Rahul Rajat Singh's blog

A Technology Journal of a Software Developer

Creating ASP.NET Applications with N-Tier Architecture

JUNE 24, 2014

This article describes how to build ASP.NET applications using n-tier architecture. The benefits of having n-tier architecture is that all the modules having dedicated functionality will be independent of each other. Changing one tier will not effect other tiers and there is no single point of failure even if some tier is not working.

Background

In a typical n-tier application there will be 4 Layers. The bottom most layer is the Data layer which contains the tables and stored procedures, scaler function, table values function. This Data layer is typically the database engine itself. We will be using `SqlServer` as the data layer in our example.

On top of Data Layer, we have a Data Access Layer (DAL). This layer is responsible for handling Database related tasks i.e. only data access. This `Data access layer` is created as a separate solution so that the changes in `DAL` only need the recompilation of DAL and not the complete website. The benefit of having this layer as a separate solution is that in case the database engine is changes we only need to change the `DAL` and the other areas of the website need not be changed and recompiled. Also the changes in other areas outside this solution will not demand for `DAL` recompilation.

Search …

Categories

ADO.NET

AngularJs

ASP.NET

ASP.NET Core

ASP.NET MVC

Azure

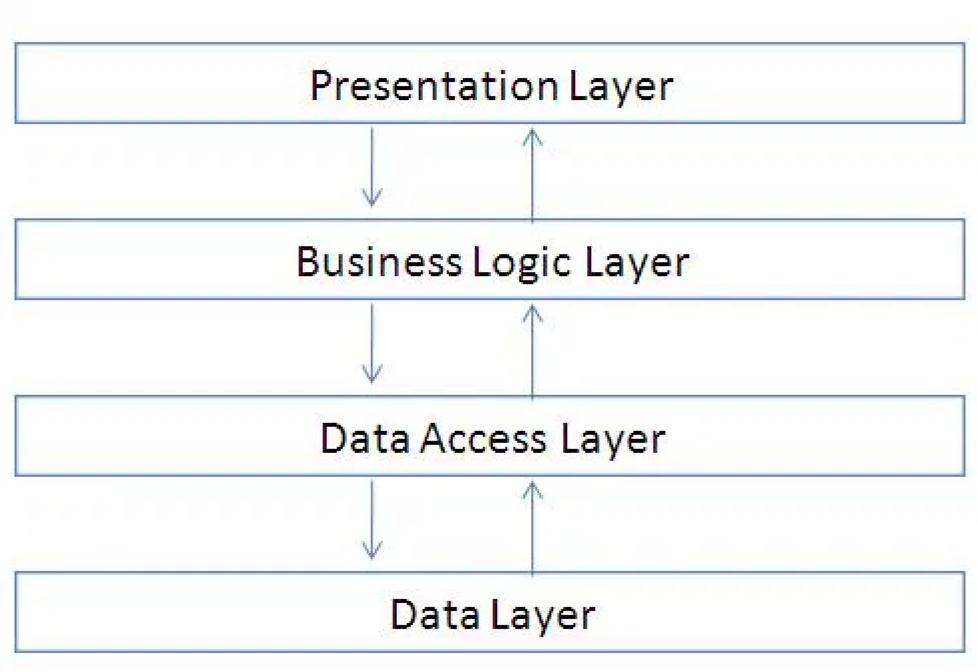
BackboneJS

On top of DAL, we have our `Business Logic Layer(BLL)`. `BLL` contains all the calculations and Business Rule validations that are required in the application. It is also in a separate solution for the reason that if the Business rules change or the calculations change we only need to recompile the `BLL` and the other layers of the application will remain unaffected.

Finally on top of `BLL` we have our Presentation Layer. The Presentation layer for an ASP.NET web forms application is all the Forms (`apsx` pages and their code behinds) and the classes contained in the App_Code folder. The Presentation layer is responsible for taking the user input, showing the data to the user and mainly performing input data validation.

Note: Input data filtration and validation is typically done at the Presentation Layer(Both client side and server side). The business Rule validation will be done at the `BLL`.

