Arquitectura 3 Capas

Problema: Introducción

Objetivo: Añadir capas para mejorar este código.

Tenemos un formulario para sumar números.

<div>

<asp:Label ID="lblNum1" runat="server" Text="Numero 1:"></asp:Label>

<asp:TextBox ID="txtNum1" runat="server" Width="20"></asp:TextBox>

<br />

<asp:Label ID="lblNum2" runat="server" Text="Numero 2:"></asp:Label>

<asp:TextBox ID="txtNum2" runat="server" Width="20"></asp:TextBox>

<br />

<asp:Button ID="btnAdd" runat="server" Text="Sumar" OnClick="btnSumar\_Click" />

<asp:Label ID="lblResultado" runat="server" Text=""></asp:Label>

</div>

Y el CodeBehind es:

protected void btnSumar\_Click(object sender, EventArgs e)

{

int num1 = 0;

int num2 = 0;

int res = 0;

num1 = Convert.ToInt32(txtNum1.Text);

num2 = Convert.ToInt32(txtNum2.Text);

res = num1 + num2;

lblResultado.Text = res.ToString();

}

**RESPUESTA**:

Crear una clase:

public class clsNumeros

{

public int Num1 { get; set; }

public int Num2 { get; set; }

public clsNumeros()

{

Num1 = 0;

Num2 = 0;

}

public int Sumar()

{

return Num1 + Num2;

}

}

Instanciar la clase en la página

protected void btnSumarconClases\_Click(object sender, EventArgs e)

{

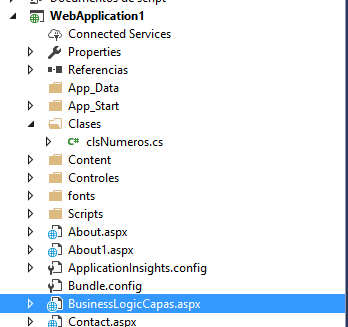
clsNumeros objNum = new clsNumeros();

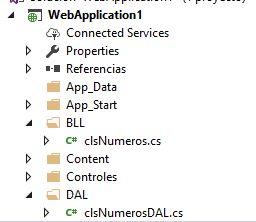
objNum.Num1 = Convert.ToInt32(txtNum1.Text);

objNum.Num2 = Convert.ToInt32(txtNum2.Text);

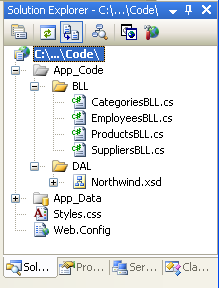
lblResultado.Text = objNum.Sumar().ToString();

}









**Lógica de Negocio**

Cuando se trabaja con datos de una opción es insertar la lógica específica de datos directamente en la capa de presentación (en una aplicación web, la capa de presentación constituyen las páginas ASP.NET). Esto puede adoptar la forma de escribir código de ADO.NET en la parte del código de la página ASP.NET o mediante el control SqlDataSource desde la parte del marcado. En cualquier caso, este enfoque acopla estrechamente la lógica de acceso a datos con la capa de presentación. Sin embargo, el enfoque recomendado es separar la lógica de acceso a datos de la capa de presentación. Esta capa independiente se conoce como la capa de acceso a datos, la capa DAL para abreviar y normalmente se implementa como un proyecto de biblioteca de clases independiente.

Todo el código que es específico del origen de datos subyacente como la creación de una conexión a la base de datos, emitir **seleccione**, **insertar**, **actualización**, y **ELIMINAR** comandos etc. deben encontrarse en la capa DAL. La capa de presentación no debe contener todas las referencias a dicho código de acceso de datos, pero en su lugar debe realizar llamadas en la capa DAL para las solicitudes de todos los datos. Niveles de acceso a datos normalmente contienen métodos para tener acceso a la base de datos subyacente. Por ejemplo, tiene la base de datos Northwind, **productos** y **categorías** las tablas que registran los productos de venta y las categorías a las que pertenecen. En nuestro DAL tenemos métodos, como:

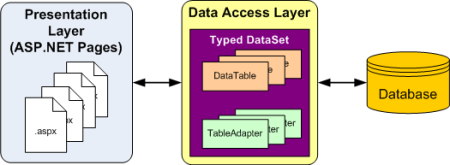
* **GetCategories(),** que devolverá información sobre todas las categorías
* **GetProducts()**, que devolverá información sobre todos los productos
* **GetProductsByCategoryID (categoryID)**, que devolverá todos los productos que pertenecen a una categoría especificada
* **GetProductByProductID (productID)**, que devolverá información sobre un producto determinado

Estos métodos, cuando se invoca, conectarse a la base de datos, ejecute la consulta adecuada y devolver los resultados. ¿Cómo se devuelven estos resultados es importante. Estos métodos podrían devolver simplemente un conjunto de datos o DataReader rellenada por la consulta de base de datos, pero lo ideal es que se deben devolver estos resultados con objetos fuertemente tipados. Un objeto fuertemente tipado es uno cuyo esquema se define de forma rígida en tiempo de compilación, mientras que lo contrario, un objeto fuertemente tipado, es uno cuyo esquema no se conoce hasta el tiempo de ejecución.

