ESCUELA TÉCNICA SUPERIOR DE INGENIERIA DE TELECOMUNCIACIÓN

UNIVERSIDA POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB MULTIMEDIA DESARROLLADA EN ASP.NET Y MySQL

AUTOR: Antonio Javier Martínez Vaillo.

DIRECTOR: Antonio Javier García Sánchez.

Enero/2007

Autor:	Antonio Javier Martínez Vaillo
Email del autor:	antoniojvaillo@hotmail.com
Director:	Antonio Javier García Sánchez
Email del Director:	antoniojavier.garcia@upct.es
Título del PFC:	Diseño e implementación de una aplicación Web Multimedia desarrollada en ASP.NET y MySQL.
Resumen:	Se ha desarrollado una aplicación web multimedia bajo el entorno .NET de Microsoft haciendo uso de sus lenguajes ASP.NET y C#. Dicha aplicación consiste en un servidor web programado con ASP.NET y en el que se ha incluido una base de datos realizado con MySQL para control del acceso de los diversos usuarios de este servicio. Además el usuario puede conectarse a una cámara situada en el servidor web.
Titulación:	Ingeniería Técnica de Telecomunicación
Especialidad:	Telemática
Departamento:	Departamento de Tecnologías de la Información y las Comunicaciones
Fecha de Presentación:	Enero de 2007

Agradecimientos

Agradecer a *Dr. Antonio Javier García Sánchez* por sus ideas, aportaciones y tiempo empleado en el desarrollo de este proyecto.

Agradezco a todos mis compañeros por haberme demostrado ser amigos antes que compañeros a lo largo de estos años. También dar las gracias a todos mis amigos por su apoyo y su ánimo constante.

Agradecer, como no, a toda mi familia por su apoyo a lo largo de todos estos años de estudio.

Índice general

Índice de figuras

1. Introducción.

La revolución de Internet que tuvo lugar a finales de los años 90 supuso un drástico cambio en la forma en la que los individuos y las empresas se comunican entre sí. Las aplicaciones tradicionales como procesadores de texto y programas de contabilidad se conciben como productos independientes, ya que permiten a los usuarios realizar operaciones utilizando para ello datos almacenados en el sistema en el que se encuentran y desde el que se ejecuta la aplicación.

Por el contrario, los nuevos programas se basan en un modelo distribuido en el que las aplicaciones colaboran entre sí para ofrecer sus servicios y compartir sus funciones.

Como resultado, el papel principal de los nuevos programas se centra en la compatibilidad de intercambio de información (por medio de servidores y navegadores Web), colaboración (por medio de correo electrónico y mensajes instantáneos) y expresiones individuales (a través de registros Web, conocidos como Blogs y e-zines, revistas basadas en la Web). Básicamente, el nuevo software ha pasado de ofrecer una discreta funcionalidad a ofrecer servicios mucho más completos.

La estructura .NET representa un conjunto de servicios y bibliotecas unificado y orientado a objetos que engloba el nuevo papel de los programas basados y dirigidos a la red. De hecho, esta estructura es la primera plataforma diseñada esencialmente con Internet como base.

Partiendo de esta idea el objetivo de este proyecto es desarrollar, bajo este entorno, una Aplicación Web Multimedia.

Dicha aplicación consistirá en un servidor Web programado con ASP.NET y en el que se incluirá una base de datos realizada en MySQL para control del acceso de los diversos usuarios de este servicio.

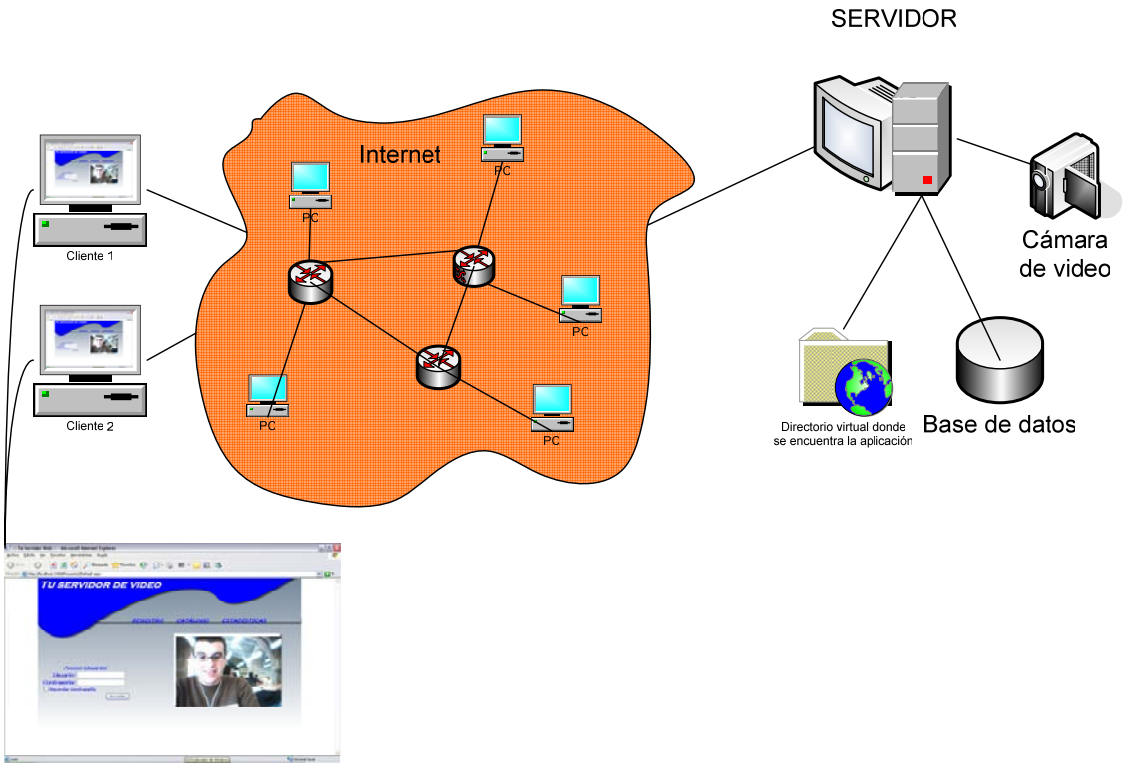
Además, las implementaciones realizadas deben tener en cuenta los conceptos referidos a la seguridad en la transmisión/recepción de información y autentificación de usuarios.

La opción de realizar una aplicación Multimedia se hace evidente ante la importancia que van obteniendo las aplicaciones Web usando transmisión de video en campos tan diversos como domótica, seguridad en edificios, etc.

La combinación de la plataforma .NET y este tipo de aplicaciones proporciona un entorno novedoso de implementación.

2. Entorno de Trabajo

En la imagen siguiente se muestra el entorno de trabajo de la aplicación:



Entorno de trabajo

Tal y como se puede observar, en el entorno de trabajo se diferencian tres conceptos importantes.

Por un lado se encuentra el Servidor de la aplicación en el que se deberá crear la base de datos correspondiente, donde irá conectada la cámara Web y en donde se ubicará el directorio virtual de la aplicación.

Por otro lado se encuentran los clientes que acceden a la aplicación. Estos clientes accederán a través del navegador de su ordenador conectado a la red.

Por último se puede observar el elemento de conexión entre el servidor y los diversos clientes, esta conexión se realiza a través de Internet, aunque también se podría hacer en una red local.

3. Descripción de Componentes.

3.1. La Estructura .NET.

Desde el año 1995, Microsoft lleva realizando importantes esfuerzos para trasladar toda su atención de las plataformas basadas en Windows a las basadas en Internet. Como fruto de este enfoque, se ha desarrollado en el estándar ASP (Páginas activas de servidor), dirigidas a la programación en Internet.

No obstante, el diseño de scripts ASP (scripts interpretados) se puede considerar tradicional en comparación con la programación orientada a objetos ya existente. Es más, el código ASP sin estructurar resultaba difícil de depurar y de mantener, era preferible combinar código de lenguajes estructurados orientados a objetos, como puede ser C#, con el código ASP. No obstante, solamente se podría combinar el código C# como componente.

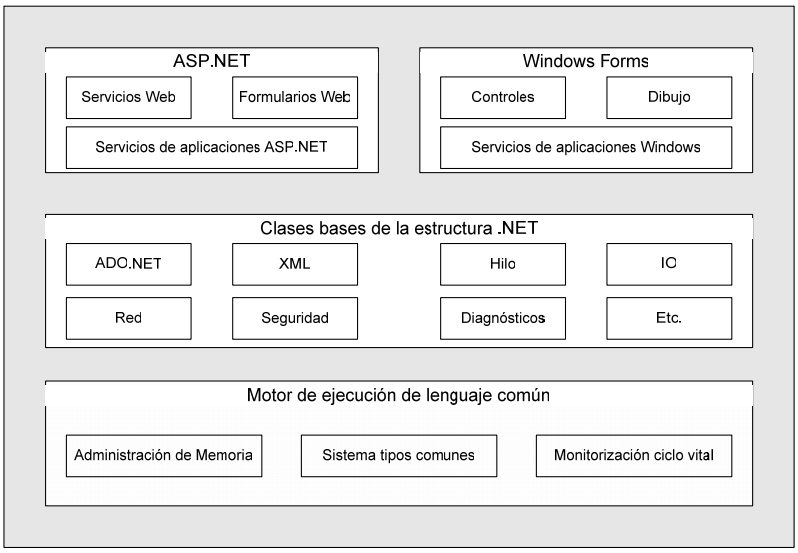
La tarea de integrar el software para el desarrollo Web resulta bastante complicada y exige que los programadores dispongan de completos conocimientos sobre tecnologías y aspectos de integración. Por esta razón, resultaba necesario el desarrollo de una estructura que permitiera el desarrollo de aplicaciones Web de forma más coherente.

Recientemente, Microsoft ha presentado la estructura .NET como iniciativa para que los programadores puedan distribuir software con funciones adaptadas y compatibles con Internet.

