Livia Ferraz Ximenes,

Event Programming

420-P46-AS

FINAL PROJECT:

**Hi-Tech Management System**

Presented to M. Quang Hoang Cao

LaSalle College

April 18th, 2017

Contents

[Introduction 3](#_Toc480193375)

[1 Project Description 4](#_Toc480193376)

[2 Project Development Process 4](#_Toc480193377)

[2.1 Analysis 4](#_Toc480193378)

[2.2 Design 5](#_Toc480193379)

[2.3 Implementation 13](#_Toc480193380)

[2.4 Testing 18](#_Toc480193381)

[2.5 Deploying the Application 20](#_Toc480193382)

[3 Conclusion 20](#_Toc480193383)

# Introduction

In this document you will find all the pertinent information about the final project from the course Event Programming in LaSalle College. This document is divided in two sections. The first section contains the project description with all the modules created for this application. The second section depicts the project development process, with analysis, design, implementation and testing of this project.

# 1 Project Description

Hi-Tech Management System is a windows-based solution to provide control of four major areas for the Hi-Tech Company. The solution is divided in four modules:

1. MIS Manager Module: allows the MIS manager to save, update, delete, search and list employees;
2. Sales Manager Module: allows the sales manager to save, update, delete, search and list client information;
3. Inventory Module: allows the inventory controller to save, update, delete, search and list product information;
4. Order Module: allows the sales clerks to save, update, delete, search and list sales orders;

The solution also allows the users to change their password at the log in.

# 2 Project Development Process

# 2.1 Analysis

In order to fulfill the requirements of the project, different programming techniques were used to build the application, and they were divided by module. They were:

1. MIS Manager Module: ADO.Net Object Model – Connected Mode
2. Sales Manager Module: ADO.Net Object Model – Disconnected Mode
3. Inventory Module: .Net Entity Framework
4. Order Module: .Net Entity Framework and LINQ

To construct the database for this application, SQL Server 2014 were used, and 11 tables were built to manage all the functionalities necessary.

For the MIS Manager Module, in connected mode, a table Employees was built, to store all pertinent information regarding the employees of HiTech, like name, job title and email.

For the Sales Manager Module, in disconnected mode, a table Clients was build, to store all pertinent information regarding the clients of HiTech, like name, address and credit limit.

For the Inventory Module, using Entity Framework, the following tables were built: Softwares, Books, Authors, AuthorBooks, Categories and Publishers. Those tables were built to allow the sales manager to create and manipulate all data regarding Books and Softwares. The tables Publishers, Categories and Author provide information for the table Books. The table Categories also provide information for the table Softwares.

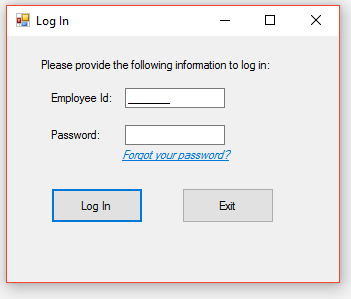
For the Order Module, using Entity Framework and LINQ, the following tables were built: Orders, OrderLineSoftwares and OrderLineBooks were created. Those tables were built to allow the order clerks to create and manipulate orders of books and softwares.

# 2.2 Design

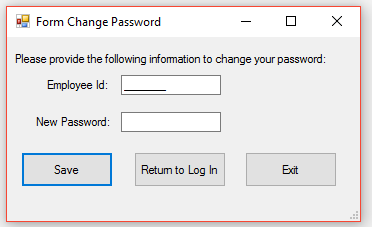
As mentioned before, 11 tables were created in SQL Server to provide all functionalities necessary for this application. Their design, fields and connection to other tables are as follow:



