CAPÍTULO 2 – ASP (ACTIVE SERVER PAGES)

2.1.- Introducción.

El lenguaje de marcas HTML fue diseñado exclusivamente para dar formato a los contenidos textuales presentes en un documento Web, aunque posteriormente se ampliaran las funcionalidades de sus etiquetas para permitir con ellas maquetar adecuadamente las páginas.

Con este planteamiento, los únicos medios de los que disponía el diseñador Web para interactuar con los visitantes de sus páginas eran los hiperenlaces y el uso de formularios, aunque con estos últimos la interacción se limitaba a poder recoger de una forma plana los datos introducidos en sus campos.

Buscando romper esta estaticidad de las páginas Web diferentes empresas y desarrolladores fueron introduciendo diversas tecnologías y lenguajes de script que permitían una interacción más amplia con el usuario. La primera de ellas digna de consideración fueron los script CGI, que mediante el uso de código escrito en C, C++ o Perl y ejecutado en el servidor Web permitieron recuperar los datos de los formularios de forma dinámica y actuar en tiempo real de una forma u otra dependiendo de los mismos.

En 1995 la empresa Microsoft lanzó al mercado su propio servidor Web, denominado Internet Information Server, IIS, que incluía una pieza fundamental para el posterior desarrollo de la tecnología ASP: el ISAPI, o Internet Server Aplication Program Interface, conjunto de funciones que IIS ponía a disposición de los desarrolladores de bibliotecas mediante la que estos podían escribir código ejecutable en el servidor Web, que se activara cuando el cliente solicitaba una página con una determinada extensión.

La tecnología ASP (Active Server Page – Página Activa en el Servidor) vio la luz con la versión 3.0 de IIS, en diciembre de 1996, estando actualmente por su versión 3.0 (distribuido con IIS 5.0), y no es más que una DLL de tipo ISAPI (Internet Server API, API para servidores de Internet) que es invocada automáticamente por los archivos de extensión ".asp"

ASP no es en sí mismo un lenguaje de programación, si no más bien un marco sobre el que se construyen aplicaciones basadas en Internet, apoyándose para ello en el lenguaje HTML, en lenguajes de script conocidos (generalmente VBScript, pero también JavaScript –Jscript para Microsoft-, Perl, etc.), en motores de bases de datos y en el lenguaje de consulta SQL.

Debido a esta conjunción de tecnologías diferentes es más acertado considerar una aplicación ASP en su conjunto que una serie de páginas Web aisladas.

Active Server Pages es una tecnología propietaria de Microsoft que permite de forma sencilla la ejecución de programas en un servidor. Para ello, se emplea comúnmente VBScript que es un lenguaje de secuencias orientado a objetos, rápido, portátil e interpretado que procesa código fuente incrustado directamente en páginas HTML. No es un lenguaje compilado, sino que el servidor HTTP procede a su interpretación cada vez que se produce una petición al servidor de una sus páginas
Las aplicaciones construidas con ASP serán, por tanto, más lentas que otras aplicaciones compiladas, como CGI. Sin embargo, su fácil manejo y la gran disponibilidad de funciones y herramientas de desarrollo hacen de este lenguaje uno de los más utilizados en la actualidad.
Para correr aplicaciones ASP bajo sistemas con Windows 95 o 98 es necesario emplear el servidor Personal Web Server, el cual está incluido en el CD de instalación de Windows 98. Si trabajamos en sistemas Windows NT, 2000 ó XP hay que usar el IIS (Internet Information Server). Hoy en día, es posible correr páginas ASP bajo servidores Unix/Linux, utilizando algún software como puede ser Instant ASP o Chilisoft (Sun Java System Active Server Pages), o bien empleando un módulo para ASP en Apache.

Una de las características más importantes de las páginas ASP es la posibilidad de conectar con diferentes tipos de bases de datos para extraer, agregar y eliminar datos de ellas, y generar páginas con esos datos. Estas páginas se generan en forma dinámica, dependen de las sentencias que se establezcan, para obtener los resultados del proceso realizado. Pueden conectarse a motores de bases de datos SQL, Access, Oracle, y a cualquier otro con soporte de conexión ODBC.