Por ejemplo, DataReader y el conjunto de datos (de forma predeterminada) son objetos fuertemente tipado dado que su esquema se define mediante las columnas devueltas por la consulta de base de datos utilizada para rellenarlos. Para obtener acceso a una columna concreta de una tabla de datos fuertemente tipado que necesitamos utilizar una sintaxis como: **DataTable. Filas [índice] ["columnName"]**. La DataTable flexible escribiendo en este ejemplo se mostró por el hecho de que necesitamos para tener acceso al nombre de columna mediante una cadena o el índice ordinal. Un objeto DataTable fuertemente tipado, por otro lado, tendrá cada uno de sus columnas que se implementan como propiedades, lo que genera código similar al siguiente: **DataTable. Filas [índice].columnName**.

Para devolver objetos fuertemente tipados, los desarrolladores pueden crear sus propios objetos de negocios personalizada o usar conjuntos de datos con tipo. Un objeto de negocios se implementa con el desarrollador como representa una clase cuyas propiedades suelen reflejan las columnas de la tabla de base de datos subyacente del objeto comercial. Un conjunto de datos con tipo es una clase generada automáticamente por Visual Studio basado en un esquema de base de datos y cuyos miembros son fuertemente tipada de acuerdo con este esquema. El conjunto de datos con tipo propio consta de las clases que extienden las clases de DataRow, DataTable y DataSet de ADO.NET. Además de tablas de datos fuertemente tipados, los conjuntos de datos con tipo ahora también incluyen los TableAdapters, que son clases con métodos para rellenar tablas de datos del conjunto de datos y propagar las modificaciones de las tablas de datos a la base de datos.

Figura 3 ilustra el flujo de trabajo entre los diferentes niveles de una aplicación que utilice conjuntos de datos con tipo.

[](https://docs.microsoft.com/es-es/aspnet/web-forms/overview/data-access/introduction/creating-a-data-access-layer-cs/_static/image5.png)

**Figura 3**: código de acceso de todos los datos es relegado a la capa DAL.

Recursos

<https://www.codeproject.com/Articles/493389/%2FArticles%2F493389%2FFour-ways-of-passing-data-between-layers>

<https://docs.microsoft.com/es-es/aspnet/web-forms/overview/data-access/introduction/creating-a-data-access-layer-cs>

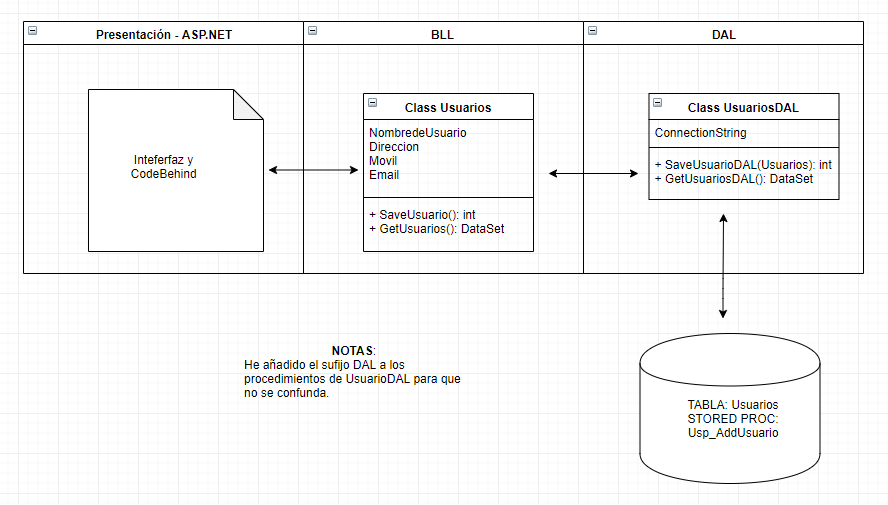
Ejemplos

<https://www.c-sharpcorner.com/UploadFile/4d9083/create-and-implement-3-tier-architecture-in-Asp-Net/>

<https://www.c-sharpcorner.com/article/three-tier-architecture-in-asp-net-with-example/>

**Proyecto**

Vamos a crear una página web usando una versión ligera de la arquitectura 3-tier.





Al añadir un usuario, se verá la nueva IDENTIDAD del usuario en un label o Response.write.

Código para la interfaz de usuario en un Web Form:

*GITHUB: Desarrollo-de-Aplicaciones-Multiplataforma-0486/Capas3tier/Web/BusinessLogicCapasUsuarios.aspx*

Separar las capas usando estructura de carpetas / directorios: BLL, DLL

Código subyacente para la capa de presentación:

protected void BtnSaveUser\_Click(object sender, EventArgs e)

{

int iID = 0;

Usuarios oUsuario = new Usuarios(); // Instanciar un objeto Usuario

oUsuario.NombredeUsuario = txtNombre.Text;

// Completar las propiedades de del objeto

// Guardar los datos de Usuario

iID = oUsuario.SaveUsuario();

// Mostrar la nueva identidad y un mensaje en la página. Response.Write o Label

}

Clases para la lógica de BLL:

public int SaveUsuario()

{

UsuarioDAL uDAL = new UsuarioDAL();

return uDAL.AddUsuarios(this); // Llamar el DAL

}

Clases para el acceso a la base de datos DAL:

public int AddUsuariosaBasedeDatos(Usuarios oUsuario)

{

int iResult = -1;

// Abrir Conexión

SqlConnection connection = new SqlConnection(connectionString);

try

{

connection.Open();

SqlCommand cmd = new SqlCommand("usp\_AddUsuario", connection);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@NombredeUsuario", oUsuario.NombredeUsuario.ToString());

// Añadir más valores a los Parameters

**Mejoras**:  
Se incluye código de Javascript para evaluar los datos en la capa de presentación.