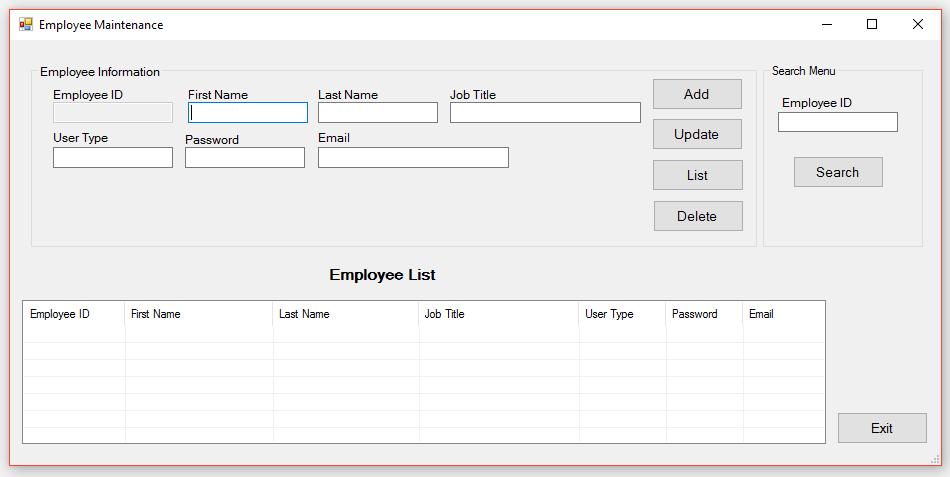
In the GUI, when the user runs the application, he has to log in in order to access the system. The users have restricted access to the application, and are only allowed to access the modules pertinent to their job function.



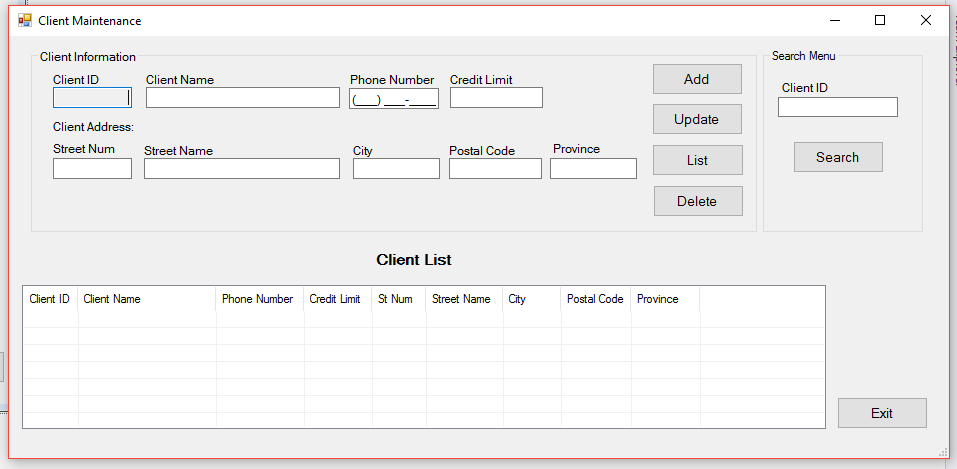
The user has to type their Employee ID and password in order to access the application. If he/she forget his/her password, there is the option “Forgot your password?”, that will lead to another form to create a new password.



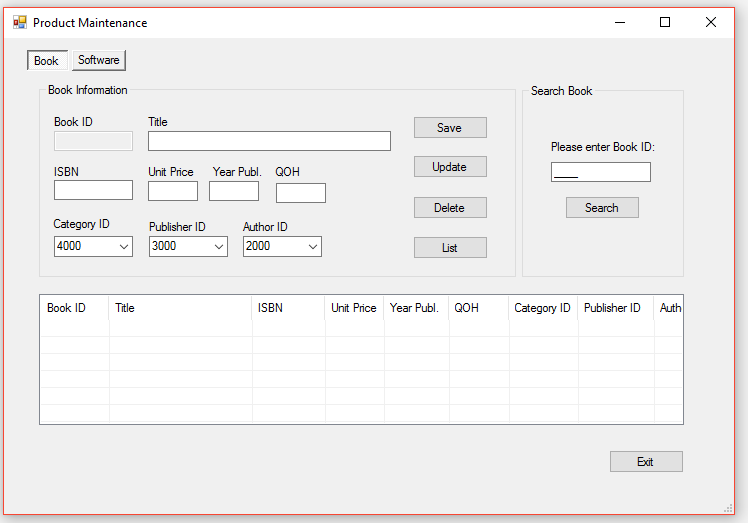
The MIS Manager has access to the Employee Maintenance Form, which allows him to create, update, list, delete and search for an employee.