Las aplicaciones escritas, se guardarán en ficheros con extensión `.asp` y pueden ser invocadas mediante un formulario HTML o bien mediante un hiperenlace. El archivo que contiene la aplicación ASP contendrá en general elementos HTML. De hecho, es posible renombrar los archivos `.html` o `.htm` a archivos `.asp` de forma que si el archivo no contiene funciones ASP, el servidor prescinde del proceso de secuencia de comandos y envía el archivo tal cual al cliente. Por tanto, es posible asignar a los archivos la extensión `.asp` incluso si no se piensan agregar funciones ASP hasta más adelante.
Aquellas partes escritas en VBScript deben ir encerradas entre las etiquetas `<%` y `%>`

Ejemplo:

```
<HTML>
<HEAD><TITLE>Ejemplo de VBScript</TITLE></HEAD>
<BODY>
<H1>Página de ejemplo ASP</H1>
<BR>
<% Response.Write("Hola, mundo" ) %>
<HTML>
```

2.2.- Variables en ASP.

En VBScript sólo existe un tipo de datos llamado Variant. Cuando creamos una variable no indicamos el tipo de datos que va a contener, puesto que puede ser potencialmente cualquier cosa.
No será necesario hacer una declaración explícita de las variables que vayan a usarse, salvo que al inicio de la página ASP añadamos la orden:

```
<% Option Explicit %>
```

Para declarar una variable en VBScript se utilizan 3 tipos de instrucciones:

Dim, que declara una variable general. Esta variable es accesible en todo momento sólo si se declara fuera de cualquier función. Si se ha declarado dentro de una función, sólo será asequible por ella. Es decir, una variable declarada con Dim puede ser pública o privada, dependiendo de dónde se ha declarado.

Public, que declara una variable pública, accesible por todas las funciones del código.

Private, que declara una variable privada, sólo accesible por la función que la maneja o contiene, y dentro de la cual se ha declarado. Si se declara fuera de toda función, no será accesible por ninguna de ellas.

Si inicializamos una variable sin haberla declarado previamente, adoptará por defecto el carácter Dim, por lo que si se inicializa dentro de una función, sólo será accesible por ella.

Respecto a los nombres de las variables, estos pueden ser una letra o grupo de ellas. Pueden ser numéricos, alfanuméricos (letras y números) o una cadena de texto, teniéndose que respetar los siguientes puntos:

- deben comenzar por un carácter alfabético.
- no pueden contener puntos (.)
- no pueden superar los 255 caracteres.
- no pueden contener el carácter 'ñ', ni otro tipo de caracteres no anglosajones, ni letras reservadas, como '&', ni llamarse como palabras reservadas por el lenguaje, como Dim.
- debe ser única en su ámbito.

Una vez declarada una variable, hay que inicializarla, asignándole un valor; si no se hace así, se le asignará por defecto un valor de tipo NULL.

Visual Basic Script no distingue entre mayúsculas y minúsculas.

Lo que sí existe en VBScript son los subtipos, que mostramos a continuación:

Subtipo	Descripción
Empty	La variable Variant no está inicializada. El valor es 0 para variables numéricas o una cadena de longitud cero ("") para variables de cadena.
Null	La variable Variant, de manera intencionada, no contiene ningún dato válido.
Boolean	Subtipo que contiene valores lógicos True o False.
Byte	Contiene un número entero entre 0 y 255.
Integer	Contiene un número entero entre -32.768 y 32.767.
Currency	De -922.337.203.685.477,5808 a 922.337.203.685.477,5807.
Long	Contiene un número entero entre -2.147.483.648 y 2.147.483.647.
Single	Contiene un número de punto flotante de precisión simple comprendido entre -3,402823E38 y -1,401298E-45 para valores negativos y entre 1,401298E-45 y 3,402823E38 para valores positivos.
Double	Contiene un número de punto flotante de precisión doble comprendido entre -1,79769313486232E308 y 4,94065645841247E-324 para valores negativos y entre 4,94065645841247E-324 y 1,79769313486232E308 para valores positivos.
Date (Time)	Contiene un número que representa una fecha entre el 1 de enero del año 100 y el 31 de diciembre del año 9999. Para guardar fechas en una variable, se emplea el formato mm/dd/aaaa, introducido entre los símbolos #.
String	Contiene una cadena de longitud variable que puede ser de hasta aproximadamente 2 mil millones de caracteres.
Object	Contiene un objeto.
Error	Contiene un número de error.