Esta estructura incluye numerosos lenguajes, bibliotecas de clases y una plataforma de ejecución común. Además, dispone de protocolos que permiten a los programadores integrar programas en Internet y en servidores Enterprise .NET, como puede ser SQL Server 2000, Commerce Server 2000 y BizTalk Server.

De esta forma, la plataforma .NET ofrece las mayores prestaciones en lo que a integración de software se refiere. Además, esta plataforma simplifica al máximo el desarrollo de aplicaciones de Internet.

La estructura .NET libera al programador de todos los aspectos específicos de sistemas operativos, como puede ser la administración de memoria y el procesamiento de archivos, ya que cubre todos los niveles de desarrollo de software situados por encima del sistema operativo. En la siguiente figura se muestran los distintos elementos de esta estructura:



Estructura .NET

El nivel superior representa las interfaces de usuario y de programa e incluye Formularios de Windows, Formularios y Servicios Web y Servicios de aplicaciones. Los formularios Web ofrecen una interfaz de usuario basada en la Web. Los servicios Web son las interfaces de programa más innovadoras ya que permiten la comunicación entre programas por Internet. Las interfaces de programa basadas en Internet, entre las que se incluyen los formularios y los servicios Web, se implementan por medio de ASP.NET que a su vez se integra en la estructura .NET.

El nivel intermedio representa las clases de la estructura .NET, disponibles en diversos lenguajes de forma universal. El uso de dichas clases se mantiene de una forma coherente en todos los lenguajes incluidos en la estructura .NET.

El nivel base representa la plataforma de ejecución común, denominada Entorno Común de Ejecución (CLR). Se trata del componente más importante de la estructura. Ofrece compatibilidad con varios lenguajes y permite derivar elementos entre ellos. Por ejemplo, se puede derivar una clase escrita en Visual Basic desde una clase creada en Visual C++. Con .NET, el lenguaje de programación que utilice depende del usuario. La estructura permite crear aplicaciones utilizando para ello diversos lenguajes. Esta compatibilidad multilenguaje resulta posible gracias a que el motor CLR ofrece un sistema común de tipos de datos. Además, se encarga de la gestión de memoria y evalúa la totalidad del ciclo vital de los objetos, además de realizar su seguimiento y recoger los elementos eliminados y sobrantes.

Visual Studio .NET (VS.NET) es el primer conjunto de productos basados en la estructura .NET. Incluye Visual Basic, Visual C++ y C#. Ofrece un Entorno de Desarrollo Integrado (IDE) para todos los lenguajes. Por esta razón, los programadores siempre trabajan en un mismo entorno independientemente del lenguaje que utilicen.

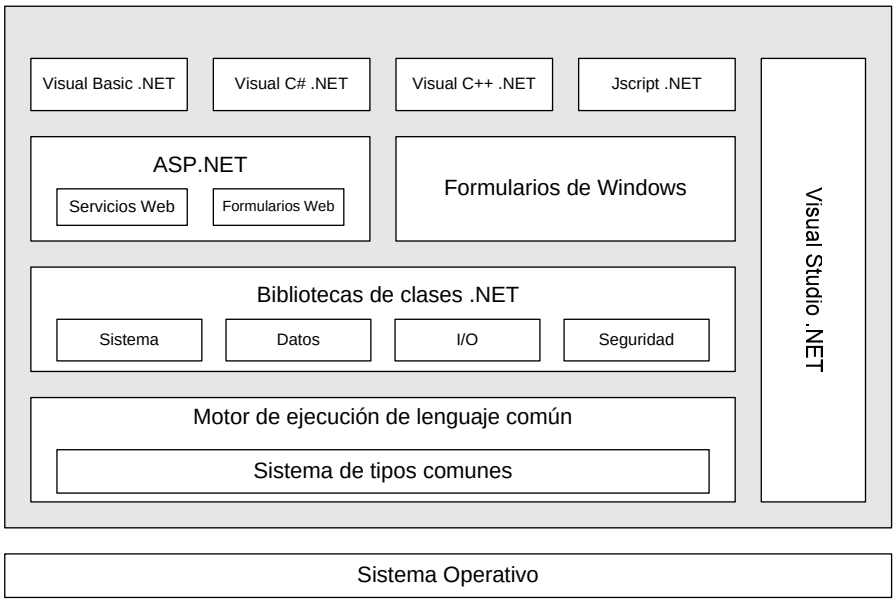
A continuación se expondrá la Arquitectura .NET Framework.

3.2. Arquitectura .NET Framework

La arquitectura .NET Framework es un conjunto de objetos y diseños de Microsoft para crear aplicaciones. .NET Framework proporciona la funcionalidad subyacente a todas las tecnologías existentes en la estructura .NET.

Todas las aplicaciones desarrolladas bajo .NET Framework tienen ciertas características distintivas que garantizan la compatibilidad, seguridad y estabilidad. Son tres los elementos fundamentales:

- Entorno común de ejecución.
- Biblioteca de clases .NET.
- Componentes de unificación.



Componentes de la estructura .NET

A continuación se describen más detalladamente cada uno de los tres elementos fundamentales de los que se compone la arquitectura .NET Framework:

3.2.1. Entorno común de ejecución.

El entorno de ejecución (CLR) es el nivel situado entre una aplicación y el sistema operativo en el que se ejecuta. Este entorno simplifica el diseño de la aplicación y reduce la cantidad de código ya que ofrece una serie de servicios de ejecución que incluye la gestión de memoria, de hilos, de duración de los componentes y el procesamiento predeterminado de errores. La principal ventaja de este entorno es que proporciona estos servicios de ejecución a todas las aplicaciones, independientemente del lenguaje de programación en el que se haya creado y sin que el programador tenga que realizar operaciones adicionales.

El entorno común de ejecución se encarga de compilar el código antes de ejecutarlo. En lugar de generar una representación binaria del código, como los compiladores tradicionales, .NET genera una representación de un lenguaje común a la estructura .NET: el Lenguaje Intermedio de Microsoft (MSIL), también denominado IL. Cuando se ejecuta un código por primera vez, el CLR invoca un compilador especial denominado JIT, que transforma el lenguaje IL en instrucciones ejecutables específicas al tipo y modelo del procesador del sistema.

3.2.1.1. Sistema de Tipos Común.

El Sistema de Tipos Común (CTS) es un componente del motor CLR que ofrece un conjunto de tipos de datos con un sistema de comportamientos común a todos ellos. En Visual Basic, por ejemplo, el tipo de datos String (Cadena) se asigna a la clase CTS System.String. Por esta razón, si un cliente Jscript.NET tiene que comunicarse con un componente implementado en VB.NET, no es necesario modificar el intercambio de información ya que el tipo es común a Jscript.NET y a VB.NET. El sistema CTS elimina los problemas de compatibilidad que pueden generarse fuera de .NET.

Los lenguajes de programación .NET recurren a las ventajas del sistema CTS para que los programadores puedan utilizar los tipos de datos propios de sus lenguajes.

3.2.2. Biblioteca de clases .NET

Para que el .NET Framework resulte más sencillo de entender y de manejar, se divide en espacios de nombres. El espacio de nombre raíz de la biblioteca de clases .NET se denomina System y contiene las clases principales y tipos de datos como Int32, Object, Array y Console. Los espacios de nombre secundarios se adjuntan al nombre System.

A continuación se muestran algunos ejemplos de espacios de nombre anidados:

- **System.Diagnostic**: Incluye clases para trabajar con el Registro de eventos.
- **System.Data**: Facilita el manejo de datos de diferentes fuentes de datos (System.Data.OleDb se encuentra en este espacio de nombre y contiene las clases ADO.NET).
- **System.IO**: Contiene clases para trabajar con flujos de archivos y datos.

Entre las ventajas de la biblioteca de clases .NET se pueden encontrar un consistente conjunto de servicios disponibles para todos los lenguajes .NET y una implementación simplificada, ya que esta biblioteca se utiliza en todas las implementaciones de la estructura .NET.

3.2.3. Componentes de Unificación

Hasta el momento, se han analizado los diferentes componentes de nivel inferior de la estructura .NET. Los componentes de unificación, que se describen a continuación, permiten acceder a los servicios que ofrece esta estructura.

3.2.3.1. ASP.NET

El componente ASP.NET ofrece dos elementos principales: Formularios Web y Servicios Web.

- **Formularios Web**: Los programadores que no estén familiarizados con el desarrollo Web pueden perder mucho tiempo en determinados aspectos de este proceso, como por ejemplo en la validación de la dirección de correo electrónico de un formulario. Esta información se puede validar por medio de un script de tipo cliente o servidor. La elección de unos de los dos tipos de script es compleja ya que cada enfoque ofrece una serie de ventajas y de inconvenientes, muchos de los cuales aparecían a menos que se realicen operaciones de diseño. Si se valida el formulario con ayuda de un código JScript de tipo cliente, es necesario considerar el navegador que los usuarios utilizarán para acceder al formulario. No todos los navegadores representan los documentos del mismo modo en una interfaz gráfica. Si se valida el formulario en el servidor, habrá que conocer la carga que los usuarios colocan en el mismo, ya que el servidor tiene que validar los datos y devolver el resultado al cliente. Los formularios Web simplifican el desarrollo Web para que incluso se pueda arrastrar y soltar los controles sobre el diseño que se utilice para editar una página y, de esta forma, crear aplicaciones Web interactivas.
- **Servicios Web**: Un Servicio Web es una aplicación que muestra una interfaz a través de métodos de accesos estándar. Este tipo de servicios se puede utilizar por otras aplicaciones y componentes y su diseño no está dirigido al uso directo por usuarios humanos. Facilitan el diseño de aplicaciones que integran elementos de recursos remotos. Por ejemplo, se puede escribir un Servicio Web que ofrezca información meteorológica a los suscriptores del servicio en lugar que los suscriptores tengan que enlazar a una página o descargar un archivo de un sitio Web. Basta con que los clientes invoquen un método del servicio Web del mismo modo que si se tratara de un método de un componente instalado en su sistema. De esta forma, pueden acceder a la información meteorológica disponible en un formato de sencillo manejo que pueden integrar en sus propias aplicaciones o páginas Web personales.

