**GUIA DE APRENDIZAJE NO. 001**

# **INTRODUCCIÓN AL DESARROLLO CON ASP.NET**

***INSTRUCTOR: Ing. Ariel Almonacid Arias***

***AREA:*** *Programación en ASP.NET.*

***ESTUDIANTE: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.***

***GRADO: \_\_\_\_\_\_\_\_\_\_\_.***

OBJETIVOS: presentar a los estudiantes conceptos básicos que se deben conocer para introducirse en el mundo de la programación en asp.NET.

**CONTENIDOS**

.NET ofrece un entorno gestionado de ejecución de aplicaciones, nuevos lenguajes de programación y compiladores, permite el desarrollo de todo tipo de funcionalidades: desde programas de consola o servicios Windows hasta aplicaciones para dispositivos móviles, pasando por desarrollos de escritorio o para Internet. Son estos últimos de los que nos ocuparemos en este periodo lectivo.

Más que una evolución sobre ASP, ASP.NET es una auténtica revolución en el mundo del desarrollo Web, y no sólo en el ámbito relacionado con Microsoft. Lo que mejor define al desarrollo combinado con ASP.NET y Visual Studio es que ha equiparado la creación de aplicaciones para Internet a lo que era común en entornos de escritorio, salvando las inherentes dificultades que ello conlleva de forma transparente al programador.

Los programadores de ASP.NET disponen de todo lo que "siempre" han disfrutado los programadores de Windows: diseñadores visuales, asistencia avanzada contextual, código compilado de alto rendimiento, transparencia acerca de dónde se ejecuta cada parte del código, enlace a datos, rejillas, etc...

**ASP.NET** es la parte del .NET Framework dedicada al desarrollo web. A través del servidor web, nuestras aplicaciones ASP.NET se ejecutarán bajo el CLR (**El Common Lenguage Runtime “ejecución común de lenguaje”**) y podremos usar el conjunto de clases del .NET Framework para desarrollarlas, obteniendo así una versatilidad y una potencia nunca antes conseguida en las aplicaciones ASP.

El CLR/CLI esencialmente define un entorno de ejecución virtual independiente en el que trabajan las aplicaciones escritas con cualquier lenguaje .NET. Este entorno virtual se ocupa de multitud de cosas importantes para una aplicación: desde la gestión de la memoria y la vida de los objetos hasta la seguridad y la gestión de subprocesos.

1. La sigla ASP significa Active Server Pages.
2. Microsoft es el creador.
3. Se ejecuta en el servidor donde está alojada la página web.
4. El cliente (navegador del usuario) no nota diferencias con una página normal (HTML).
5. Por medio de ASP se puede tener acceso a bases de datos.
6. Para su implementación se utiliza lenguaje script como VBScript o JScript.
7. Se puede utilizar HTML y ASP en una misma página.
8. Es totalmente ampliable gracias a que es compatible con la tecnología ActiveX.

**Como Funciona**

Aquí están paso a paso como funciona una página ASP.

1. Un usuario por medio del navegador solicita una página ASP.
2. Esta solicitud llega al servidor el cual tiene alojada la página pedida.
3. Este servidor procesa la página ASP y devuelve código HTML.
4. El usuario visualiza la página en su navegador.

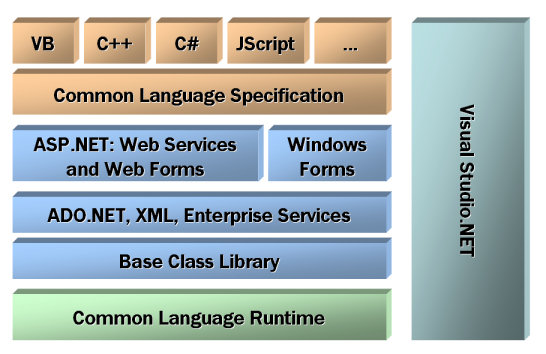
Noten que para el usuario no existe diferencia en ASP y HTML porque a su navegador siempre llega código HTML puro, el único que requiere un trabajo extra es el servidor el cual tiene que procesar el código ASP y transformarlo en HTML para su posterior envió al cliente.