So to visualize the above mentioned architecture:



Note: The Data Access Layer in this article was written using classic `ADO.NET`, due to which the amount of code in `DAL` is little too much. Nowadays using ORMs like `Entity framework` to

Blog

C#

Dapper

Design Patterns

Entity Framework

JavaScript

Software Architecture

TIPS

WCF

WEB API

Web Services

Subscribe via Email

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

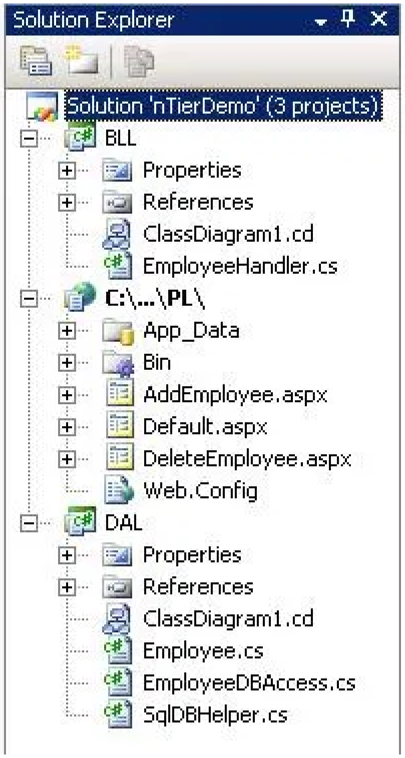
Email Address

generate the `DAL` is recommended. The `DAL` code will be generated by ORM itself.

Using the code

Let us develop a small Toy `ASP.NET` application that will use n-tier architecture. We will develop a small Employee Management application for the `NorthWind` Database. (For simplicity, I have removed all other tables from the DB and some columns from the Employee table). This application should be able to perform the basic CRUD operations on the DB.

The solution for this application will contain separate projects for `DAL` and `BLL`. The Data Layer will be `SqlServer`. The Presentation Layer is an ASP.NET website running on top of these projects.


Solution Explorer

```
Solution 'nTierDemo' (3 projects)
⊟ BLL
    ⊞ Properties
    ⊞ References
      ClassDiagram1.cd
      EmployeeHandler.cs
⊟ C:\...\PL\
    ⊞ App_Data
    ⊞ Bin
    ⊞ AddEmployee.aspx
    ⊞ Default.aspx
    ⊞ DeleteEmployee.aspx
      Web.Config
⊟ DAL
    ⊞ Properties
    ⊞ References
      ClassDiagram1.cd
      Employee.cs
      EmployeeDBAccess.cs
      SqlDBHelper.cs
```

The Data Layer

The data layer in this example contain only one table called Employee. The data layer also contains the stored procedures for all the basic operations on the Employee table. So let us look at the table and all the stored Procedures we have in our Data Layer.

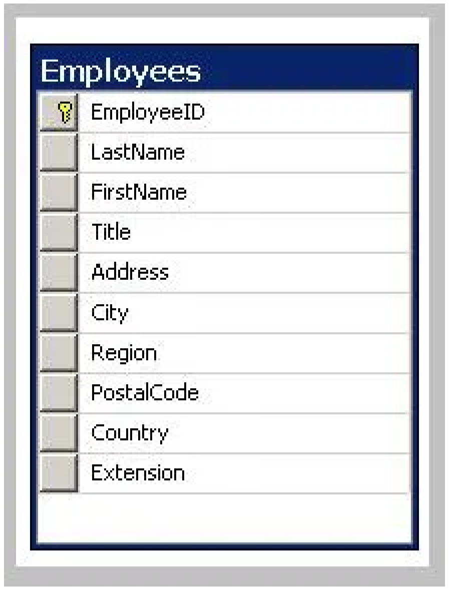
Rahul Rajat Singh

@rahul-rajat-singh

Architect

Author | Legend

Follow me on Twitter



Employees
- EmployeeID
- LastName
- FirstName
- Title
- Address
- City
- Region
- PostalCode
- Country
- Extension

Now we will create a set of stored procedures to perform the operations on the Employees Table.

```
--1. Procedure to add a new employee
CREATE PROCEDURE dbo.AddNewEmployee
    (
        @LastName   nvarchar(20),
        @FirstName  nvarchar(10),
        @Title      nvarchar(30),
        @Address    nvarchar(60),
        @City       nvarchar(15),
        @Region     nvarchar(15),
        @PostalCode nvarchar(10),
        @Country    nvarchar(15),
        @Extension  nvarchar(4)
    )
AS
    insert into Employees
    (LastName, FirstName, Title, Address, City, Region, PostalCode, Country, Extension)
    values
    (@LastName, @FirstName, @Title, @Address, @City, @Region, @PostalCode, @Country, @Ext
    RETURN

--2. Procedure to delete an employee
CREATE PROCEDURE dbo.DeleteEmployee
    (
    @empId int
    )
```

Tweets by @rahulrajatsingh

Rahul Singh Retweeted

Jeremy Clark
@jeremybytes

Me [mentions something about Kubernetes]
My wife: "What's that?"
Me: Well... [starts explanation about containers and orchestration]
10 minutes in...
My wife: "I'm really sorry I asked that."