3.2.3.2. Formularios de Windows.

El nombre de Formularios de Windows hace referencia al conjunto unificado de clases compatibles con la creación de aplicaciones personales, aplicaciones que disponen de una interfaz gráfica de usuario (GUI). Este tipo de componentes facilita el diseño de aplicaciones destinadas al usuario que utilizan cualquier lenguaje de programación .NET. Es más, con ayuda de Visual Studio .NET, los programadores pueden diseñar de forma sencilla formularios por medio de las técnicas de arrastrar y soltar.

3.2.3.3. Visual Studio .NET

Se trata de un entorno de desarrollo apto para todos los lenguajes. Visual Studio .NET simplifica las labores de desarrollo en un entorno multilenguaje a través de funciones de depuración extremo a extremo entre todos los lenguajes de programación: diseños visuales para XML, HTML, datos y código de varios tipos.

Este entorno consigue tal nivel de integración ya que se basa en las funciones de la estructura .NET. El diseño de formularios Web y formularios de Windows mejora la productividad del proceso de desarrollo. La integración de las funciones de implementación mejora la productividad durante el proceso de depuración post-implementación. Algunas de las principales opciones de Visual Studio .NET son:

- **IDE**: Simplifica el desarrollo en varios lenguajes y es compatible con Visual Basic, C++, C# y JScript.NET.
- **Lista de tareas**: Organiza las tareas y gestiona los errores y advertencias en un mismo punto. Las tareas se obtienen de comentarios especializados en código fuente y se representan en forma de tablas. Se puede pulsar dos veces sobre la tarea para acceder al fragmento de código en el que se introdujo la tarea.
- **Explorador de soluciones**: Ofrece una vista jerárquica de una solución organizada en proyectos. Permite la gestión de proyectos relacionados en una misma solución.
- **Explorador de servidores**: Gestiona el equipo del usuario y el resto de equipos de una red, incluyendo recursos como SQL Server, mensajes en cola, servicios, etc. Integra el seguimiento de rendimiento y de eventos y de servicios Web.
- **Compatibilidad multimonitor**: Optimiza el uso del espacio de pantalla disponible.
- **IntelliSense**: Garantiza el cumplimiento de instrucciones de forma consistente en todos los lenguajes compatibles.
- **Ayuda dinámica**: Genera documentación en función del ámbito de trabajo actual.
- **Depuración extremo a extremo**: Facilita el proceso entre lenguajes y la depuración del sistema por medio del depurador Visual Studio .NET. La curva de aprendizaje se reduce y los programadores pueden aprovechar con mejores resultados las opciones del depurador.
- **Compatibilidad de implementación**: Integra la implementación en cada solución (proyectos). Al efectuar cambios en la solución, se actualiza la información de implementación. Se puede implementar la solución por medio de una configuración tradicional (instalación en un solo sistema), configuración Web y descarga Web. También facilita la implementación de depuración entre lenguajes.

3.3. Lenguaje C#.

Resulta muy sencillo escribir código C# en VS.NET. No es necesario recordar todas las funciones y clases que componen este lenguaje ya que el menú IntelliSense permite seleccionar las clases, métodos y funciones requeridas. Las palabras clave se visualizan en azul para facilitar las labores de lectura y compresión del código. A continuación se desarrollan algunas de las características del este lenguaje de programación:

3.3.1. Características:

- **Sencillez:** C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo:

    1. El código escrito en C# es autocontenido, lo que significa que no necesita de ficheros adicionales al propio fuente tales como fichero de cabecera o ficheros IDL.
    2. El tamaño de los tipos de datos básicos es fijo e independiente del compilador, Sistema Operativo o máquina para quienes se compile, lo que facilita la portabilidad del código.
    3. No se incluyen elementos poco útiles de lenguajes como C++ tales como macros o herencia múltiple.

- **Modernidad:** C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose que son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes como Java o C++ hay que simular, como un tipo básico decimal que permita realizar operaciones de alta precisión con reales de 128 bits, o como la inclusión de una instrucción *foreach* que permite recorrer arrays con facilidad y es ampliable a tipos definidos por el usuario.

- **Orientación a objetos:** Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

    C# soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.

    En lo referente a la encapsulación es importante señalar que aparte de los típicos modificadores *public*, *private* y *protected*, C# incluye un cuarto modificador llamado *internal*, que puede combinarse con *protected* e indica que al elemento a cuya definición procede sólo puede accederse desde su mismo ensamblado, es decir, un elemento *protected* es público dentro la propia clase y en su heredadas.

    Respecto a la herencia (a diferencia de C++ y al igual que Java), C# sólo admite herencia simple de clases, ya que la múltiple provoca más

complicaciones que facilidades, y en la mayoría de los casos su utilidad puede ser simulada con facilidad mediante herencia múltiple de interfaces.

- **Orientación a componentes:** La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas.
- **Gestión automática de memoria:** Todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR. Esto tiene efecto en el lenguaje de que no es necesario incluir instrucciones de destrucción de objetos. Sin embargo, dado que la destrucción de los objetos a través del recolector de basura sólo se realiza cuando éste se activa (ya sea por falta de memoria, finalización de la aplicación o solicitud explícita en el código), C# también proporciona un mecanismo de liberación manual de recursos a través de la instrucción *Using.* Ésta instrucción *Using* se comporta como una construcción de bloques *Try…Finally* en la que el bloque *Try* utiliza los recursos y el bloque *Finally* los desecha.
- **Seguridad de tipos:** C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto y es especialmente necesario en un entorno gestionado por un recolector de basura. Para ello se toman medidas del tipo:

    1. Sólo se admiten conversiones entre tipos compatibles. Esto es, entre un tipo y antecesores suyos, entre tipos para los que explícitamente se haya definido un operador de conversión, y entre un tipo y un tipo hijo suyo del que un objeto del primero almacenase una referencia del segundo (downcasting). Obviamente, lo último sólo puede comprobarlo en tiempo de ejecución el CLR y no el compilador, por lo que en realidad el CLR y el compilador colaboran para asegurar la corrección de las conversiones.
    2. No se pueden usar variables no inicializadas. El compilador da a los campos un valor por defecto consistente en ponerlos a cero y controla mediante análisis del flujo de control de la fuente que no se lea ninguna variable local sin que se haya asignado previamente algún valor.
    3. Se comprueba que todo el acceso a los elementos de una tabla se realice con índices que se encuentren dentro del rango de la misma.
    4. Pueden definirse métodos que admitan un numero indefinido de parámetros de un cierto tipo, y a diferencia de lenguajes como C o C++, en C# siempre se comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados.

- **Instrucciones seguras:** Para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control mas comunes. Toda condición ha de ser una expresión condicional y no aritmética, con lo que se evitan errores por confusión del operador de

igualdad (= =) con el de asignación (=), y en todo caso de un *switch* ha de terminar en un *break* o *goto* que indique cuál es la siguiente acción a realizar, lo que evita la ejecución accidental de casos y facilita su reordenación.

- **Sistema de tipos unificados:** A diferencia de *C++*, en *C#* todos los tiposde datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada "System.Object", por lo que dispondrán de todos los miembros definidos en ésta clase (es decir, serán "objetos") A diferencia de *Java*, en *C#* esto también es aplicable a los tipos de datos básicos. Además, para conseguir que ello no tenga una repercusión negativa en su nivel de rendimiento, se ha incluido un mecanismo transparente de *boxing y unboxing* con el que se consigue que sólo sean tratados como objetos cuando la situación lo requiera, y mientras tanto pueden aplicárseles optimizaciones específicas. Las conversiones *boxing y unboxing* permiten tratar los tipos de valor como objetos. La aplicación de la conversión *boxing* a un tipo de valor empaqueta el tipo en una instancia del tipo de referencia *Object*. Esto permite almacenar el tipo de valor en el montón del recolector de elementos no utilizados. La conversión *unboxing* extrae el tipo de valor del objeto. El hecho de que todos los tipos del lenguaje deriven de una clase común facilita enormemente el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.

- **Extensibilidad de tipos básicos:** *C#* permite definir, a través de estructuras, tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos. Es decir, que se puedan almacenar directamente en pila (así su creación, destrucción y acceso serán más rápidos) y se asignen por valor y no por referencia.

- **Extensibilidad de operadores:** Para facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los básicos predefinidos en el lenguaje, al igual que *C++* y a diferencia de *Java*, *C#* permite redefinir el significado de la mayoría de los operadores (incluidos los de conversión, tanto para conversiones implícitas como explícitas) cuando se apliquen a diferentes tipos de objetos. Las redefiniciones de operadores se hacen de manera inteligente, de modo que a partir de una única definición de los operadores (++) y (–) el compilador puede deducir automáticamente como ejecutarlos; y definiendo operadores simples (como +), el compilador deduce cómo aplicar su versión de asignación compuesta (+=). Además, para asegurar la consistencia, el compilador vigila que los operadores con opuesto siempre se redefinan por parejas (por ejemplo, si se redefine (= =), también hay que redefinir (!=).

- **Eficiente:** En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta con marcar regiones de código como inseguras (modificador *unsafe*) y podrán

usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad de procesamiento muy grandes.

- **Compatible:** Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes, sino que el CLR también ofrece, a través de los llamados *Platform Invocation Services* (*Pinvoke*), la posibilidad de acceder con facilidad a este tipo de funciones, ya que éstas muchas veces esperan recibir o devuelven punteros. También es posible acceder desde código escrito en C# a objetos COM. Para facilitar esto, *.NET Framework SDK* incluye unas herramientas llamadas *tlbimp* y *regasm* mediante las que es posible generar automáticamente clases proxy que permitan, respectivamente, usar objetos COM desde .NET como si de objetos .NET se tratase y registrar objetos .NET para su uso desde COM. Finalmente también se da la posibilidad de usar controles ActiveX desde código .NET y viceversa.