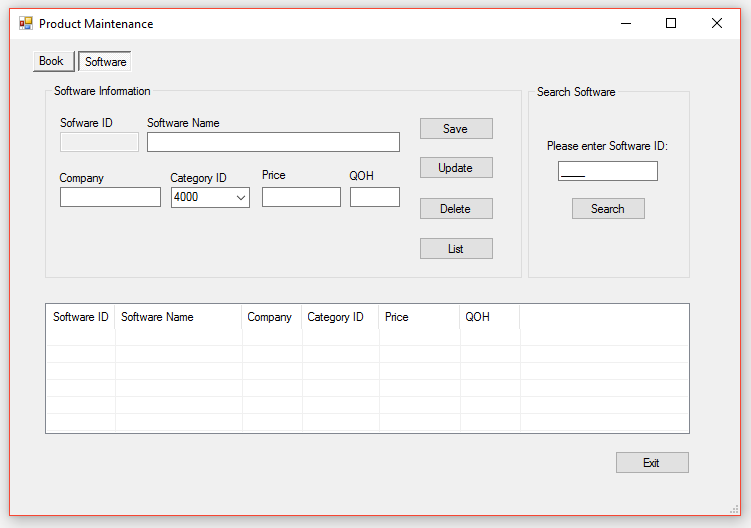


The Sales Manager has access to the Client Maintenance Form, which allows him to create, update, list, delete and search for a client.

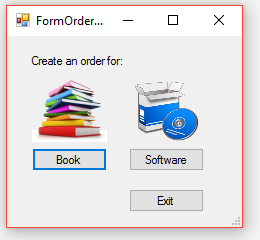


The Inventory Controller has access to the Product Maintenance Form, which allows him to create, update, list, delete and search for a book or software. This form has two tabs to separate books from software.