Feb 24, 2020

Rahul Singh

Embed View on Twitter

Rahul Rajat Singh...
500 likes

Like Page

Be the first of your friends to like this

Rahul Rajat Singh's Blog
about a year ago

```
AS
    delete from Employees where EmployeeID = @empId
    RETURN

--3. Procedure to add get an employee details
CREATE PROCEDURE dbo.GetEmployeeDetails
    (
    @empId int
    )

AS
    Select * from Employees where EmployeeID = @empId
    RETURN

--4. Procedure to get all the employees in the table
CREATE PROCEDURE dbo.GetEmployeeList
AS
    Select * from Employees

    RETURN

--5. Procedure to update an employee details
CREATE PROCEDURE dbo.UpdateEmployee
    (
        @EmployeeID int,
        @LastName   nvarchar(20),
        @FirstName  nvarchar(10),
        @Title      nvarchar(30),
        @Address    nvarchar(60),
        @City       nvarchar(15),
        @Region     nvarchar(15),
        @PostalCode nvarchar(10),
        @Country    nvarchar(15),
        @Extension  nvarchar(4)
    )
AS
    update Employees
    set
        LastName = @LastName,
        FirstName = @FirstName,
        Title = @Title,
        Address = @Address,
        City = @City,
        Region = @Region,
        PostalCode = @PostalCode,
        Country = @Country,
        Extension = @Extension
    where
```

Recent Posts

CodeProject MVP 2019

A Beginner's Tutorial On Understanding and Implementing Dependency Injection in ASP.NET Core

Tutorial on Handling Multiple Resultsets and Multiple Mapping using Dapper

An Absolute Beginner's Tutorial on Middleware in ASP.NET Core/MVC (and writing custom middleware)

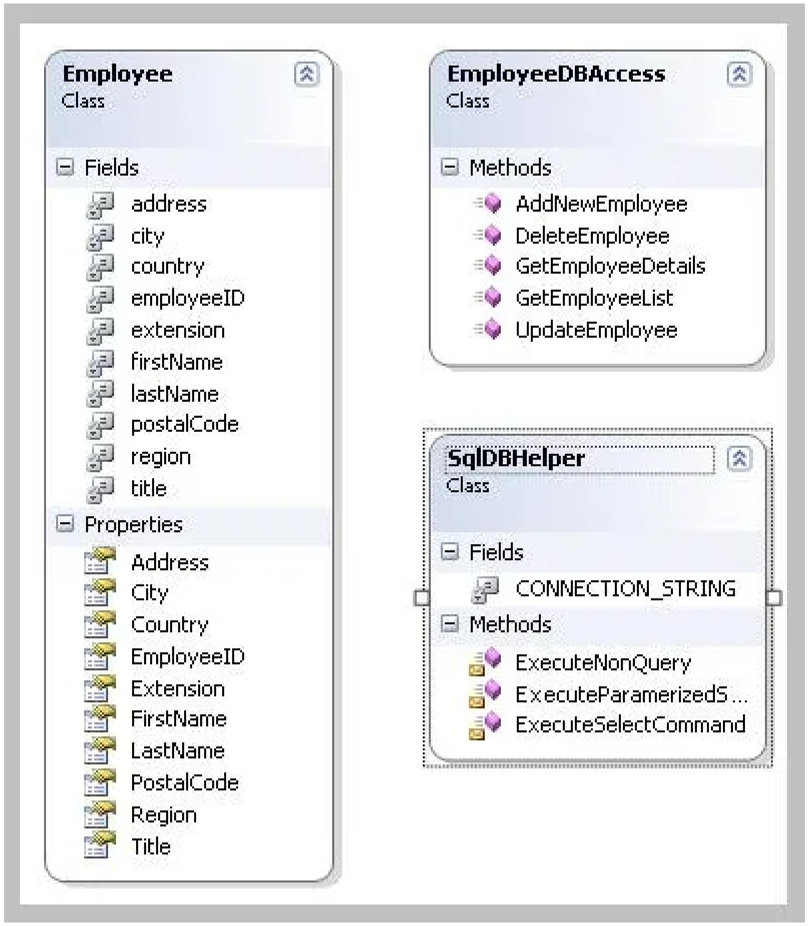
Absolute Beginner's Tutorial on understanding and using Dapper ORM

```
        EmployeeID = @EmployeeID
    RETURN
```

Now we have our Data Layer ready.