# **La biblioteca de clases de .NET**

La plataforma .NET nos ofrece infinidad de funcionalidades "de fábrica" que se utilizan como punto de partida para crear las aplicaciones. Existen funcionalidades básicas (por ejemplo todo lo relacionado con la E/S de datos o la seguridad) y funcionalidades avanzadas en las que se fundamentan categorías enteras de aplicaciones (acceso a datos, creación de aplicaciones Web...).

Toda esta funcionalidad está implementada en forma de bibliotecas de funciones que físicamente se encuentran en diversas DLL (bibliotecas de enlazado dinámico). A su conjunto se le denomina Base Classes Library (Biblioteca de clases base o ***BCL***) y forman parte integral de la plataforma .NET, es decir, no se trata de añadidos que se deban obtener o adquirir aparte.

La siguiente figura ilustra a vista de pájaro la arquitectura conceptual de la plataforma .NET. En ella se pueden observar los elementos (lenguajes, CLR, CLS...) y en qué lugar de se ubican las bibliotecas de clases base:



***Distintos elementos de la plataforma .NET y cómo se relacionan entre sí.***

## Los espacios de nombres

Dada la ingente cantidad de clases que existen debe haber algún modo de organizarlas de un modo coherente. Además hay que tener en cuenta que podemos adquirir más funcionalidades (que se traducen en clases) a otros fabricantes, por no mencionar que crearemos continuamente nuevas clases propias.

Para solucionar este problema existen en todos los lenguajes .NET los ***espacios de nombres*** o **namespaces**.

Un espacio de nombres no es más que un identificador que permite organizar de modo estanco las clases que estén contenidas en él así como otros espacios de nombres.

Así, por ejemplo, todo lo que tiene que ver con el manejo de estructuras de datos XML en la plataforma .NET se encuentra bajo el espacio de nombres **System.Xml**. La funcionalidad fundamental para crear aplicaciones Web está en el espacio de nombres ***System.Web***. Éste a su vez contiene otros espacios de nombres más especializados como ***System.Web.Caching*** para la persistencia temporal de datos, ***System.Web.UI.WebControls***, que contiene toda la funcionalidad de controles Web para interfaz de usuario, etc...

Las clases que nos brinda la plataforma.Net se pueden observar en el examinador de objetos, cada clase está especializada en diferentes opciones:-Ver –Otras ventanas –Examinador de Objetos o simplemente –Ver –Examinador de Objetos. (aquí podemos ver todas las bibliotecas o espacios de nombre de la plataforma .net framework y al seleccionarla nos muestra que DLL la implementa, dentro de ellas contienen los espacios de nombre que la conforman y dentro de estos están las clases.)

También podemos agregar nuevas clases y bibliotecas, para ello vamos a las propiedades del proyecto y agregamos una referencia, luego se nos habre un dialogo que muestra las bilbiotecas.net que están en nuestro disco duro, las agregamos y las podemos ver en el examinador de objetos.

Ojo: Para utilizar los espacios de nombre que se observan en el examinador de objetos se debe agregar una referencia a alguna de las clases que contienen desde nuestro código para poderlo utilizar.

Ejemplo: observen y generen este código:

Module Module1

Sub Main()

Dim i As Integer

Dim s As String = ""

For i = 32 To 255

s += Chr(i)

Next

Console.WriteLine(s)

Console.Read()

End Sub

End Module

Ahora usemos una clase especializada que está dentro del espacio de nombre systes.text que se encuentra dentro de la biblioteca mscorlib, esta clase se llama StringBuilder(posee una serie de métodos especializados en concatenar cadenas de manera más eficaz que sumarlas como en el código anterior), ahora modifiquemos lo siguiente:

Sustituimos la variable s que era una cadena normal y corriente por un stringbuilder, para ello escribo la variable s junto con la ruta completa del StringBuilder dentro del espacio de nombre asi:

Dim s As New System.Text.StringBuilder

Para evitar la pesadez del código y tener que escribir muchas veces el SpaceName, Visual Studio nos ofrece la opción de usar la instrucción Imports.