Tabla 2.1: Subtipos en ASP.

2.3.- Comentarios en ASP.

Existen dos formas de comentar el código escrito en ASP, mediante la comilla simple ‘ , o bien la palabra reservada ‘rem’. En ambos casos el comentario sólo es válido en esa misma línea.

```
<%
     ' Esto es un comentario

     rem Esto es un comentario

%>
```

Los comentarios se eliminan al procesar la secuencia de comandos, por lo que no se envían al explorador cliente.

2.4.- Operadores en ASP.

Los principales operadores de VBScript aparecen recogidos en la siguiente tabla:

descripción	símbolo	ejemplo
ARITMÉTICOS		
suma	+	15 + 4 (=19)
resta	-	25 - 10 (=15)
multiplicación	*	4 * 7 (=21)
división entera	\	38 \ 4 (=9)
división normal	/	15 / 4 (=3.75)
módulo (resto de la división entera)	Mod	17 Mod 3 (=2)
potencia o exponencial	^	5 ^ 2 (=25)
menos unario	-	-(5+3) (=-8)
CONCATENACIÓN		
concatenación de cadenas	&	"HTML" & "Web" (="HTMLWeb")
COMPARACIÓN		
igual que	=	5 = 5 (=true)
distinto de	<>	5 <> 3 (=true)
menor que	<	7 < 1 (=false)
mayor que	>	7 > 1 (=true)
menor o igual que	<=	4 <= 5 (=true)
mayor o igual que	=>	6 >= 8 (=false)
equivalencia entre objetos	Is	Set A = C Set B = C A Is B (=true)
LÓGICOS		
negación (NO lógico)	Not	3 not 2 (=true)
conjunción (Y lógico)	And	(3<4) and (4<2) (=false)
disyunción (O lógico)	Or	(3<4) or (4<2) (=true)
disyunción exclusiva (O lógico exclusivo)	Xor	(3<4) or (4<6) (=false)
equivalencia lógica	Eqv	(3<4) Eqv (3>4) (=false)
implicación lógica	Imp	(3 = 3) Imp (3 <> 4) (=true)

Tabla 2.2: Operadores en ASP.

2.5.- Estructuras de control básicas.

Se pueden clasificar en dos grandes grupos: bifurcaciones condicionales y bucles. Veamos cada una de ellas.

2.5.1.- Bifurcaciones condicionales.

- Estructura If…Then…Else…End If

  Provoca que el servidor sirva los comandos incluidos en una u otra bifurcación. La sintaxis básica es:

```
If condición Then
. . . .
(instrucciones)
. . . .
[ Else
. . . .
(instrucciones alternativas)
. . . . ]
End If
```

  Donde 'condición' es algún tipo de condición que pueda ser evaluada como True o False.

- Estructura Select…Case

  Permite tomar diferentes decisiones en función del valor que tome una variable cualquiera. Es decir, nos permite ejecutar un conjunto de sentencias dado para cada valor de la variable. Su sintaxis general es:

```
Select case expresión
case valor_1_expresión
. . . .
(instrucciones)
. . . .
case valor_2_expresión
. . . .
(instrucciones)
. . . .
. . . .
case valor_n_expresión
. . . .
(instrucciones)
. . . .
End Select
```

2.5.2.- Bucles.

- Estructura For…To…Next

Repite las sentencias (tanto de ASP como HTML) incluidas dentro del bucle un número determinado de veces. Es la estructura de bucle más estricta. Su sintaxis es:

```
For contador = valor_inicial To valor_final [Step incremento]
. . . . (instrucciones)
. . . . (instrucciones)
Next
```

Si no se indica el paso (Step), será 1 por defecto.

- Estructura While…Wend

  Esta estructura permite ejecutar un conjunto de instrucciones mientras se cumpla una condición, no siendo necesario conocer de antemano el número de veces que se tiene que iterar. Presenta la siguiente sintaxis:

```
While(condición)

   sentencias a ejecutar

Wend
```

- Estructura Do While…Loop

  Es el bucle más flexible de todos, siendo su sintaxis la mostrada a continuación:

```
Do [While/Until (condición)]
      ...
   sentencias a ejecutar
      ...

Loop [While/Until (condición)]
```

  Puede observarse como la condición se puede establecer al principio del bucle o al final, asimismo podemos especificar que se ejecuten las instrucciones del bucle mientras se cumpla la condición (While) o mientras no se cumpla (Until).