The Data Access Layer

Now we will go ahead and create a Data Access Layer for our application. The data access layer will contain 2 main type of classes. A set of classes that will represent the Table entities. And classes to perform the `CRUD` operations on the database.



The `Employee` class in the above diagram is the Entity that will represent the `Employee` table. This class has been created so that the Layers above the `DAL` will use this class to perform operations in

Employee table and they need not worry about the table schema related details.

```
public class Employee
{
    int employeeID;
    string lastName;    //  should be (20)  chars only
    string firstName;   //  should be (10)  chars only
    string title;       //  should be (30)  chars only
    string address;     //  should be (60)  chars only
    string city;        //  should be (15)  chars only
    string region;      //  should be (15)  chars only
    string postalCode;  //  should be (10)  chars only
    string country;     //  should be (15)  chars only
    string extension;   //  should be (4)   chars only

    public int EmployeeID
    {
        get
        {
            return employeeID;
        }
        set
        {
            employeeID = value;
        }
    }

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

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

    public string Title
    {
        get
        {
            return title;
        }
        set
        {
            title = value;
        }
    }

    public string Address
    {
        get
        {
            return address;
        }
        set
        {
            address = value;
        }
    }

    public string City
    {
        get
        {
            return city;
        }
        set
        {
            city = value;
        }
    }

    public string Region
    {
        get
        {
            return region;
        }
        set
        {
            region = value;
        }
```

```
    }

    public string PostalCode
    {
        get
        {
            return postalCode;
        }
        set
        {
            postalCode = value;
        }
    }

    public string Country
    {
        get
        {
            return country;
        }
        set
        {
            country = value;
        }
    }

    public string Extension
    {
        get
        {
            return extension;
        }
        set
        {
            extension = value;
        }
    }
}
```

The `EmployeeDBAccess` class expose the methods to perform the `CRUD` operations on the Employee table.

```
public class EmployeeDBAccess
{
    public bool AddNewEmployee(Employee employee)
    {
        SqlParameter[] parameters = new SqlParameter[]
        {
            new SqlParameter("@LastName", employee.LastName),
            new SqlParameter("@FirstName", employee.FirstName),
            new SqlParameter("@Title", employee.Title),
            new SqlParameter("@Address", employee.Address),
            new SqlParameter("@City", employee.City),
            new SqlParameter("@Region", employee.Region),
            new SqlParameter("@PostalCode", employee.PostalCode),
            new SqlParameter("@Country", employee.Country),
            new SqlParameter("@Extension", employee.Extension)
        };

        return SqlDBHelper.ExecuteNonQuery("AddNewEmployee", CommandType.StoredProcedure,
    }

    public bool UpdateEmployee(Employee employee)
    {
        SqlParameter[] parameters = new SqlParameter[]
        {
            new SqlParameter("@EmployeeID", employee.EmployeeID),
            new SqlParameter("@LastName", employee.LastName),
            new SqlParameter("@FirstName", employee.FirstName),
            new SqlParameter("@Title", employee.Title),
            new SqlParameter("@Address", employee.Address),
            new SqlParameter("@City", employee.City),
            new SqlParameter("@Region", employee.Region),
            new SqlParameter("@PostalCode", employee.PostalCode),
            new SqlParameter("@Country", employee.Country),
            new SqlParameter("@Extension", employee.Extension)
        };

        return SqlDBHelper.ExecuteNonQuery("UpdateEmployee", CommandType.StoredProcedure,
    }

    public bool DeleteEmployee(int empID)
    {
        SqlParameter[] parameters = new SqlParameter[]
        {
            new SqlParameter("@empId", empID)
        };

        return SqlDBHelper.ExecuteNonQuery("DeleteEmployee", CommandType.StoredProcedure,
    }

    public Employee GetEmployeeDetails(int empID)
    {
        Employee employee = null;

        SqlParameter[] parameters = new SqlParameter[]
        {
            new SqlParameter("@empId", empID)
        };
        //Lets get the list of all employees in a datataable
        using (DataTable table = SqlDBHelper.ExecuteParamerizedSelectCommand("GetEmployee
        {
            //check if any record exist or not
            if (table.Rows.Count == 1)
            {
                DataRow row = table.Rows[0];

                //Lets go ahead and create the list of employees
                employee = new Employee();

                //Now lets populate the employee details into the list of employees
                employee.EmployeeID = Convert.ToInt32(row["EmployeeID"]);
                employee.LastName = row["LastName"].ToString();
                employee.FirstName = row["FirstName"].ToString();
                employee.Title = row["Title"].ToString();
                employee.Address = row["Address"].ToString();
                employee.City = row["City"].ToString();
                employee.Region = row["Region"].ToString();
                employee.PostalCode = row["PostalCode"].ToString();
                employee.Country = row["Country"].ToString();
                employee.Extension = row["Extension"].ToString();
            }
        }

        return employee;
    }

    public List<employee> GetEmployeeList()
    {
        List<employee> listEmployees = null;

        //Lets get the list of all employees in a datataable
        using (DataTable table = SqlDBHelper.ExecuteSelectCommand("GetEmployeeList", Comm
        {
            //check if any record exist or not
            if (table.Rows.Count > 0)
            {
                //Lets go ahead and create the list of employees
                listEmployees = new List<employee>();

                //Now lets populate the employee details into the list of employees
                foreach (DataRow row in table.Rows)
                {
                    Employee employee = new Employee();
                    employee.EmployeeID = Convert.ToInt32(row["EmployeeID"]);
                    employee.LastName = row["LastName"].ToString();
                    employee.FirstName = row["FirstName"].ToString();
                    employee.Title = row["Title"].ToString();
                    employee.Address = row["Address"].ToString();
                    employee.City = row["City"].ToString();
                    employee.Region = row["Region"].ToString();
                    employee.PostalCode = row["PostalCode"].ToString();
                    employee.Country = row["Country"].ToString();
                    employee.Extension = row["Extension"].ToString();

                    listEmployees.Add(employee);
                }
            }
        }

        return listEmployees;
    }
}
</employee></employee></employee>
```