3.4. Protocolo HTTP.

HTTP (*HyperText Transfer Protocol*) es el protocolo más popular de Internet. Previsto inicialmente para la transferencia de archivos HTML, su uso ha evolucionado hacia la activación en el servidor de procesos que generan automáticamente contenido HTML. Por ello, un programa servidor HTTP (o servidor Web) debe integrar extensiones programables controladas por peticiones HTTP. Inicialmente dichas extensiones eran las CGI, pero después aparecieron las DDL ISAPI o NSAPI (servidores Windows) y los servlets Java, si bien finalmente llegaron los motores de script como ASP, PHP o JSP.

El desarrollo de una aplicación de Internet de debe tener presente dos características importantes del protocolo HTTP:

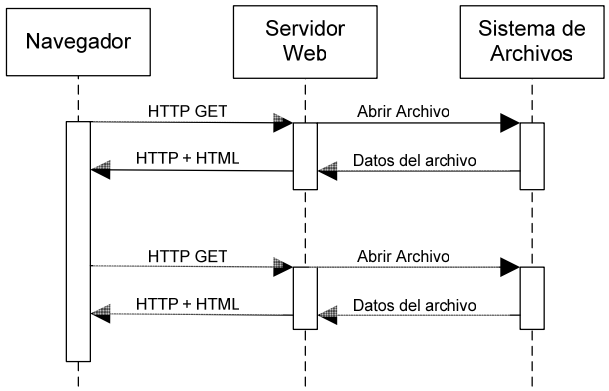
- HTTP es un protocolo desconectado.
- HTTP es un protocolo de texto.

3.4.1. HTTP es un protocolo desconectado.

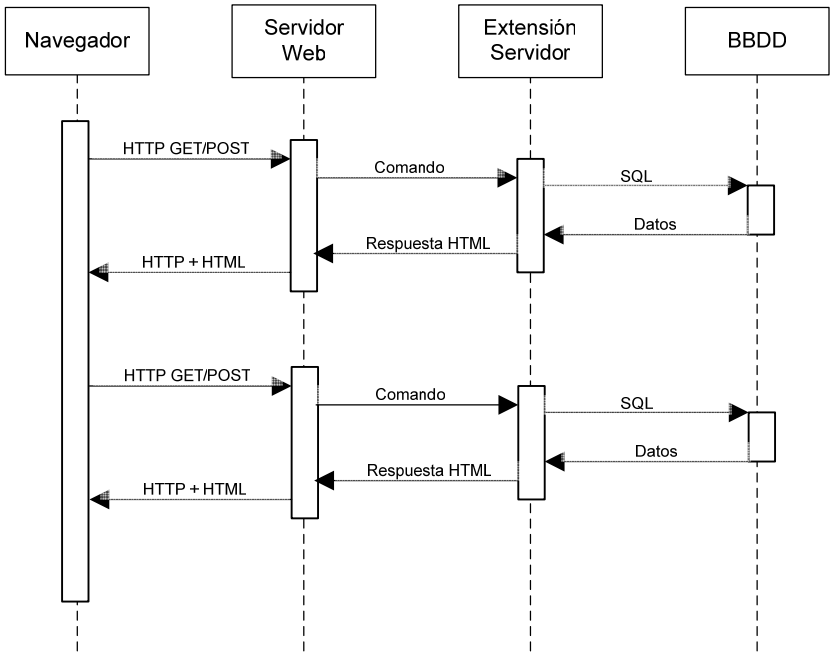
A diferencia de otros protocolos como FTP (*File Transfer Protocol*), o de otras arquitecturas cliente-servidor clásicas, HTTP no tiene noción de conexión, al menos más allá del tratamiento de una petición.

Una petición HTTP es emitida, procesada en el servidor, y es seguida de la emisión de una respuesta. El servidor Web no conserva con posterioridad ninguna información sobre la respuesta procesada. Si un mismo cliente emite una segunda petición, nada le indicará al servidor Web que se trata del mismo cliente de la respuesta anterior. Por lo tanto, será necesario implementar técnicas específicas para desarrollar una aplicación que utilice diversos formularios: cadenas de petición, campos ocultos, cookies, sesión… La implementación de dichas técnicas en ASP.NET será presentada posteriormente.

Los diagramas siguientes presentan una secuencia de dos peticiones sucesivas contra un servidor Web.



Secuencia de dos peticiones para la lectura de archivos HTML estáticos



Secuencia de dos peticiones para la generación dinámica de HTMl, a partir de los datos extraídos de una base de datos

3.4.2. HTTP es un protocolo de texto.

Las peticiones y las respuestas HTTP son codificadas en ASCII de 7bits. Los caracteres acentuados, o multi-octeto (UNICODE) y los datos binarios deben ser codificados en la emisión y descodificados en la recepción. El encabezado de la petición permite, pues, especificar la norma de codificación utilizada, así como la que se espera en respuesta. Existen distintas normas: utf-8 para los caracteres acentuados, utf-16 para UNICODE y, finalmente, MIME y base 64 para los datos binarios.

HTTP no es, por tanto, especialmente potente para la transferencia de los datos. Sin embargo sus bazas son su robustez y su universalidad que, al asociarlo con XML (*eXtensible Markup Language*), ofrece una solución eficaz para la interoperabilidad entre sistemas, base sobre la que reposa la noción de servicio Web.

3.4.3. Extensiones de servidor Web.

- **CGI:** (*Common Gateway Interface*) es la más antigua de las extensiones de servidor Web. Cuando se solicita una petición CGI el servidor instancia el programa CGI, lo ejecuta y, posteriormente, lo libera. Los datos pasan por los flujos estándar de entrada/salida. El programa CGI puede ser un archivo ejecutable compilado o un archivo de script Perl, ejecutado por su intérprete.

Todos los servidores Web, sea cual sea su plataforma, integran un extensión CGI.

- **ISAPI:** Una extensión ISAPI (*Internet Server API*) es una DLL (*Dinamic Link Lybrary*) que puede ser cargada por el servidor Web. La extensión se carga en el proceso del servidor Web, creándose un hilo (*thread*) por cada petición. El rendimiento es claramente superior al ofrecido por la CGI, debido a que la DLL es, por sistema, compilada. ISAPI requiere un servidor Web compatible, bajo Windows. Un filtro ISAPI es una extensión que procesa la petición antes de que sea procesada por el servidor Web.

- **Servlet Java:** Un servlet funciona según el mismo principio que una DLL ISAPI, si bien lo hace en el seno de un contenedor web Java que es, a su vez, extensión del servidor Web. Java es multiplataforma y existen contenedores Java para numerosos servidores web. Por ejemplo, la solución opensource Tomacat para el servidor Web Apache.

- **Monitores de script servidor:** A diferencia de los sistemas anteriores, que construyen directamente el flujo HTML de respuesta, las extensiones basadas en script utilizan páginas definidas con una plantilla HTML integrada en el código. Cuando la URL de la página es solicitada por una petición, se construye la respuesta a partir de la plantilla, completada con el resultado de la interpretación del código que contiene.

  ASP (*Active Server Pages*) es, de hecho, un intérprete VBScript dentro de una DLL ISAPI. ASP requiere el servidor web Microsoft IIS (*Internet Information Server*).

  PHP (*Php Hypertext Preprocessor*) funciona según el mismo principio que ASP, con un lenguaje de script específico. Su implementación para Apache es muy popular.

  Un script JSP (*Java Server Pages*) utiliza Java. En su primera ejecución es transformado en servlet, trabajo realizado por un servlet filtro dentro del contenedor Web Java. JSP es, de hecho, un generador de servlets.

  Zope (*Zope Object Publishing Environment*) utiliza el lenguaje Python y dispone de funcionalidades de gestión de contenidos.

3.5. ASP.NET

Internet permite a las personas de todo el mundo comunicarse entre sí mediante sus ordenadores. Esta tecnología ha presentado muchas posibilidades nuevas de comunicación, como el correo electrónico, la mensajería instantánea y el World Wide Web.

Originalmente, los sitios Web eran sencillos. Eran páginas HTML centradas en algún tema. Estas páginas eran estáticas, donde los clientes no podían comunicarse de forma alguna.

La Web evolucionó rápidamente y se agregaron nuevos niveles de funcionalidad, como imágenes, tablas y formularios, lo cual finalmente permitió que los clientes se comunicaran con los sitios Web. Los desarrolladores de sitios Web empezaron a crear otros trucos en sus sitios, como los rollovers y los menús desplegables.

Esto permitió la comunicación, pero aún no se tenía un contenido verdaderamente dinámico. Posteriormente, se presentó el procesamiento en el servidor. A partir de entonces es posible trabajar con bases de datos, procesar el contenido y determinar nuevos tipos de estadísticas de los clientes directamente en la Web.

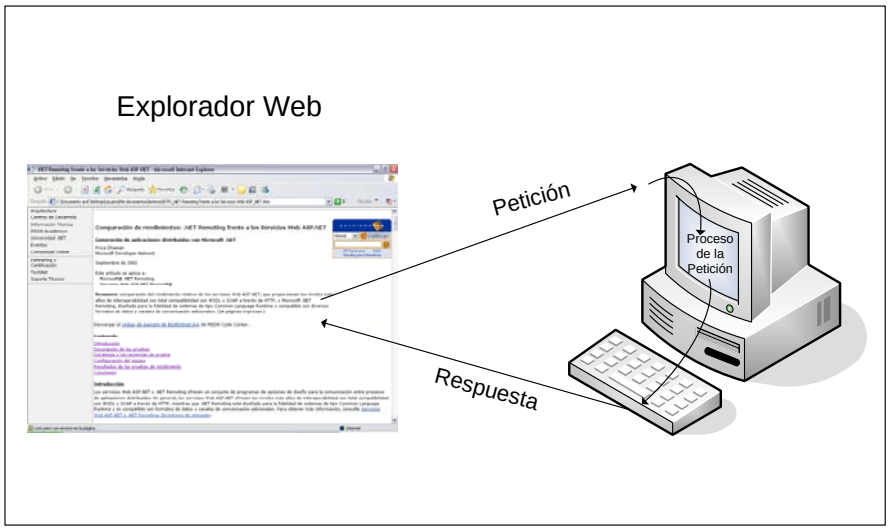
ASP.NET es una tecnología de servidor que reúne las distintas piezas de Web para dar a los desarrolladores un poder sin antecedentes. Pero antes de profundizar en ASP.NET, veamos cómo funciona el procesamiento dinámico.