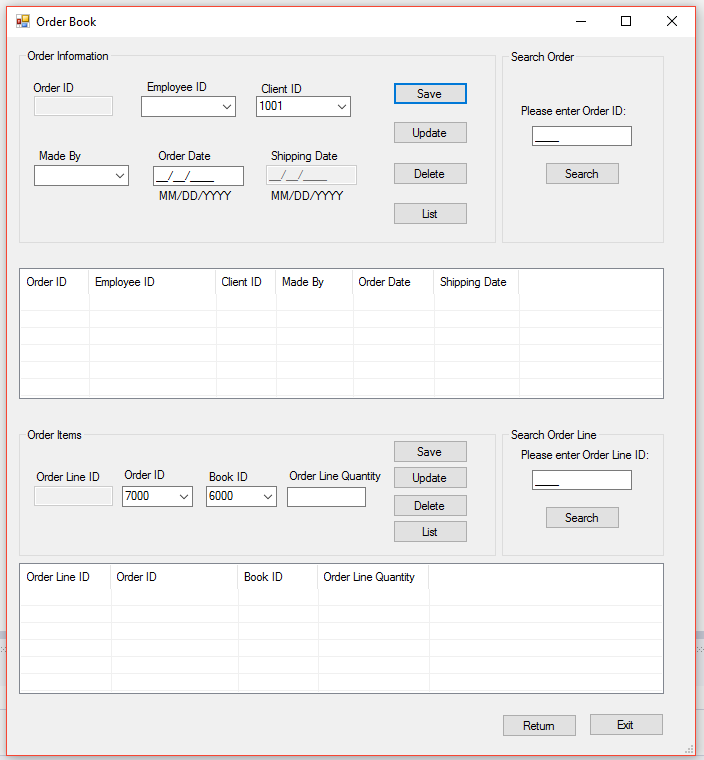




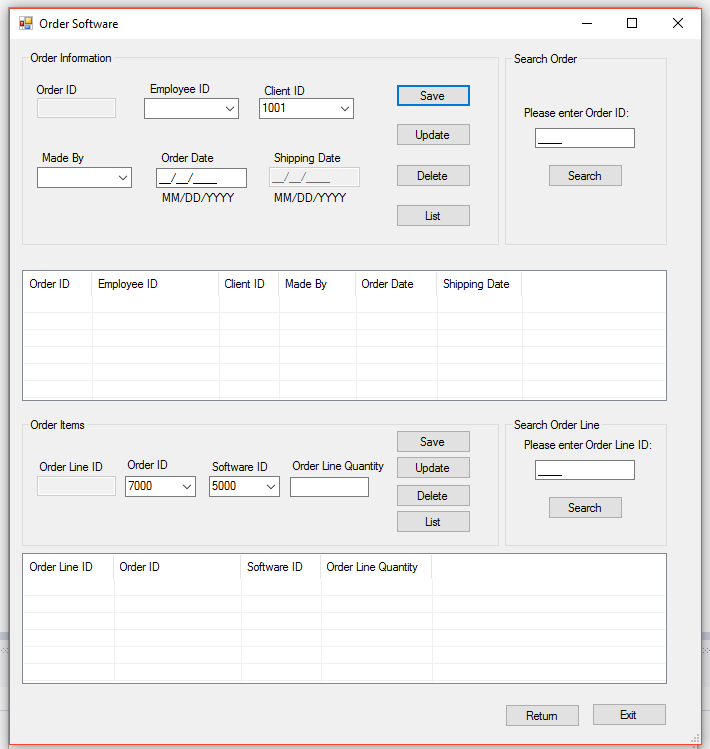
The Order Clerks have access to the Order Maintenance form, which allows to choose if they want to create a Book or a Software.



If they click in the button Book, the Order Book Form will open, and there they can create, update, list, delete and search for a book order. After they create the order, they can choose which items that order will have.



If they click in the button Software, the Order Software Form will open, and there they can create, update, list, delete and search for a software order. After they create the order, they can choose which items that order will have.



# 2.3 Implementation

In Visual Studio, two class libraries and one windows form application were created. The first class library was used to do the connected mode. It contains three folders: BLL, DAL and Validation. In the DAL folder there are two classes, EmployeeDB and UtilityDB.

The UtilityDB constains the method ConnectDB, to connect the solution with the database.

namespace ClassLibrary1.DAL

{

public static class UtilityDB

{

public static SqlConnection ConnectDB()

{

SqlConnection conn = new SqlConnection();

conn.ConnectionString = ConfigurationManager.ConnectionStrings["HiTech\_DBConnectionString"].ConnectionString;

conn.Open();

return conn;

}

}

}

The Employee DB contains all the methods to make the Employee Form work.

namespace ClassLibrary1.DAL

{

public static class EmployeeDB

{

//Save Method

public static bool SaveRecord(Employee emp)

{

bool success = false;

try

{

string sqlInsert = "INSERT INTO Employees " +

"VALUES (@FirstName,@LastName,@JobTitle,@UserType,@Password,@Email)";

SqlConnection sqlConn = UtilityDB.ConnectDB();

SqlCommand sqlCmd = new SqlCommand(sqlInsert, sqlConn);

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

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

sqlCmd.Parameters.AddWithValue("@JobTitle", emp.JobTitle);

sqlCmd.Parameters.AddWithValue("@UserType", emp.UserType);

sqlCmd.Parameters.AddWithValue("@Password", emp.Password);