2.6.- Funciones propias en ASP.

Con el objeto de reutilizar el código, es posible desarrollar funciones propias en ASP. A diferencia de los procedimientos, las funciones devuelven un valor, que será de tipo Variant. Para definir una función se utiliza la siguiente sintaxis general:

```
Function nombre_funcion (argumento1,argumento2,...,argumentoN )
...
...
nombre_funcion = resultado
End Function
```

Se declara mediante la palabra reservada 'Function', seguida de un nombre identificador único para ella y de unos paréntesis dentro de los que se incluyen los parámetros que la función necesita para ejecutar sus instrucciones. Mediante la sentencia 'End Function', se finaliza y delimita el código de la función. Si la función no precisa argumentos, se colocan paréntesis vacíos.

2.7.- Funciones del lenguaje.

En VBScript existe una librería de funciones predefinidas que el usuario puede emplear si lo considera conveniente. Al estar predefinidas no es necesario hacer nada para cargarlas en memoria. Se cargan automáticamente, con el intérprete. Solo es necesario invocarlas donde se necesiten. Estas funciones se clasifican en grupos en base al tipo de datos con el que se emplean. En los siguientes subapartados veremos aquellas funciones de mayor utilidad.

2.7.1.- Funciones relacionadas con las cadenas de caracteres.

Len(cadena_car)

Devuelve un entero que representa la longitud de la cadena cadena_car.

Left(cadena_car, n_caracteres)

Devuelve los n_caracteres más a la izquierda de la cadena cadena_car. Idéntica sintaxis tiene la función Right(), que devuelve los situados más a la derecha.

UCase(cadena_car)

Devuelve una cadena de caracteres que contiene las mismas letras que cadena_car, pero con todas las letras en mayúsculas. La función LCase(), de idéntica sintaxis, devuelve todo en minúsculas.

InStr([comienzo], cad_1, cad_2)

Devuelve la posición en la que la cadena cad_2 aparece por primera vez dentro de la cadena cad_1, comenzando desde la posición especificada en comienzo. Si se omite el argumento comienzo, la comparación tiene lugar desde el primer carácter (posición 1).

StrComp(cad_1, cad_2 [, tipo])

Devuelve cero sólo si las cadenas de caracteres cad_1 y cad_2 son iguales carácter a carácter (es decir, se distingue entre mayúsculas y minúsculas).

Si la primera es mayor que la segunda devuelve un 1, y si la segunda es mayor que la primera devuelve un -1. Si alguna cadena tiene un valor Null, devuelve Null.

La comparación entre cadenas se realiza en función de los caracteres que la forman. En este sentido un carácter es mayor o menor que otro en función de sus códigos ASCII. Así, la 'a' es mayor que la 'A', porque el código ASCII de la 'A' es 65 y el de la 'a' es 97.

El argumento 'tipo' es opcional, y es un valor numérico que indica el tipo de comparación que se va a utilizar cuando se evalúen cadenas. Si vale 0 la comparación de hará en forma binaria y si vale 1 se hará en forma textual. Si se omite, se realiza una comparación binaria.

Trim(cadena)

Recibe como argumento una cadena, variable que la contenga o expresión de cadena válida, retornando dicha cadena pero sin espacios en blanco en su inicio ni en su fin. Es decir, elimina los espacios en blanco que haya al principio y al final de la cadena. Si cadena contiene Null se devuelve Null.

Split(cadena [,delimitador, número, comparar])

Su misión es retornar una matriz unidimensional con base cero que contiene un número de subcadenas especificado. Es decir, dada una cadena formada por subcadenas que se encuentran acotadas mediante delimitadores, la función Split forma una matriz cuyos elementos son las diferentes subcadenas entre delimitadores. Los argumentos que posée son:

- cadena: obligatorio. Es una cadena, variable que la contenga o expresión de cadena válida, formada por subcadenas y delimitadores de las mismas. Si es una cadena de longitud cero, retorna una matriz vacía, es decir, una matriz sin elementos ni datos.