3.5.1. Procesamiento dinámico.

Internet se basa en el modelo cliente-servidor, en el cual dos equipos trabajan en conjunto, intercambiando información para realizar una tarea. El ejemplo común es la comunicación entre un servidor (un equipo que contiene información) y un cliente (un equipo que solicita la información).

El cliente envía una petición de información al servidor. A continuación, éste responde al cliente con la información solicitada. A este paradigma se le conoce como modelo petición-respuesta y es una parte integral del modelo cliente-servidor.

Un servidor Web es un equipo que contiene información de un sitio Web, como sus páginas HTML, imágenes, etc. El cliente es el visitante del sitio Web (específicamente, el explorador Web del visitante). La siguiente figura muestra este concepto:



Procesamiento Web

Aunque ésta esta forma de comunicarse y distribuir información es muy sencilla y estática, no puede proporcionar ninguna información dinámica o de procesamiento. El servidor tan sólo espera a que alguien le pida información para devolver lo que está almacenado en su disco duro sin saber realmente lo que es. Por lo general, una petición a una página Web estática sigue estos pasos:

1. El cliente (el explorador Web) localiza un servidor Web mediante su URL (como www.upct.es).
2. El cliente solicita una página (como index.html).
3. El servidor envía el documento solicitado.
4. El cliente recibe el documento y lo muestra.

Una vez que el cliente ha recibido la información, el proceso termina. El servidor no tiene idea de lo que ocurre en el cliente. ¿Y cómo podría, si son dos equipos distintos? Sólo se comunican entre sí durante el proceso de petición-respuesta. Una vez que la página se haya enviado, al servidor no le importará lo que ocurra.

El procesamiento se realiza en el servidor. Esto se hace por diversos medios, entre ellos la Interfaz Común de Puerta de Enlace (CGI) y las Páginas Active Server (ASP) de Microsoft, que ahora se conoce como ASP clásico. En este caso, el servidor analiza la información antes de enviarla, y puede tomar pedidos del cliente. Puede devolver datos dinámicos, como los de una base de datos, cálculos y cualquier cosa que el cliente solicite. El flujo de trabajo modificado funciona como sigue:

1. El cliente (el explorador Web) localiza un servidor Web mediante su URL.
2. El cliente solicita una página.
3. El servidor examina el archivo solicitado y procesa cualquier código que contenga.
4. El servidor traduce el resultado del procesamiento a HTML (de ser necesario) y envía el documento solicitado al cliente.
5. El cliente recibe el documento y lo muestra.

Aún en este caso, el proceso se termina cuando el cliente recibe la página. El servidor no tiene idea de lo que el cliente hace, a menos que éste realice otra petición.

3.5.2. La diferencia con ASP.NET

Hay otro modelo para el que el cliente y el servidor se comuniquen, conocido como modelo controlado por eventos. El servidor espera a que algo ocurra en el cliente. Cuando algo sucede, el servidor entra en acción y ejecuta alguna tarea.

Imagine que va a la biblioteca. Si siguiera el modelo petición-respuesta, le pediría al bibliotecario alguna información, y él le daría la respuesta o las indicaciones adecuadas. Con el modelo controlado por eventos, el bibliotecario ya sabría lo que usted hace. Si está elaborando un informe acerca de Cristóbal Colón y necesita información, el bibliotecario le llevaría automáticamente los elementos adecuados. Si tuviera sed, el bibliotecario le llevaría un vaso con agua. Si tropieza y cae, el bibliotecario le llevaría una venda.

Naturalmente, el servidor Web no sabe lo que el usuario piensa, pero si puede responder a sus actos. Si el usuario captura algo en la página Web, el servidor responderá a ello. De igual forma, si hace clic en una imagen, el servidor responderá. Este modelo es mucho más sencillo para generar aplicaciones que utilizar el de petición-respuesta. ASP.NET funciona así, detecta las acciones y responde a ellas.

¿Cómo puede saber ASP.NET lo que sucede en su equipo? ¿Cómo puede reaccionar el servidor a lo que pasa en el cliente? ASP.NET se basa en el ingenioso procesamiento en el cliente para simular un modelo controlado por eventos.

3.5.3. Procesamiento en el cliente.

Esto ocurre cuando se escribe cierto código de programación en una página HTML, que el cliente pueda comprender. Este código es simplemente HTML que el explorador Web ejecuta. Por ejemplo, el siguiente código:

```
<html>
<head>
    <script language="JavaScript">
    <!--
        alert("¡ Hola, mundo!");
    // -- >
    </script>
<head>
<body>
    ¡Bienvenido a mi página!
</body>
</html>
```

Código HTML

Cuando el explorador Web recibe esta página, la trata como HTML. Las etiquetas <script> denotan una porción de la página que contiene comandos para el cliente, conocida como secuencia de comandos.

Si nuestro explorador Web reconoce las secuencias de comandos en el cliente, comprenderá que la línea 5 le indica que le muestre al usuario un cuadro de mensaje que diga “¡Hola, mundo!”.

Así pues, ahora tenemos dos lugares para ejecutar código: en el servidor, donde todo se devuelve al cliente como HTML, y en el cliente. Estos dos lugares para el código son distintos y no tienen relación entre si. Veamos cuales son las diferencias entre el código en el cliente y en el servidor:

- **En el cliente**:

  Este código no se procesa en el servidor. Es sólo responsabilidad del cliente.

  El código se escribe en secuencias de comandos, es decir, comandos de texto que indican al cliente que haga algo.

  Generalmente se utiliza para realizar efectos dinámicos en el cliente, como variaciones de una imagen o despliegue de cuadros de mensajes.

- **En el servidor**:

  Este código se ejecuta sólo en el servidor. Cualquier resultado o información que produzca deberá ser convertido a HTML o XML antes de enviarlo al cliente.

  El código también puede describirse en secuencias de comandos, pero ASP.NET utiliza lenguajes compilados.

  Se utiliza para procesar resultados y devolver datos.

3.5.4. Como enlaza ASP.NET los elementos.

Así pues, ¿cómo sabe ASP.NET lo que ocurre en el cliente? Las secuencias de comandos en el cliente no tienen relación con el código en el servidor. No obstante, ASP.NET soluciona inteligentemente este problema. La única forma en que el cliente se puede comunicar con el servidor es mediante una petición.

A través de las secuencias de comandos en el cliente, ASP.NET proporciona información de lo que éste hace durante las peticiones. Volviendo al ejemplo de la biblioteca, por esto el bibliotecario parece tener poderes mágicos que pueden detectar lo que usted necesita. Tiene una red de espías que lo observan, aún cuando el usuario no lo sepa. Cuando el usuario hace algo, un espía se dirige rápidamente con el bibliotecario para decirlo lo que ha ocurrido. En este momento, el bibliotecario puede tomar la decisión adecuada, como llevarle un vaso de agua o una venda.

Los espías, antes mencionados, son las secuencias de comandos en el cliente. Si algo ocurre en él, se ejecutará una secuencia de comandos que enviará información al servidor, tal como consignar un formulario que envía información al servidor. El explorador Web es, sin saberlo, un cómplice. Pensemos que sólo hace la tarea de mostrar HTML. Así pues, las secuencias de comandos en el cliente no tienen relación con aquellas en el servidor, pero pueden comunicarse mediante el envío de mensajes.

De tal manera, ASP.NET vincula al servidor y al cliente, lo que permite a los desarrolladores hacer cosas en las páginas Web que antes no eran posibles. No tendrá que centrarse en manejar peticiones y respuestas, sino que podrá concentrarse en generar la lógica. Podrá reaccionar a los eventos del usuario inmediatamente en lugar de

esperar hasta que se consignen los formularios. Y podrá conocer la estructura de la interfaz de usuario y cómo manejarla previamente. ASP.NET en realidad facilita la vida de los desarrolladores.

3.5.5. Comparación entre ASP y ASP.NET

ASP.NET es una completa renovación del ASP. Por ello, ofrece una metodología muy distinta para generar aplicaciones Web.

Las siguientes secciones muestran algunas de las diferencias generales entre ASP y ASP.NET.

3.5.5.1. Modificaciones fundamentales de ASP.

El ASP clásico está generado sobre el sistema operativo de Windows e IIS. Siempre fue una entidad por separado y, por ello, su funcionalidad estaba limitada.

Por otro lado, ASP.NET es una parte integral del sistema operativo bajo el .NET Framework. Comparte muchos de los objetos que las aplicaciones tradicionales utilizarían, y todos los objetos .NET están disponibles para el uso de ASP.NET. En lugar de estar limitado a los seis objetos inherentes de ASP.NET tiene a su disposición toda una gama de componentes.

ASP también hizo muy evidente que el cliente y el servidor eran dos entidades por separado. Una vez que ASP finalizaba su trabajo en el servidor, pasaba el HTML al cliente y se olvidaba de él. ASP.NET vincula al cliente y al servidor mediante una ingeniosa técnica de codificación que no es evidente para el desarrollador. Ahora, el desarrollo Web es más parecido al desarrollo de las aplicaciones tradicionales que al modelo un tanto desvinculado petición-respuesta característico de ASP.

Es más, el código de ASP.NET es compilado, mientras que el ASP clásico utiliza lenguajes interpretados de secuencias de comandos. El uso de código compilado se traduce en un rendimiento más ágil sobre las aplicaciones de ASP. Esto hace a las páginas ASP.NET más parecidas a las aplicaciones tradicionales y así serán tratadas a lo largo de este libro.

Por éstas y otras mejoras, ASP.NET pone un gran potencial en las manos del desarrollador.

3.5.5.2. Mejoras programáticas sobre ASP.

Junto con esos cambios fundamentales, ASP.NET ofrece muchas mejoras programáticas, entre las que se incluyen el almacenamiento en caché, compilación de código y mayor simplicidad y seguridad.

