName(name, select);

}

public List<Employee> SearchAllEmployee()

{

return EmployeeDB.GetRecordList();

}

}

}

## Example of a class in Data Access Layer

*EmployeeDB.cs*

// Method to add an employee

public static void SaveRecord(Employee emp)

{

// Step 1: Connect the DB

SqlConnection connDB = UtilityDB.ConnectDB();

// Step 2: Perform Insert operation

SqlCommand cmdInsert = new SqlCommand("INSERT INTO Employees(EmployeeId,FirstName,LastName,PhoneNumber,Email,JobId) VALUES(@EmployeeId,@FirstName,@LastName,@PhoneNumber,@Email,@JobId);", connDB);

cmdInsert.Parameters.AddWithValue("@EmployeeId", emp.EmployeeId);

cmdInsert.Parameters.AddWithValue("@FirstName", emp.FirstName);

cmdInsert.Parameters.AddWithValue("@LastName", emp.LastName);

cmdInsert.Parameters.AddWithValue("@PhoneNumber", emp.PhoneNumber);

cmdInsert.Parameters.AddWithValue("@Email", emp.Email);

cmdInsert.Parameters.AddWithValue("@JobId", emp.JobId);

cmdInsert.ExecuteNonQuery();

// Step 3: Close the DB

connDB.Close();

}

// Method to update employee information

public static void UpdateRecord(Employee emp)

{

//step 1: Connect the DB

SqlConnection connDB = UtilityDB.ConnectDB();

//step 2: Perform Update operation

SqlCommand cmdUpdate = new SqlCommand("UPDATE Employees SET FirstName = @FirstName, LastName = @LastName,PhoneNumber = @PhoneNumber, Email = @Email, JobId = @JobId WHERE EmployeeId = @EmployeeId", connDB);

cmdUpdate.Parameters.AddWithValue("@EmployeeId", emp.EmployeeId);

cmdUpdate.Parameters.AddWithValue("@FirstName", emp.FirstName);

cmdUpdate.Parameters.AddWithValue("@LastName", emp.LastName);

cmdUpdate.Parameters.AddWithValue("@PhoneNumber", emp.PhoneNumber);

cmdUpdate.Parameters.AddWithValue("@Email", emp.Email);

cmdUpdate.Parameters.AddWithValue("@JobId", emp.JobId);

cmdUpdate.ExecuteNonQuery();

//step 3: Close DB

connDB.Close();

}

// Method to delete an employee

public static void DeleteRecord(int empId)

{

//step 1: Connect the DB

SqlConnection connDB = UtilityDB.ConnectDB();

//step 2: Perform Delete operation

SqlCommand cmdDelete = new SqlCommand("DELETE FROM Employees WHERE EmployeeId = @EmployeeId", connDB);

cmdDelete.Parameters.AddWithValue("@EmployeeId", empId);

cmdDelete.ExecuteNonQuery();

//step 3: Close DB

connDB.Close();

}

// Method to search for an employee by Id

public static Employee GetRecord(int empId)

{

// Step 1: Connect the Database

SqlConnection connDB = UtilityDB.ConnectDB();

//Step 2: Perform Search operation

SqlCommand cmdSelect = new SqlCommand("SELECT \* FROM Employees WHERE EmployeeId = @empId", connDB);

cmdSelect.Parameters.AddWithValue("@empId", empId);

SqlDataReader sqlReader = cmdSelect.ExecuteReader();

Employee emp = new Employee();

if (sqlReader.Read())

{

emp.EmployeeId = Convert.ToInt32(sqlReader["EmployeeId"]);

emp.FirstName = sqlReader["FirstName"].ToString();

emp.LastName = sqlReader["LastName"].ToString();

emp.PhoneNumber = sqlReader["PhoneNumber"].ToString();

emp.Email = sqlReader["Email"].ToString();

emp.JobId = Convert.ToInt32(sqlReader["JobId"]);

}

else

{

emp = null;

}

// Step 3: Close the database

connDB.Close();

return emp;

}

// Method to list all employees

public static List<Employee> GetRecordList()

{

List<Employee> listEmp = new List<Employee>();

// Step 1: Connect the Database

SqlConnection connDB = UtilityDB.ConnectDB();

// Step 2: Perform Select all operation

SqlCommand cmdSelectAll = new SqlCommand("SELECT \* FROM Employees", connDB);

SqlDataReader sqlReader = cmdSelectAll.ExecuteReader();

Employee emp;

while (sqlReader.Read())

{

emp = new Employee();

emp.EmployeeId = Convert.ToInt32(sqlReader["EmployeeId"]);

emp.FirstName = sqlReader["FirstName"].ToString();

emp.LastName = sqlReader["LastName"].ToString();

emp.PhoneNumber = sqlReader["PhoneNumber"].ToString();

emp.Email = sqlReader["Email"].ToString();

emp.JobId = Convert.ToInt32(sqlReader["JobId"]);

listEmp.Add(emp);

}

// Step 3: Close the database

connDB.Close();

return listEmp;

}

// Method to list employees by first name or last name

public static List<Employee> GetRecordListbyName(string empName, string select)

{

List<Employee> listEmp = new List<Employee>();

// Step 1: Connect the Database

SqlConnection connDB = UtilityDB.ConnectDB();

// Step 2: Perform Select all operation

SqlCommand cmdSelectName = new SqlCommand("SELECT \* FROM Employees " + "WHERE " + select + " = @Name ", connDB);

cmdSelectName.Parameters.AddWithValue("@Name", empName);

SqlDataReader sqlReader = cmdSelectName.ExecuteReader();

Employee emp;

while (sqlReader.Read())

{

emp = new Employee();

emp.EmployeeId = Convert.ToInt32(sqlReader["EmployeeId"]);

emp.FirstName = sqlReader["FirstName"].ToString();

emp.LastName = sqlReader["LastName"].ToString();

emp.PhoneNumber = sqlReader["PhoneNumber"].ToString();

emp.Email = sqlReader["Email"].ToString();

emp.JobId = Convert.ToInt32(sqlReader["JobId"]);

listEmp.Add(emp);

}

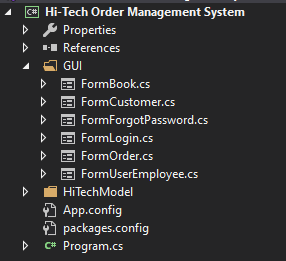
// Step 3: Close the Database

connDB.Close();

return listEmp;

}

## Windows Forms App: Hi-Tech Order Management System



# 4. Project Testing

|  |  |  |
| --- | --- | --- |
| User | Operation | Testing results |
| MIS Manager  (Henry Brown) | * Add/update/delete user * Search/List user information * Add/update/delete employee * Search/list employee information | * Working * Working * Working * Working |
| Sales Manager  (Thomas Moore) | * Add/update/delete customer * Search/List user information | * Working * Working |
| Inventory Controller  (Peter Wang) | * Add/update/delete/search/list book information and related information | * Working |
| Order Clerks  - Mary Brown  - Jennifer Bouchard | * Add/Update/Cancel customers’ orders * Search/List customers’ orders | * Working * Working |

# 5. Conclusion

This project is not simply about creating an application, it also requires us to logically design the database and thinks carefully before starting to write the code. Throughout the project, I get to know what happens behind an application, how we go from designing database, designing the interface… to testing. Generally speaking, I really enjoyed working on this project. This project will be very helpful for the next semester and also in the future.