- delimitador: opcional. Carácter de cadena utilizado para identificar límites de subcadenas. Si se omite, se toma por defecto el carácter de espacio (" ") como delimitador. Si delimitador es una cadena de longitud cero (""), se devuelve una matriz de elemento único que contiene cadena completa, sin cambios.

- número: opcional. Número de subcadenas que se va a devolver; -1 indica que se devuelven todas las subcadenas.

- comparar: opcional. Valor numérico que indica el tipo de comparación que se va a utilizar cuando se evalúen subcadenas. Sus valores posibles son 0, que realiza la comparación en forma binaria (valor por defecto) y 1, que la realiza en forma textual.

Replace (cadena_madre, cadena_buscada, cadena_cambio [, comienzo, veces, comparación])

Su funcionalidad es buscar una subcadena dentro de una cadena madre y reemplazarla por otra. Sus argumentos son:

- cadena_madre: obligatorio. Cadena, variable que la contenga o expresión de cadena válida, que va a ser la cadena madre en la que vamos a buscar coincidencias.

- cadena_buscada: obligatorio. Cadena, variable que la contenga o expresión de cadena válida, que va a ser la subcadena que vamos a buscar dentro de la cadena madre.

- cadena_cambio: obligatorio. Cadena, variable que la contenga o expresión de cadena válida, que va a ser la subcadena que vamos a cambiar por cadena2 dentro de la cadena madre.

- comienzo: opcional. Posición dentro de cadena_madre desde donde se va a comenzar a buscar cadena_buscada. Si se omite, se supone 1. Se debe utilizar junto con veces.

- veces: opcional. Número de sustituciones de subcadena que se va a realizar. Si se omite, el valor predeterminado es -1, que significa hacer todas las sustituciones posibles. Se debe utilizar junto con comienzo.

- comparación: opcional. Valor numérico que indica el tipo de comparación que se va a utilizar cuando se evalúen subcadenas. Si es 0 (valor por defecto) se realiza una comparación de tipo binario, mientras que si es 1 se realiza una comparación textual.

Dependiendo de los resultados de la búsqueda, la función Replace puede devolver los siguientes valores:

si...	**devuelve**
cadena es de longitud cero	cadena de longitud cero ("")
cadena es Null	un error
cadena_buscada es de longitud cero	copia de *cadena*
cadena_cambio es de longitud cero	copia de cadena con todas las apariciones de cadena_buscada eliminadas
comienzo >longitud de *cadena*	cadena de longitud cero
veces = 0	copia de *cadena*

Tabla 2.3: Valores devueltos por la función Replace.

2.7.2.- Funciones de tiempo.

Time()

Devuelve la hora actual del sistema, en formato hh:mm:ss

Hour()

Devuelve un número entero entre 0 y 23, que especifica la hora del sistema.

Minute()

Devuelve un número entre 0 y 59 que indica el minuto actual del sistema.

Second()

Devuelve un número entre 0 y 59 que indica el segundo actual del sistema.

2.7.3.- Funciones de fecha.

Date()

Devuelve la fecha del sistema en el formato de fecha mm/dd/aaaa

Day(fecha)

Devuelve un número comprendido entre 1 y 31 que indica el día del mes al que corresponde la fecha contenida en fecha.

Month(fecha)

Devuelve un número comprendido entre 1 y 12 que indica el mes al que corresponde la fecha contenida en fecha.

Year(fecha)

Devuelve un número que indica el año al que corresponde la fecha contenida en fecha.

Weekday(fecha)

Devuelve un número indicando el día de la semana al que corresponde la fecha especificada en fecha (en la forma en la que se presenta en este documento, si la fecha corresponde a domingo devuelve un 1, si corresponde a lunes devuelve un 2, y así sucesivamente).

2.8.- Captura de datos de un formulario mediante ASP

Un usuario puede introducir datos en un documento HTML mediante el empleo de formularios. El pulsado por parte del usuario del botón de envío provoca la ejecución de la aplicación especificada en la opción ACTION del formulario. Si esa aplicación es una aplicación ASP, la forma de recoger los datos que el usuario ha introducido depende si en el formulario del documento HTML se ha empleado la opción de envío POST o GET.

En el caso que se haya optado por el método POST, se emplea el método Form() del objeto Request, cuya sintaxis es la siguiente:

```
valor_capturado = Request.Form(nombre_elemento)
```

donde valor_capturado es una variable donde se almacenan los datos introducidos por el usuario en el formulario en el elemento de nombre nombre_elemento.