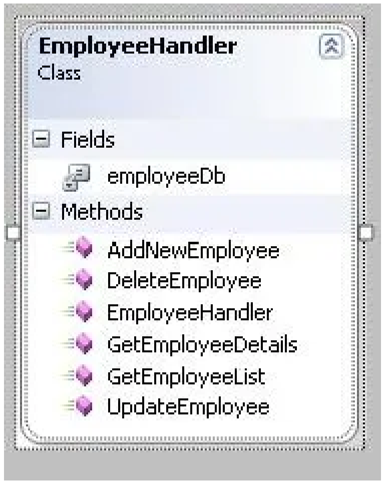
The class `SqlDbHelper` is a wrapper class for `ADO.NET` functions providing a more simpler interface to use by the rest of DAL.

```
class SqlDBHelper
{
    const string CONNECTION_STRING = @"Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDir

    // This function will be used to execute R(CRUD) operation of parameterless commands
    internal static DataTable ExecuteSelectCommand(string CommandName, CommandType cmdTyp
    {
        DataTable table = null;
        using (SqlConnection con = new SqlConnection(CONNECTION_STRING))
        {
            using (SqlCommand cmd = con.CreateCommand())
            {
                cmd.CommandType = cmdType;
                cmd.CommandText = CommandName;

                try
                {
                    if (con.State != ConnectionState.Open)
                    {
                        con.Open();
                    }

                    using (SqlDataAdapter da = new SqlDataAdapter(cmd))
                    {
                        table = new DataTable();
                        da.Fill(table);
                    }
                }
                catch
                {
                    throw;
                }
            }
        }

        return table;
    }

    // This function will be used to execute R(CRUD) operation of parameterized commands
    internal static DataTable ExecuteParamerizedSelectCommand(string CommandName, Commanc
    {
        DataTable table = new DataTable();

        using (SqlConnection con = new SqlConnection(CONNECTION_STRING))
        {
            using (SqlCommand cmd = con.CreateCommand())
            {
                cmd.CommandType = cmdType;
                cmd.CommandText = CommandName;
                cmd.Parameters.AddRange(param);

                try
                {
                    if (con.State != ConnectionState.Open)
                    {
                        con.Open();
                    }

                    using (SqlDataAdapter da = new SqlDataAdapter(cmd))
                    {
                        da.Fill(table);
                    }
                }
                catch
                {
                    throw;
                }
            }
        }

        return table;
    }

    // This function will be used to execute CUD(CRUD) operation of parameterized commanc
    internal static bool ExecuteNonQuery(string CommandName, CommandType cmdType, SqlPara
    {
        int result = 0;

        using (SqlConnection con = new SqlConnection(CONNECTION_STRING))
        {
            using (SqlCommand cmd = con.CreateCommand())
            {
                cmd.CommandType = cmdType;
                cmd.CommandText = CommandName;
                cmd.Parameters.AddRange(pars);

                try
                {
                    if (con.State != ConnectionState.Open)
                    {
                        con.Open();
                    }

                    result = cmd.ExecuteNonQuery();
                }
                catch
                {
                    throw;
                }
            }
        }

        return (result > 0);
    }
}
```

Note: If we use any `ORM` (Object Relation Mapper) then DAL need not be written. The `ORM` will generate all the DAL code. `Entity framework` is one of the best `ORMs` available. This DAL can simply be replaced with a class library containing the `Entity Framework` generated Entities and Contexts.