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

sqlCmd.ExecuteNonQuery();

sqlConn.Close();

success = true;

}

catch (SqlException ex)

{

throw ex;

}

return success;

}

//Delete Method

public static bool DeleteRecord(int empId)

{

bool success = false;

try

{

string sqlDelete = "DELETE FROM employees " +

"WHERE employeeId = " + empId;

SqlConnection sqlConn = UtilityDB.ConnectDB();

SqlCommand sqlCmd = new SqlCommand(sqlDelete, sqlConn);

sqlCmd.ExecuteNonQuery();

sqlConn.Close();

success = true;

}

catch (SqlException ex)

{

throw ex;

}

return success;

}

//Search Method

public static Employee SearchRecordById(int empId)

{

Employee emp = new Employee();

string sqlSelect = "SELECT \* FROM Employees " +

"WHERE employeeId = " + empId;

SqlConnection sqlConn = UtilityDB.ConnectDB();

SqlCommand sqlCmd = new SqlCommand(sqlSelect, sqlConn);

SqlDataReader sqlReader = sqlCmd.ExecuteReader();

if (sqlReader.Read())

{

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

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

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

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

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

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

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

}

else

{

emp = null;

}

sqlConn.Close();

return emp;

}

//List Method

public static List<Employee> ListRecords()

{

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

string sqlSelect = "SELECT \* FROM Employees ";

SqlConnection sqlConn = UtilityDB.ConnectDB();

SqlCommand sqlCmd = new SqlCommand(sqlSelect, sqlConn);

SqlDataReader sqlReader = sqlCmd.ExecuteReader();

while (sqlReader.Read())

{

Employee emp = new Employee();

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

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

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

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

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

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

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

listEmp.Add(emp);

}

sqlConn.Close();

return listEmp;

}

//Authentication Method

public static string Password\_Authentication(int empId)

{

string sqlSelect = "SELECT password FROM Employees " +

"WHERE employeeId = " + empId;

SqlConnection sqlConn = UtilityDB.ConnectDB();

SqlCommand sqlCmd = new SqlCommand(sqlSelect, sqlConn);

SqlDataReader sqlReader = sqlCmd.ExecuteReader();

string sqlPassword = string.Empty;

if (sqlReader.HasRows)

{

while (sqlReader.Read())

{

sqlPassword = sqlReader["Password"].ToString();

}

}

sqlConn.Close();

return sqlPassword;

}

//Change password

public static bool ChangePassword(Employee emp)

{

bool success = false;

try

{

string sqlUpdate = "UPDATE Employees " +

" SET password = '" + emp.Password + "'" +

" WHERE EmployeeId = " + emp.EmployeeId;

SqlConnection sqlConn = UtilityDB.ConnectDB();

SqlCommand sqlCmd = new SqlCommand(sqlUpdate, sqlConn);

sqlCmd.ExecuteNonQuery();

sqlConn.Close();

success = true;

}

catch (SqlException ex)

{

throw ex;

}

return success;

}

}

}

The second class library was used to do the disconnected mode. It contains two folders: BLL and DAL. In the DAL there is the UtilityDB class and in the BLL there is the Client class.

The UtilityDB contains the connection to database, SqlAdapter and DataSet methods.

namespace ClassLibrary\_Disconnected\_Mode.DAL

{

public static class UtilityDB

{

private static string GetConnectionString()

{

string connectionString = ConfigurationManager.ConnectionStrings["HiTech\_DBConnectionString"].ConnectionString;

return connectionString;

}

public static SqlConnection ConnectDB()

{

SqlConnection sqlConn = new SqlConnection(GetConnectionString());

sqlConn.Open();

return sqlConn;