Imports nos permite indicarle al compilador que vamos a hacer uso de un determinado espacio de nombre para evitar poner la ruta completa para utilizar las clases que este contiene. Entonces solo encabezamos nuestro código con imports system.text para que dentro de la declaración de la variable s de nuestro ejemplo no haya que ponerle system.Text sino simplemente Stringbuilder.

Además sustituimos la suma por un método Append de StringBuilder que permite añadir un fragmento de cadena o carácter al final de esa cadena que estamos montando. El código quedaría asi:

Module Module1

Sub Main()

Dim i As Integer

Dim s As New System.Text.StringBuilder()

For i = 32 To 255

s.Append(Chr(i))

Next

Console.WriteLine(s.ToString())

*Este ToString es un metodo que nos permite coger los fragmentos que le hemos proporcinado y los convierte en una cadena, mostrandolo en pantalla.*

Console.Read()

End Sub

End Module

# **Acceso a datos con ADO.NET**

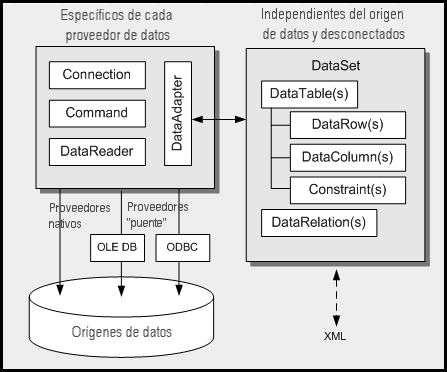
El acceso a fuentes de datos es algo indispensable en cualquier lenguaje o plataforma de desarrollo. La parte de la BCL que se especializa en el acceso a datos se denomina de forma genérica como ***ADO.NET***.

ADO.NET (ActiveX Data Objects): Es un modelo de acceso mucho más orientado al trabajo desconectado de las fuentes de datos de lo que nunca fue ADO. Si bien este último ofrecía la posibilidad de desconectar los Recordsets y ofrecía una forma de serialización de estos a través de las diferentes capas de una aplicación, el mecanismo no es ni de lejos tan potente como el que nos ofrece ADO.NET.

El objeto más importante a la hora de trabajar con el nuevo modelo de acceso a datos es el **DataSet**. Sin exagerar demasiado podríamos calificarlo casi como un motor de datos relacionales en memoria. Aunque hay quien lo asimila a los clásicos Recordsets su funcionalidad va mucho más allá.

## Arquitectura de ADO.NET

El concepto más importante que hay que tener claro sobre ADO.NET es su modo de funcionar, que se revela claramente al analizar su arquitectura:



Arquitectura de ADO.NET

Existen dos capas fundamentales dentro de su arquitectura: la ***capa conectada*** y la ***desconectada***.

### Capa conectada

La primera de ellas contiene objetos especializados en la conexión con los orígenes de datos. Así, la clase genérica ***Connection*** se utiliza para establecer conexiones a los orígenes de datos. La clase ***Command*** se encarga de enviar comandos de toda índole al origen de datos. Por fin la clase ***DataReader*** está especializada en leer los resultados de los comandos mientras se permanece conectado al origen de datos.

La clase ***DataAdapter*** hace uso de las tres anteriores para actuar de puente entre la capa conectada y la desconectada.

Estas clases son abstractas, es decir, no tienen una implementación real de la que se pueda hacer uso directamente. Es en este punto en donde entran en juego los ***proveedores de datos***. Cada origen de datos tiene un modo especial de comunicarse con los programas que los utilizan, además de otras particularidades que se deben contemplar. Un proveedor de datos de ADO.NET es una implementación concreta de las clases conectadas abstractas que hemos visto, que hereda de éstas y que tiene en cuenta ya todas las particularidades del origen de datos en cuestión.

Así, por ejemplo, las clases específicas para acceder a SQL Server se llaman **SqlConnection**, **SqlCommand**, **SqlDataReader** y **SqlDataAdapter** y se encuentran bajo el espacio de nombres **System.Data.SqlClient**. Es decir, al contrario que en ADO clásico no hay una única clase Connection o Command que se use en cada caso, si no que existen clases especializadas para conectarse y recuperar información de cada tipo de origen de datos.