The Business Logic Layer

The business logic layer will have a reference to the DAL and will mainly perform Business rule validation and business logic specific calculations. In out example, I will write a simple `BLL` that will govern the IO between the `DAL` and Presentation layer. In real applications the `BLL` will contain more logic and code.



EmployeeHandler
Class

⊟ Fields
- employeeDb

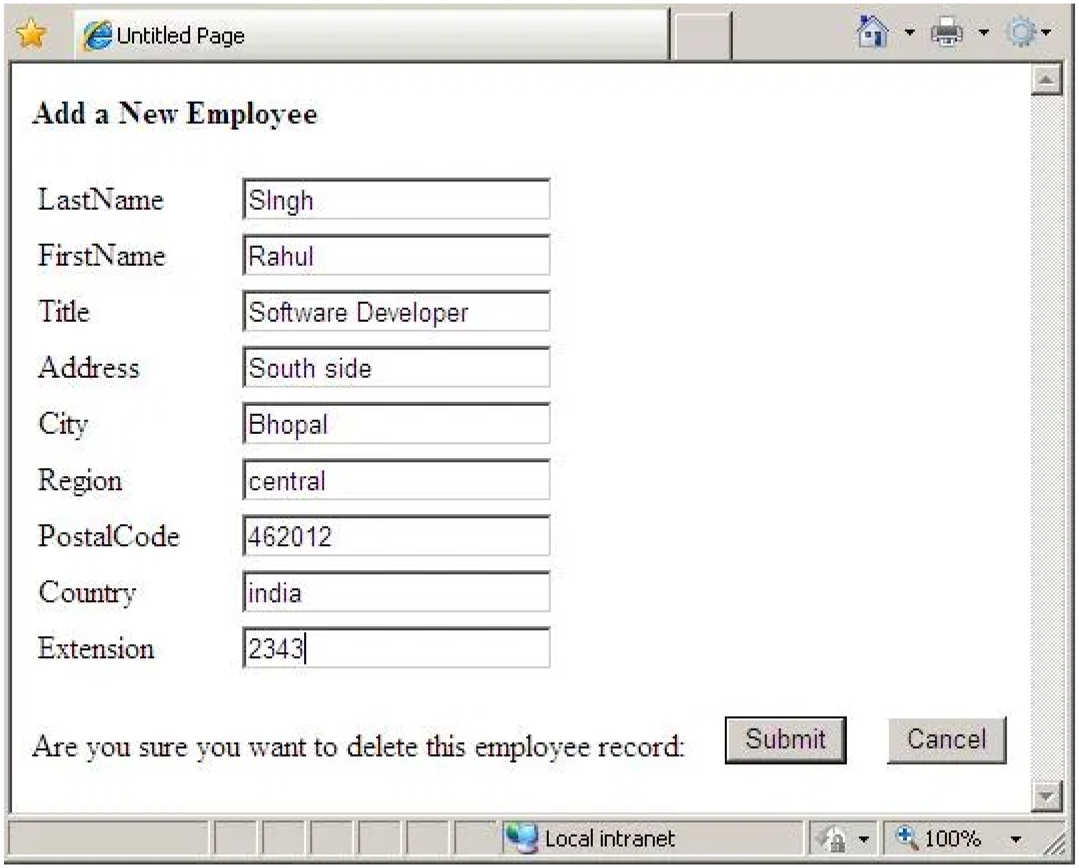
⊟ Methods
- AddNewEmployee
- DeleteEmployee
- EmployeeHandler
- GetEmployeeDetails
- GetEmployeeList
- UpdateEmployee

```
public class EmployeeHandler
{
    // Handle to the Employee DBAccess class
    EmployeeDBAccess employeeDb = null;

    public EmployeeHandler()
    {
        employeeDb = new EmployeeDBAccess();
    }

    // This fuction does not contain any business logic, it simply returns the
    // list of employees, we can put some logic here if needed
    public List<employee> GetEmployeeList()
    {
        return employeeDb.GetEmployeeList();
    }

    // This fuction does not contain any business logic, it simply returns the
    // list of employees, we can put some logic here if needed
    public bool UpdateEmployee(Employee employee)
    {
        return employeeDb.UpdateEmployee(employee);
    }

    // This fuction does not contain any business logic, it simply returns the
    // list of employees, we can put some logic here if needed
    public Employee GetEmployeeDetails(int empID)
    {
        return employeeDb.GetEmployeeDetails(empID);
    }
    // This fuction does not contain any business logic, it simply returns the
    // list of employees, we can put some logic here if needed
    public bool DeleteEmployee(int empID)
    {
        return employeeDb.DeleteEmployee(empID);
    }

    // This fuction does not contain any business logic, it simply returns the
    // list of employees, we can put some logic here if needed
    public bool AddNewEmployee(Employee employee)
    {
        return employeeDb.AddNewEmployee(employee);
    }
}
```