}

private static SqlDataAdapter sqlDa = new SqlDataAdapter();

private static SqlCommandBuilder sqlCmdBuilder;

public static string tableNameClients = "Clients";

private static string sql = "SELECT \* FROM Clients";

public static SqlDataAdapter GetDataAdapterClients()

{

sqlDa = new SqlDataAdapter(sql, GetConnectionString());

sqlCmdBuilder = new SqlCommandBuilder(sqlDa);

return sqlDa;

}

public static DataSet GetDataSetClients(DataSet ds, DataTable dt)

{

sqlDa = GetDataAdapterClients();

sqlDa.FillSchema(ds, SchemaType.Source, tableNameClients);

dt.PrimaryKey = new DataColumn[] { dt.Columns["ClientId"] };

sqlDa.Fill(ds, tableNameClients);

return ds;

}

}

}

The class client contains the properties.

namespace ClassLibrary\_Disconnected\_Mode.BLL

{

public class Client

{

public int ClientId { get; set; }

public string ClientName { get; set; }

public int StreetNumber { get; set; }

public string StreetName { get; set; }

public string City { get; set; }

public string PostalCode { get; set; }

public string Province { get; set; }

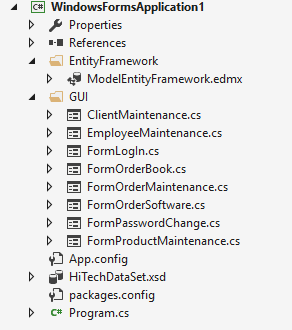
public string Phone { get; set; }

public float CreditLimit { get; set; }

}

}

The WindowsFormsApplication contains all the forms of the GUI and the code to make them work. It also contains all the Entity Framework classes.



# 2.4 Testing

|  |  |
| --- | --- |
| INPUT | RESULT |
| Log in – Employee id: 1111111  Password: admin | Successful Log in |
| Log in – Employee id: 1111111  Password: wrong | Message: Authentication error. Please try again. |
| Password change – Employee id: 1111111  Password: admin | Message: Password modified |
| Employee Maintenance  Add Employee | Successful |
| Employee Maintenance  Add Employee – Name: 1 | Message: Enter characters only |
| Employee Maintenance  List Employee | Successful |
| Employee Maintenance  Search Employee - 1111112 | Successful |
| Employee Maintenance  Search Employee - aa | Enter numbers only |
| Employee Maintenance  Update Employee | Successful |
| Employee Maintenance  Delete Employee | Successful |
| Client Maintenance  Add Client | Successful |
| Client Maintenance  Add Employee – Name: 1 | Message: Enter characters only |
| Client Maintenance  List Client | Successful |
| Client Maintenance  Search Client - 1001 | Successful |
| Client Maintenance  Search Client - b | Enter numbers only |
| Client Maintenance  Update Client | Successful |
| Client Maintenance  Delete Client | Successful |
| Book Maintenance  Add Book | Successful |
| Book Maintenance  Add Book – Name: 1 | Message: Enter characters only |
| Book Maintenance  List Book | Successful |
| Book Maintenance  Search Book - 6000 | Successful |
| Book Maintenance  Search Book - b | Enter numbers only |
| Book Maintenance  Update Book | Successful |
| Book Maintenance  Delete Book | Successful |
| Order Book  Add Order | Successful |
| Order Book  Add Order – Name: 1 | Message: Enter characters only |
| Order Book  List Order | Successful |
| Order Book  Search Order - 7000 | Successful |
| Order Book  Search Order - b | Enter numbers only |
| Order Book  Update Order | Successful |
| Order Book  Delete Order | Successful |

# 2.5 Deploying the Application

In order to run the application, the computer has be able to run Windows operational system and to have SQL Server and Visual Studio installed.

# 3 Conclusion

From this project, I’ve learned to create and implement a moderate size database. I also learned how to integrate connected, disconnected mode and Entity Framework in one project. Besides that, I learned how to save, update, delete, list and search data in ADO.Net Object Model and .Net Entity Framework.