Si el método empleado es el GET, se emplea el método QueryString(), también del objeto Request. Su sintaxis es:

```
valor_capturado = Request.QueryString(nombre_elemento)
```

donde valor_capturado y nombre_elemento tienen el mismo significado que en el método Form().

2.9.- Acceso a bases de datos con ASP

ASP permite el acceso a un buen número de bases de datos. Vamos a centrarnos en el acceso a una base de datos enlazada a un origen de datos ODBC.

Los pasos para acceder a la base de datos son los siguientes:

- Creación de una conexión
- Apertura del origen de datos ODBC
- Envío de la petición/peticiones SQL
- Recogida de los datos de la petición
- Cierre de la base de datos

En los siguientes subapartados, entramos en los detalles de cada uno de estos pasos

2.9.1.- Creación de una conexión.

Se realiza creando una instancia del objeto ADODB.Connection, de la siguiente forma:

```
Set conexion = Server.CreateObject("ADODB.Connection")
```

donde 'conexión' es el objeto creado.

2.9.2.- Apertura del origen de datos ODBC.

Supuesto que se ha creado un objeto 'conexión' como el del apartado anterior, la apertura del origen de datos se realiza invocando al método Open() del objeto, con la siguiente sintaxis:

```
conexion.Open(origen_datos_ODBC)
```

donde origen_datos_ODBC es una cadena de caracteres que representa el origen de datos ODBC que contiene las tablas que se quieren consultar/modificar. Para más detalles sobre los orígenes de datos ODBC, ver el Capítulo 5.

2.9.3.- Envío de la petición SQL.

Una vez abierto el origen de datos ODBC, es posible ejecutar en las tablas de ese origen de datos una sentencia SQL empleando el método Execute() del objeto 'conexión' creado anteriormente:

```
conexion.Execute(sentencia_SQL)
```

donde 'sentencia_SQL' es una cadena de caracteres que contiene una sentencia SQL válida.

2.9.4.- Recogida de los datos de la petición SQL.

Si la sentencia SQL debe devolver datos (por ejemplo, una sentencia de tipo SELECT) es preciso capturar los datos devueltos por el origen de datos. El método Execute() visto anteriormente, devuelve un objeto de tipo RecordSet, que contiene los datos devueltos y la estructura de los mismos. Así, basta guardar el resultado de la ejecución de la siguiente forma:

```
Set resultado = conexion.Execute(sentencia_SQL_datos)
```

donde 'sentencia_SQL_datos' es una cadena de caracteres que contiene una sentencia SQL que devuelve datos.

La extracción de datos del objeto RecordSet se realiza con una serie de métodos propios de este objeto. Puesto que el resultado de la consulta es una tabla, el objeto RecordSet es una especie de cursor que apunta a la primera

fila de la tabla. Así, para acceder a la columna n-ésima de la fila actual a la que apunta el cursor, se emplea el método Fields(), con la siguiente sintaxis:

```
valor_columna = resultado.Fields(columna)
```

donde columna puede ser, bien una cadena de caracteres que indica el nombre de la columna de la consulta n-ésima, o bien un número que indica el número de columna n–1 (la primera columna es la 0).
'valor_columna' es una variable que recoge el valor contenido en la columna n-ésima.

Para mover el cursor por las filas se utiliza el conjunto de métodos del objeto RecordSet descrito a continuación:

Método	Descripción
`MoveNext()`	Desplaza el cursor una fila hacia delante
`MovePrevious()`	Desplaza el cursor una fila hacia atrás
`MoveLast()`	Desplaza el cursor hasta la última fila
`MoveFirst()`	Desplaza el cursor hasta la primera fila
`EOF`	Devuelve True si se ha sobrepasado la última fila de la tabla devuelta por la consulta.

Tabla 2.4: Métodos de Recordset para movimiento por filas.

2.9.5.- Cierre de la conexión.

Una vez realizadas las peticiones SQL y recibidos los datos, es conveniente cerrar la conexión. Supuesto que se tiene un objeto ADODB.Connection llamado 'conexión' abierto actualmente, el cierre de la conexión se realizará de la siguiente forma:

```
conexion.Close()
```