The Presentation Layer

The presentation layer now contains only a set of pages and code behinds and it will use the `BLL` and the the Employee class to perform all the operations. The add Operation can be seen as an example how the `BLL` is being used to perform an operation.

```
Employee emp = new Employee();

emp.LastName = txtLName.Text;
emp.FirstName = txtFName.Text;
emp.Address = txtAddress.Text;
emp.City = txtCity.Text;
emp.Country = txtCountry.Text;
emp.Region = txtRegion.Text;
emp.PostalCode = txtCode.Text;
emp.Extension = txtExtension.Text;
emp.Title = txtTitle.Text;

EmployeeHandler empHandler = new EmployeeHandler();

if (empHandler.AddNewEmployee(emp) == true)
{
    //Successfully added a new employee in the database
    Response.Redirect("Default.aspx");
}
```

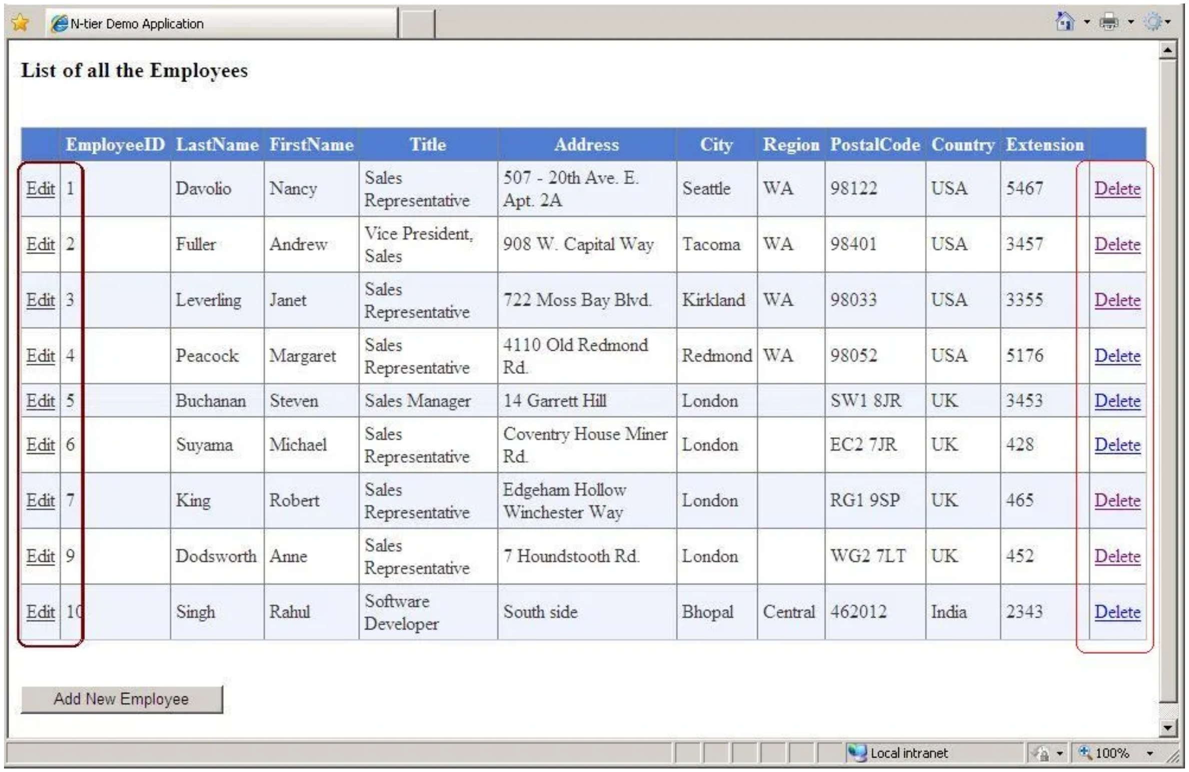


Add a New Employee

LastName	SIngh
FirstName	Rahul
Title	Software Developer
Address	South side
City	Bhopal
Region	central
PostalCode	462012
Country	india
Extension	2343