Nota:  
El hecho de utilizar clases concretas para acceso a las fuentes de datos no significa que no sea posible escribir código independiente del origen de datos. Todo lo contrario. La plataforma .NET ofrece grandes facilidades de escritura de código genérico basadas en el uso de herencia e implementación de interfaces. De hecho la versión 2.0 de .NET ofrece grandes novedades específicamente en este ámbito.

Existen ***proveedores nativos***, que son los que se comunican directamente con el origen de datos (por ejemplo el de SQL Server o el de Oracle), y **proveedores "puente"**, que se utilizan para acceder a través de ODBC u OLEDB cuando no existe un proveedor nativo para un determinado origen de datos.

**Nota**:  
Estos proveedores nativos, si bien muy útiles en determinadas circunstancias, ofrecen un rendimiento menor debido a la capa intermedia que están utilizando (ODBC u OLEDB). Un programador novel puede sentir la tentación de utilizar siempre el proveedor puente para OLEDB y así escribir código compatible con diversos gestores de datos de forma muy sencilla. Se trata de un error y siempre que sea posible es mejor utilizar un proveedor nativo.

### Capa desconectada

Una vez que ya se han recuperado los datos desde cualquier origen de datos que requiera una conexión ésta ya no es necesaria. Sin embargo sigue siendo necesario trabajar con los datos obtenidos de una manera flexible. Es aquí cuando la capa de datos desconectada entra en juego. Además, en muchas ocasiones es necesario tratar con datos que no han sido obtenidos desde un origen de datos relacional con el que se requiera una conexión. A veces únicamente necesitamos un almacén de datos temporal pero que ofrezca características avanzadas de gestión y acceso a la información.

Por otra parte las conexiones con las bases de datos son uno de los recursos más escasos con los que contamos al desarrollar. Su mala utilización es la causa más frecuente de cuellos de botella en las aplicaciones y de que éstas no escalen como es debido. Esta afirmación es especialmente importante en las aplicaciones Web en las que se pueden recibir muchas solicitudes simultáneas de cualquier parte del mundo.

Finalmente otro motivo por el que es importante el uso de los datos desconectado de su origen es la transferencia de información entre capas de una aplicación. Éstas pueden encontrarse distribuidas por diferentes equipos, e incluso en diferentes lugares del mundo gracias a Internet. Por ello es necesario disponer de algún modo genérico y eficiente de poder transportar los datos entre diferentes lugares, utilizarlos en cualquiera de ellos y posteriormente tener la capacidad de conciliar los cambios realizados sobre ellos con el origen de datos del que proceden.

Todo esto y mucho más es lo que nos otorga el uso de los objetos ***DataSet***. Es obvio que no se trata de tareas triviales, pero los objetos DataSet están pensados y diseñados con estos objetivos en mente. Como podremos comprobar más adelante en este curso es bastante sencillo conseguir estas funcionalidades tan avanzadas y algunas otras simplemente usando de manera adecuada este tipo de objetos.

**Nota**:  
Otra interesante característica de los DataSet es que permiten gestionar simultáneamente diversas tablas (relaciones) de datos, cada una de un origen diferente si es necesario, teniendo en cuenta las restricciones y las relaciones existentes entre ellas.

Los DataSet, como cualquier otra clase no sellada de .NET, se pueden extender mediante herencia. Ello facilita una técnica avanzada que consiste en crear tipos nuevos de DataSet especializados en la gestión de una información concreta (por ejemplo un conjunto de tablas relacionadas). Estas nuevas tipos clases se denominan genéricamente **DataSet Tipados**, y permiten el acceso mucho más cómodo a los datos que representan, verificando reglas de negocio, y validaciones de tipos de datos más estrictas.