Quizá una de las mejoras más notables en ASP.NET es su facilidad en la distribución. Los metadatos almacenan toda la información necesaria para las aplicaciones, por lo que ya no hay que registrar aplicaciones Web u objetos COM. Cuando distribuimos aplicaciones clásicas de ASP, necesitamos copiar las DLL correspondientes y utilizar REGSVR32.EXE para registrar los componentes. Con

ASP.NET, todo lo que se necesita es copiar los archivos DLL. .NET Framework se ocupará de lo demás.

El estado de la sesión era un concepto muy importante en el ASP clásico. Es la facultad de determinar automáticamente si una secuencia de peticiones proviene del mismo cliente, principalmente mediante el uso de las cookies. Esta administración de sesiones facilita el control de los usuarios y sus acciones. Con ello aparecieron los carritos de la compra y el desplazamiento de datos. No obstante, conforme los sitios Web empezaban a moverse hacia las granjas de servidores (conjunto de servidores que manejan el mismo sitio Web), los desarrolladores empezaron a darse cuenta de las limitaciones del manejo de sesiones basado en ASP. Entre estas limitaciones está el que las sesiones no pueden transferirse entre servidores.

El manejo de sesiones se facilitó mucho y se hizo más potente con ASP.NET. Esta tecnología resuelve este detalle al integrar el uso de sesiones que puede ampliarse a granjas Web. También, ofrece la confiabilidad de que podrá, incluso, sobrevivir a colapsos del servidor y trabajar con exploradores Web que no pueden utilizar las cookies.

En el ASP clásico, casi todo el código se ejecutaba en bloques proveedores de código. En ASP.NET, este tipo de código no se compila y no se recomienda para un uso frecuente. En vez de ello, utilizaremos bloques de declaración de código, que son compilados y dan un mejor rendimiento. Estos bloques también evitan el tener código ejecutable y HTML mezclado por la página, lo que dificulta su lectura en una página ASP. Los bloques de declaración de código pueden colocarse al principio de la página, separados del resto del HTML, y aún así podremos controlar los aspectos visuales de la página.

3.5.5.3. Diferencias en metodologías programáticas.

Debido a que ASP.NET vincula al servidor y al cliente de formas que eran imposibles en el ASP clásico, el desarrollo de aplicaciones ASP.NET necesita una metodología más intuitiva. El desarrollador ya no tendrá que preocuparse de recordar la información del estado de usuario o de solicitar variables de captura, todo lo maneja ASP.NET. En lugar de ello, el desarrollador puede centrarse en responder a las acciones del usuario sin preocuparse por los detalles del modelo petición-respuesta.

Además, ASP.NET ahora es totalmente orientado a objetos. El ASP clásico hizo un esfuerzo por presentar el concepto de programación orientada a objetos (OOP), pero no pudo porque era un paradigma de programación fundamentalmente distinto. Los desarrolladores que estén habituados a los aspectos de la OOP trabajarán sin problemas con ASP.NET.

Sin embargo, pese a las muchas modificaciones en ASP.NET, se denota que ASP.NET facilita mucho las cosas la mayor parte del tiempo. ASP.NET es un adelanto lógico en la programación para Internet.

3.5.6. Ciclo de vida de una aplicación

En este apartado vamos a estudiar la interconexión entre los controles y el código servidor de la aplicación, así como la dinámica de los eventos.

3.5.6.1. Organización del código de una página.

La unidad ejecutable de una aplicación ASP.NET, desde el punto de vista del cliente, es una página aspx que se accede mediante su URL.

La página está compuesta de una plantilla (*template*) HTML y de código escrito en un lenguaje .NET. En la plantilla se puede declarar controles específicos ASP.NET. A cada plantilla se la asocia una clase derivada de **System.Web.UI.Page**. La llamada de la página provoca la instanciación de la clase asociada. Cada elemento de la plantilla está representada por un objeto instanciado en el servidor. La organización jerárquica de los controles de la plantilla se conserva dentro de la propiedad **Controls** del objeto página. Ésta es el contenedor de todos los controles del primer nivel que, a su vez, pueden contener otros controles, etc. Cada control realiza su propia restitución HTML en la respuesta al navegador cliente. Incluso puede adaptar dicha respuesta en función del tipo de navegador.

El código servidor relativo a la página puede definirse en la plantilla, aislado en un marcador <script runat=″server″>, o en un archivo fuente separado (*Codebehind*).

En comparación con tecnologías como ASP o PHP, este modelo tiene las siguientes ventajas:

- Separación clara de la lógica (código) y de la presentación (HTML).
- Adaptación simplificada a los distintos tipos de navegador.

En la primera ejecución de una página (que sigue a una petición HTTP sobre su URL), ASP.NET compila la clase asociada a la página y crea un ensamblado en un directorio temporal, por defecto: **C:\WINNT\Microsoft.NET\Framework\v1.0.3705\Temporary ASP.NET Files.** El ensamblado se ejecuta para responder a la petición.

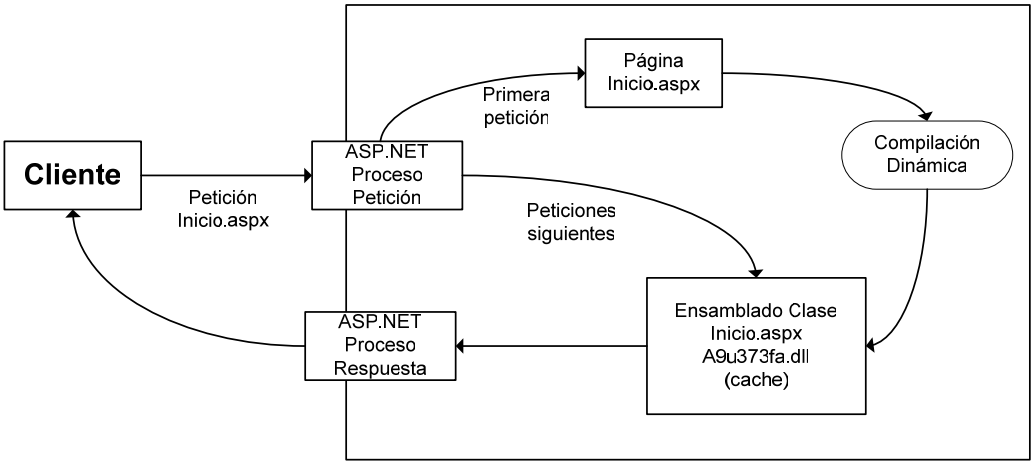
3.5.6.1.1. Página de archivo único.

En el modo de archivo único el programador no declara la clase asociada a la página; en su lugar se contenta con implementar sus elementos internos (campos, métodos, propiedades, gestores de eventos) en el marcador <script runat=″server″>. Los gestores de eventos son referenciados en los atributos adecuados de los marcadores de la plantilla.

Este modo tiene la ventaja de la simplicidad (para programar basta con un simple editor de texto), y presenta las siguientes restricciones:

- El código se distribuye en el servidor (pero no está accesible para los usuarios del sitio Web).

- En la etapa de concepción del código no se realiza ninguna compilación del mismo. Por tanto, los errores de escritura de código sólo se detectan en la primera ejecución.



Procesamiento de página de archivo único

El modo de archivo único se usa al desarrollar con las herramientas **.NET Framework SDK**.

3.5.6.1.2. Página con archivo de código.

En este modo, utilizado por Visual Studio, se asocia al archivo aspx un archivo fuente C# o VB.

```
<%@ Page Language="C#" AutoEventWireup="false" CodeFile="Test.aspx.cs"
Inherits="ApliTest.Test" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
      <head runat="server">
      <title>Test</title>
            <meta name="GENERATOR" content="Microsoft
            Visual Studio 7.0" />
            <meta name="CODE_LENGUAGE" content="C#"/>
            <meta name="vs_defaultClientScript"
            content="JavaScript" />
            <meta name="vs_targetSchema"
            content="http://schemas.microsoft.com/intellisense/ie5"/>
      </head>
      <body>
            <form id="TestPrueba" method="post"runat="server">
                  <asp:Label id="Label1
                  runat="server">Label</asp:Label>
            </form>
      </body>
</html>
```

La directiva **Page** contiene atributos específicos al modo **Codebehind** (fichero fuente asociado a la página aspx):

- El atributo **CodeFile** indica el nombre del archivo fuente asociado.
- El atributo **AutoEventWireUp** en false indica que la conexión de los eventos se realiza en el archivo fuente asociado (*Codebehind*), y no mediante un atributo en la plantilla HTML.
- El atributo **Inherits** indica que la clase generada en la compilación debe derivar de la clase indicada, es decir: la definida en el archivo fuente asociado (*Codebehind*).