Are you sure you want to delete this employee record: Submit Cancel

Note: All the CRUD operations have been implemented. Please refer tio the sample code for all the details. When we run the application we can see all the EDIT/UPDATE, DELETE and ADD operations in action.



List of all the Employees

	EmployeeID	LastName	FirstName	Title	Address	City	Region	PostalCode	Country	Extension	
Edit	1	Davolio	Nancy	Sales Representative	507 - 20th Ave. E. Apt. 2A	Seattle	WA	98122	USA	5467	Delete
Edit	2	Fuller	Andrew	Vice President, Sales	908 W. Capital Way	Tacoma	WA	98401	USA	3457	Delete
Edit	3	Leverling	Janet	Sales Representative	722 Moss Bay Blvd.	Kirkland	WA	98033	USA	3355	Delete
Edit	4	Peacock	Margaret	Sales Representative	4110 Old Redmond Rd.	Redmond	WA	98052	USA	5176	Delete
Edit	5	Buchanan	Steven	Sales Manager	14 Garrett Hill	London		SW1 8JR	UK	3453	Delete
Edit	6	Suyama	Michael	Sales Representative	Coventry House Miner Rd.	London		EC2 7JR	UK	428	Delete
Edit	7	King	Robert	Sales Representative	Edgeham Hollow Winchester Way	London		RG1 9SP	UK	465	Delete
Edit	9	Dodsworth	Anne	Sales Representative	7 Houndstooth Rd.	London		WG2 7LT	UK	452	Delete
Edit	10	Singh	Rahul	Software Developer	South side	Bhopal	Central	462012	India	2343	Delete

Point of Interest

I created this small application to demonstrate application development using n-tier architecture. The demo application has been created to show the basic idea behind the 3-tier architecture. There are many things that are still missing from this sample from the completion perspective. Client side validation and server side validation in presentation layer, Business rule validation and calculations in BLL are some missing things.

Since the idea here was to talk about how to put n-tier architecture in actual code, I think this article might have provided some useful information on that. I hope this has been informative.

[UPDATE] Note: In this article I am reusing the `Employee` model in the presentation layer. This model is defined in Data Access Layer. Due to this the presentation layer has to refer to the data access layer. This is not ideal in the real world scenarios(as pointed out in many of the comments below). Ideal solution for this would be to have two different models for `Employee`. the current model which is defined in the data access layer can be called as the data model and the business logic layer can create a model for employee which will be called as domain model. The business logic layer will then have to contain the code for mapping the data model to the domain model and vice versa. This mapping can be done either manually or a tool like `AutoMapper` can also be used to perform such mapping. With this change the presentation layer need not refer to the data access layer but it can refer to the business logic layer and use the Employee domain model from that.

In this article the n-tier architecture is specifically a data centric n-tier and not a domain centric one. If we need to design the application in a domain centric n-tier architecture then we need to follow a different way of organizing our layers. But perhaps that is a topic which deserves a separate discussion altogether but I wanted to point out the possibility of a domain centric n-tier architecture in this article.

Download sample code for this article: nTierDemo

Share this:

Twitter | Facebook | LinkedIn | Reddit | Email

Like this:

Like

Be the first to like this.

Related

YaBlogEngine - A Tiny Blog Engine written in ASP.NET/C#
June 20, 2014
In “ASP.NET”

YaMessaging - A simple e-mail like messaging application
June 23, 2014
In “ASP.NET”

Beginner's Guide for Designing ASP.NET MVC Applications using SQL Server and Entity Framework
June 24, 2014
In “ASP.NET MVC”

ASP.NET 3 TIER ARCHITECTURE, ASP.NET, N TIER, N TIER ARCHITECTURE

← BEGINNER’S TUTORIAL ON GLOBALIZATION AND LOCALIZATION IN ASP.NET MVC

CREATING UNIT TESTABLE APPLICATIONS IN ASP.NET MVC – A BEGINNER’S TUTORIAL →