**VBScript.-**

Por medio de VBScript podemos crear nuestras páginas ASP, debemos tener en cuenta que antes de escribir cualquier código ASP debemos abrir nuestro código con el símbolo '<%' y al finalizar cerrarlo  
con '%>'

|  |
| --- |
| <% 'este es un comentario Response.Write ("Mi Primer Pagina ASP") %> |

VBScript es un lenguaje script derivado de Visual Basic así que también hereda sus funciones básicas. a continuación veremos algunas de las más utilizadas.

**VBScript - Funciones Básicas**

Si estas familiarizado con Visual Basic podes pasar esta parte dado que los funciones básicas son iguales en ambos lenguajes, y si no conoces Visual Basic acá va una ayuda.

**Variables**

La forma de declaran variables es por medio de la palabra 'Dim', aclaremos que la declaración de variables no es necesaria, salvo que al inicio de nuestra página ASP pongamos la palabra clave 'Opción Explicit' que hace que la declaración de variables sea necesaria si no nos daría un error.

|  |
| --- |
| <% Opción Explicit  Dim I 'declaramos la variable  I= 14 ' le asignamos un valor Response.Write (I) 'mostramos su valor en el navegador %> |

**Condicionales**

El condicional más utilizado en cualquier lenguaje es el famoso IF, VBScript no podía quedarse afuera, el modo de uso es poner la palabra clave 'If' después la condición a evaluar, luego 'Then' y el código a ejecutar si se cumple la condición dada, al finalizar se cierra el bloque 'If' con la palabra clave 'End If'

|  |
| --- |
| <% Dim I  I = 14  If I = 14 Then Response.Write ("La variable I es igual a 14") End If %> |

También existe la alternativa 'Else' (sino), es decir si no se cumple la condición hacer lo siguiente, veamos un ejemplo

|  |
| --- |
| <% Dim I  I = 14  If I = 14 Then Response.Write ("La variable I es igual a 14") Else Response.Write ("La variable I es distinta a 14")  End If %> |

Fíjense que pusimos 'I = 14' también podemos hacer otro tipo de condición como puede ser distinto 'I <> 14', mayor 'I > 14', menor 'I < 14', mayor o igual 'I >= 14', menor o igual 'I <= 14'

**Bucles**

VBScript posee un bucle llamador 'For' que repite una acción la cantidad de veces indicada.

|  |
| --- |
| <% Dim I  For I = 1 to 5  Response.Write ("Esto se repite 5 veces") Next %> |

Podemos modificar el número 5 y poner un 10 y lo repetirá 10 veces, también podemos modificar el numero 1 por el 3 y el 5 por el 6, esto hará que el bucle se repita 4 veces, '3 to 6' (desde 3 hasta 6), también podemos indicar el incremento que tendrá por medio de la palabra clave 'step'

|  |
| --- |
| <% Dim I  For I = 1 to 10 step 2 Response.Write ("Esto se repite de dos en dos, i = " & i) Next %> |

Gracias a esto podemos hacer un bucle inverso por ejemplo de 10 a 1

|  |
| --- |
| <% Dim I  For I = 10 to 1 step -1 Response.Write ("i = " & i) Next %> |

La palabra clave 'Next' indica el final del bucle, es decir se repite todo lo que está entre el 'For' y 'Next'

While...Wend

Este bucle a diferencia del 'For' repite una acción siempre que se cumpla una condición, veamos un ejemplo

|  |
| --- |
| <% Dim I  I = 0 'inicializa i igual a 0  While I <> 10 'repita mientras i sea distinto a 10 Response.Write ("i = " & i) 'muestra por pantalla el valor de i I = I + 1 'incrementa i en uno End While %> |

***“LA INTELIGENCIA CONSISTE NO SOLO EN EL CONOCIMIENTO SINO TAMBIEN EN LA DESTREZA DE APLICAR LOS CONOCIMIENTOS EN LA PRACTICA”. Aristóteles***

***NOTA: Este glosario es solo un introductorio, el estudiante debe fortalecer estos conceptos a medida que se van viendo las clase y anexarlos al portafolio.***

**FIRMA DEL ESTUDIANTE FIRMA Y CEDULA DEL ACUDIENTE**

**FIRMA DEL INSTRUCTOR**