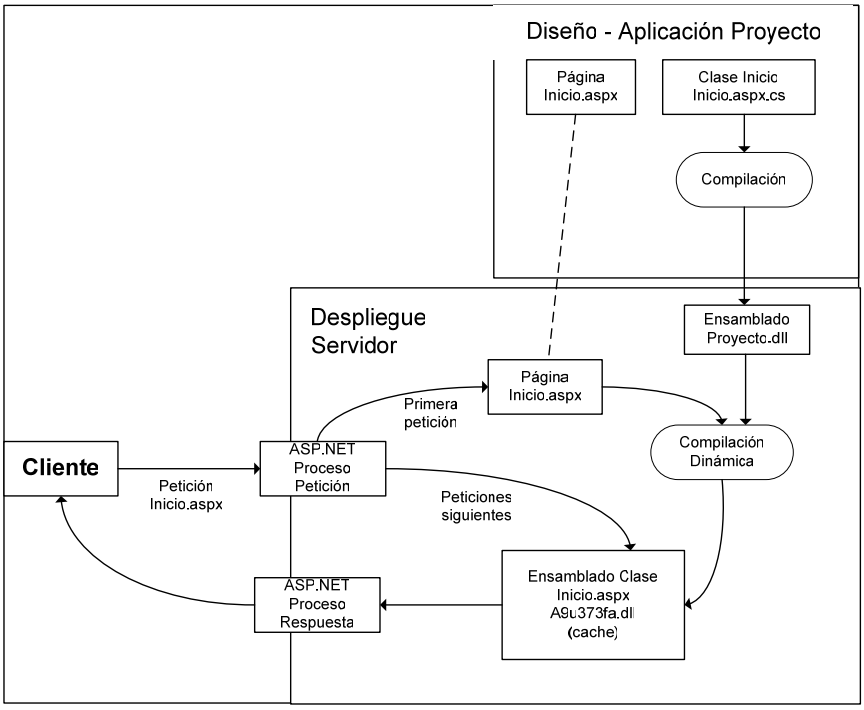
Código de una página escrita en Visual Studio (Codebehind):

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

namespace ApliTest
{
    /// <summary>
    /// Descripción breve de [!ouput SAFE_CLASS_NAME].
    /// </summary>

    public partial class Test : System.Web.UI.Page
    {

        private void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = "ASP.NET";

        }
        #region Web Form Desingner generate code
        protected override void OnInit(EventArgs e)
        {
            //
            // CODEGEN: llamada requerida por el Diseñador de Web Forms ASP.NET.
            //
            InitializeComponent();
            base.OnInit(e);
        }
        ///<sumary>
        ///Método necesario para admitir el Diseñador, no se
        ///puede modificar el contenido del método con el editor
        ///de código
        ///</sumary>
        private void InitializeComponent()
        {
            this.Load += new System.EventHandler(this.Page_Load);
        }
        #endregion

    }
}
```

El ciclo de compilación es más complejo que en el caso del archivo único:

- Una primera compilación tiene lugar en la fase de concepción; genera un ensamblado que debe distribuirse con los archivos aspx (en su subdirectorio **bin**).
- Debido a la posible existencia previa de código de enlace de datos en el archivo aspx, se deriva automáticamente una nueva clase a partir del ensamblado que contiene la primera clase, y que integra el código de enlace de datos.

- El código fuente intermedio puede ser consultado en el directorio temporal de compilación.



Procesamiento de página con archivo

3.5.6.2. Eventos de los controles de Servidor.

Los controles HTML clásicos insertados en un archivo son enviados directamente al navegador y tratados allí donde corresponda. Sus eventos generan script en el lado del cliente o envían peticiones HTTP que deberán tratarse manualmente en el servidor.

Los controles ASP.NET generan en el servidor eventos de forma automática, cuyos controladores son los métodos de la clase que representan la página.

Se envía al servidor una petición HTTP y se trata automáticamente para generar el método adecuado. Tras la ejecución del método, se retorna una respuesta HTTP al navegador, si bien el programador no debe preocuparse por restaurar la página, porque su estado fue guardado automáticamente.

Esta gestión de eventos requiere, pues, de un servidor de ida y vuelta, llamado **PostBack**, y no puede aplicarse a eventos con una frecuencia elevada como, por ejemplo, los eventos de ratón, que saturarían al servidor.

Otros eventos con una frecuencia media, como los de introducción de caracteres en una zona de texto, podrían impactar negativamente en el rendimiento si son tratados sistemáticamente en el servidor. El nombre de estos eventos acaba en **Changed**, porque generalmente reflejan un cambio en el valor del control. En el texto se referencia a estos controles con el término **xxxChanged**. Por defecto, estos eventos se colocan en caché en el cliente, para tratarlos luego de forma asíncrona: cuando se produce una petición **POST** son enviados al servidor. Podemos forzar estos controles para que funcionen en modo **PostBack**, asignándoles el valor **true** a su propiedad **AutoPostBack**.

Esta es la lista de los controles ASP.NET afectados:

- **CheckBox**
- **CheckBosList**
- **DropDownList**
- **ListBox**
- **RadioButton**
- **RadioButtonList**
- **TextBox**

3.5.6.3. Objeto Page.

El objeto **Page** es el equivalente Web de la ventana (o ficha) de los proyectos gráficos. Sirve como soporte al desarrollo visual basado en controles, a través del archivo aspx asociado e integrado en el código .NET enlazado a los controles.

3.5.6.3.1. Ciclo de vida, ViewState

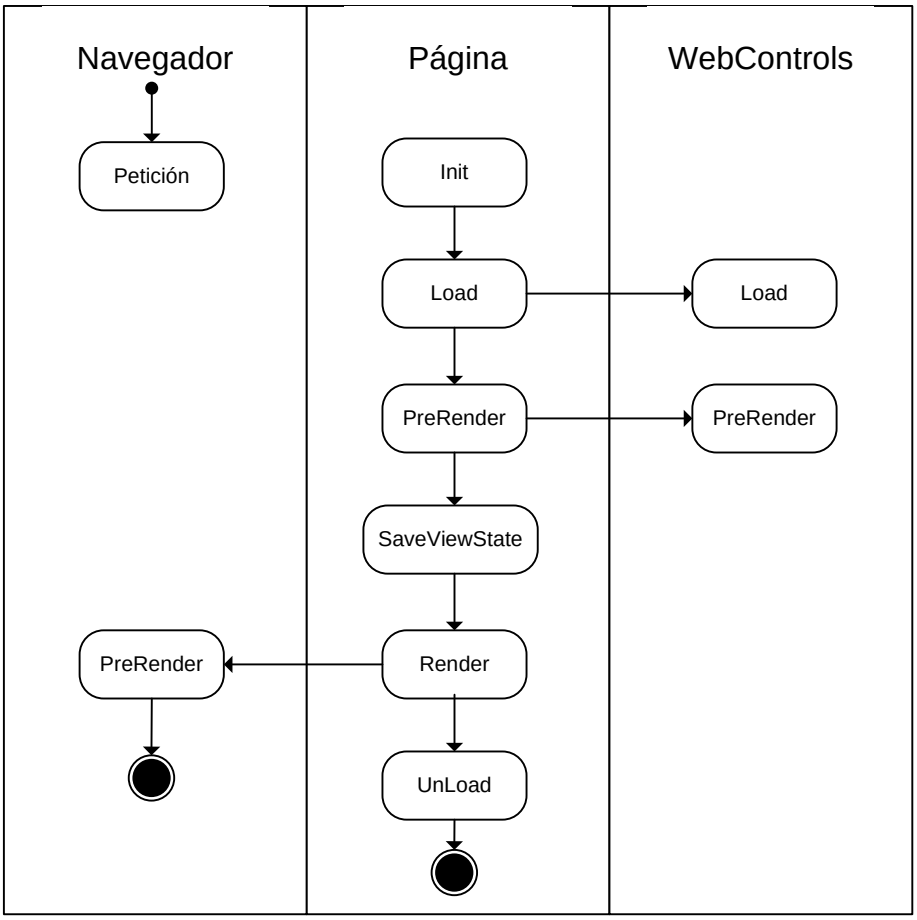
En cada petición (incluidas las de un **PostBack**), el objeto **Page** es instanciado, ejecutado y destruido, tras haber emitido un cierto número de eventos y haber creado el flujo HTML y la respuesta HTTP al navegador.

El estado de los valores de las propiedades de los controles se reinyecta en un campo caché, el **ViewState**, que permite reconstituir en el servidor la visualización de forma automática en el siguiente **PostBack**.

Esta es la cronología de los eventos de la página:

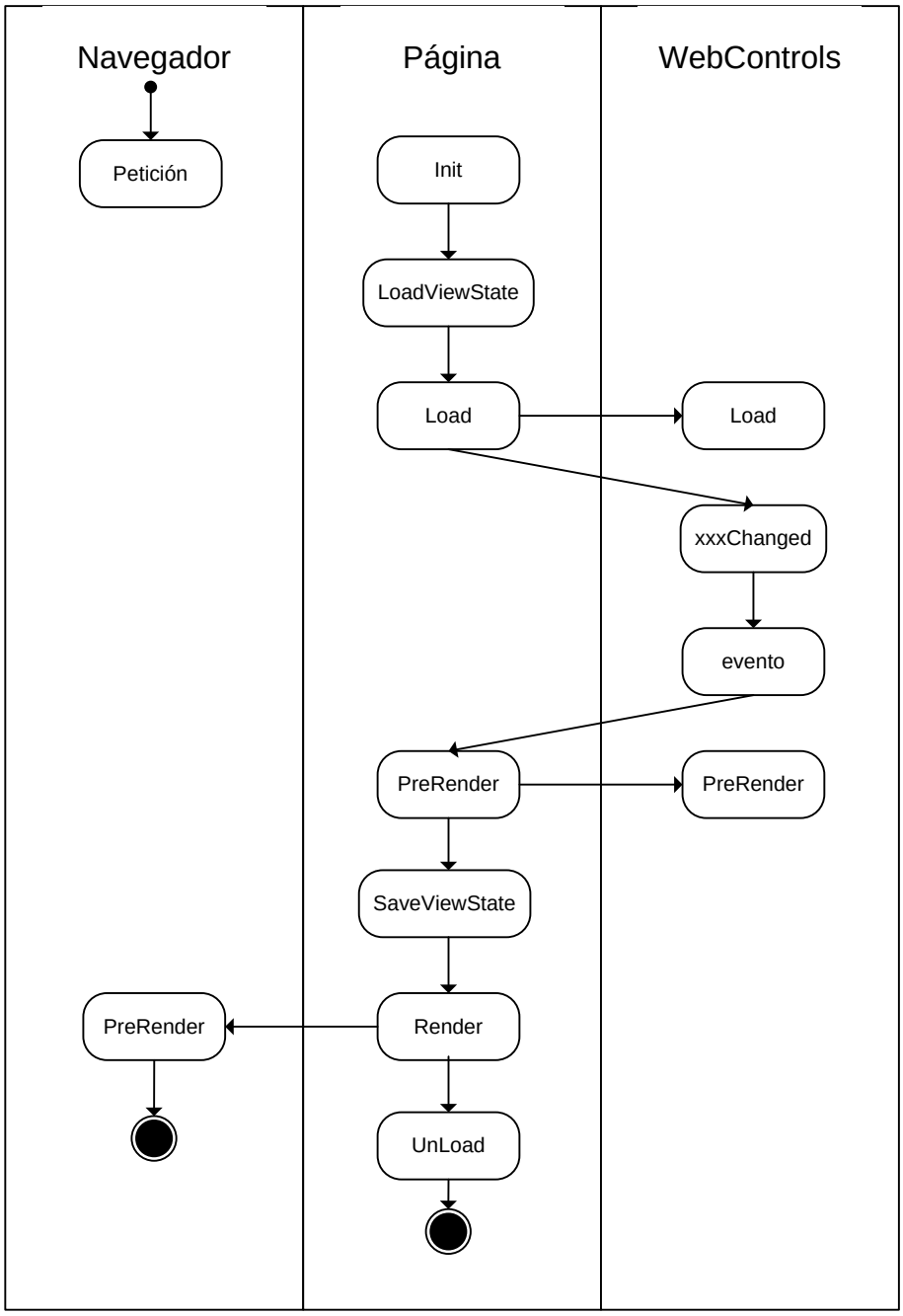
- **Init**: evento emitido tras la instanciación de los controles, pero antes de la lectura de **ViewState** (por tanto, una posible inicialización de propiedad a este nivel se perderá).
- **Load**: evento emitido antes de que los controles sean inicializados con los datos (en caso de **PostBack**, el **ViewState** ha sido leído por el método protegido **LoadViewState**).
- en caso de **PostBack**, emisión de los eventos **xxxChanged** de los controles necesarios y del propio evento origen del **PostBack**.
- **PreRender**: evento emitido antes de la restitución HTML y la grabación del **ViewState** (aquí puede escribirse el código de las modificaciones visuales).
- grabación del **ViewState** (método protegido **SaveViewState**).
- restitución HTML (método protegido **Render**).
- **Unload**: evento emitido antes de la destrucción de la página, para liberar todos los recursos. Se emite un evento **Unload** para cada WebControl.

El esquema siguiente muestra el ciclo de una página desde su primera llamada:



Ciclo de vida de una página 3

El esquema siguiente muestra el ciclo de una página desde un **PostBack**:



Ciclo de vida de una página 4

3.5.6.3.2. PostBack, modo Trace.

Para saber si una página está cargada en el marco de un **PostBack** podemos comprobar la propiedad **IsPostBack**. Normalmente esta prueba se escribe en el evento **Load** (método **Page_Load**), para que la inicialización sólo se haga en la primera carga.

La directiva **Page** tiene un atributo **trace** que aporta una ayuda a la depuración de errores y que permite, por ejemplo, visualizar el encadenamiento de los eventos.

3.5.6.4. Archivos de configuración.

3.5.6.4.1. Jerarquía de configuración.

La configuración de ASP.NET se basa en un sistema de archivos XML con múltiples niveles.

En el archivo **machine.config**, situado en uno de los directorios del Framework .NET podemos definir parámetros que afectan a toda la máquina.

Los parámetros de una aplicación ASP.NET se definen en el archivo **Web.config** de su directorio virtual. Este archivo aparece en el explorador de soluciones de Visual Studio .NET.

Por último, podemos crear un archivo en cada subdirectorio de la aplicación para definir ciertos parámetros específicos.

Los parámetros pueden ser definidos en cada nivel.

La modificación de un parámetro se aplica inmediatamente, y no requiere reiniciar la aplicación.

3.5.6.4.2. Parámetros personalizados.

Es posible definir elementos XML personalizados en los archivos de configuración, para almacenar así parámetros específicos.

El formato de los elementos de los archivos de configuración es utilizado por los gestores, clases que implementan la interfaz **IConfiguration-SectionHandler**.

Los elementos y su gestor deben declararse en el elemento **/configuration/configsections**. El archivo **machine.config** contiene las declaraciones de los elementos estándar de configuración. El elemento **appSettings** se define por defecto para almacenar los parámetros personalizados.

La recuperación de los datos del elemento **appSettings** se hace mediante la colección **AppSettings**, definida estáticamente en la clase **System.ConfigurationSettings**.

3.5.6.5. Encapsulación de HTTP.

ASP.NET encapsula la petición HTTP actual en varios objetos, accesible a través de distintas propiedades en clases distintas (**Page**, **HttpContext**).

3.5.6.5.1. Objeto Context.

La propiedad **Context** del objeto **Page** es un objeto de tipo **HttpContext**, que encapsula la información relativa a la petición que se está procesando es ese momento. Supervisa objetos de la clase **HttpRequest**, **HttpResponse** y **HttpServerUtility**, así como objetos de gestión del estado, directamente accesibles desde el objeto **Page**.

La colección **HttpContext.Items** se puede utilizar para vehicular informaciones globales durante el procesamiento de una petición http como, por ejemplo, para permitir el acceso a datos de una página desde los controles de usuario.

3.5.6.5.2. Objeto Server.

La propiedad **Server** del objeto **Page** es un objeto de tipo **HttpServerUtility**, que ofrece distintos métodos útiles para la gestión de las peticiones:

- **UrlEncode( )**: corrige los caracteres no válidos de un cadena representada por una URL.
- **UrlDecode( )**: es la operación inversa a la anterior.
- **HtmlEncode( )/HtmlDecode( )**: mismo principio que UrlEncode ( ), pero en este caso sobre una cadena HTML.

3.5.6.5.3. Objeto Request.

El objeto **Page** dispone de una propiedad **Request** que representa a un objeto de tipo **HttpRequest**. Es un atajo de **Context.Request**. El objeto **Request** ha diseccionado la petición HTTP en proceso, y expone todos sus elementos mediante propiedades de tipo collection, siendo las principales:

- **Form**: variables de un formulario HTML (petición HTTP POST).
- **QueryString**: argumentos de una petición HTTP GET.
- **Cookies**: valores de las cookies.
- **ServerVariables**: variables del servidor Web.
- **Params**: supercolección que reagrupa a todos los valores que pueden ser accedidos por **Form**, **QueryString**, **Cookies** y **ServerVariables**.
- **Headers**: encabezados HTTP.
- **Browser**: información sobre el navegador.

Ejemplo de lectura en QuerySring:

```
string video = Request.QueryString.Get("video");
```

3.5.6.5.4. Objeto Response.

La propiedad **Response** del objeto **Page** es un objeto de tipo **HttpResponse**, que permite construir la respuesta HTTP que debe enviarse al cliente como contestación a una petición http. Es un atajo de **Context.Response**.

Ejemplos:

- El método **Write( )** permite construir directamente el flujo HTML. El método **WriteFile( )** añade el contenido de un archivo.

- La propiedad **Cookies** permite escribir los valores de los cookies que deben enviarse al cliente.
- El método **Redirect( )** permite cargar automáticamente una nueva página en el navegador.

Ejemplo de redirección:

```
video = @"C:\Documents and Settings\usuario
      \Escritorio\Videos wmv\MOV00094.MPG";
Response.Redirect("vervideo.aspx?video="+video);
```

3.5.6.6. Gestión del Estado.

Se han desarrollado distintas técnicas para ofrecer la persistencia de la información entre las páginas de una aplicación ASP.NET.

Estas técnicas almacenan información en el cliente o en el servidor. La selección de una determinada técnica depende del nivel de seguridad deseado, de la cantidad de información y de restricciones de rendimiento. Las técnicas del lado cliente dan preferencia a las prestaciones y a los pequeños volúmenes. Las técnicas del lado servidor dan más seguridad y soportan volúmenes de información más importantes.

3.5.6.6.1. Gestión del estado del lado cliente.

Un primer método para salvaguardar el estado del cliente consiste en escribir la información en el código HTML generado por una página. De ese modo la información puede transmitirse a la página siguiente, quien podrá utilizarla y, en su caso, volverla a reenviar. Así funcionan, por ejemplo, las cadenas de petición, los campos ocultos y el **ViewState**.

La técnica de las cookies, que utiliza el navegador para almacenar la información, supone una alternativa al método anterior. La cookie es una colección de pares clave/valor. El servidor define la cookie en una respuesta HTTP. El navegador la intercepta y almacena en el disco la información que contiene, asociándola a una URL. En cada nueva petición a dicha URL, los datos contenidos en la cookie se transmiten al servidor. La validez de los datos (permanentes o temporales) es definida por el servidor.

- **Cadenas de petición**: Una cadena de petición es una lista de pares argumento=valor que siguen a una URL: http://localhost:2488/Proyecto2/vervideo.aspx?video=C:\Documents%20and%20Settings\usuario\Escritorio\Videos%20wmv\MOV00041.MPG. Con el fin de propagar la información entre las páginas, los argumentos son escritos por la página que genera dinámicamente el enlace utilizando la URL de la página destino. Los datos pueden ser leídos con el objeto **Page.Request**.

- **Campos ocultos**: La información se escribe en la propiedad **value** de un campo oculto de formulario HTML <input type="hidden"…>. Puede utilizarse el control ASP.NET **HtmlInputHidden**. Los datos complejos deben ser transformados en cadenas mediante el programa. Los campos

ocultos no son visibles para el usuario, excepto si se consulta el código fuente de la página. Requieren de una petición HTTP de tipo POST.

- **ViewState**: ASP.NET gestiona automáticamente la persistencia del estado de visualización en caso de PostBack mediante un diccionario llamado ViewState, conservado con la ayuda de un campo oculto. **ViewState** contiene una vista binario (indexada, cifrada y comprimida) de los valores de las propiedades de la página y de todos sus controles, gestionada automáticamente. Sin embargo es posible añadir información mediante código, facilitando la gestión de datos complejos. **ViewState** es, pues, una alternativa mejorada a la utilización clásica de los campos ocultos, con un mejor nivel de seguridad, aunque reservado al **PostBack** (página que se llama a si misma).

- **Cookies**: ASP.NET encapsula la gestión de las cookies en los objetos **Page.Response.Cookies** y **Page.Request.Cookies**, que son colecciones de objetos **HttpCookie**. La cookies son enviadas sistemáticamente por el navegador, incluso en las peticiones anidadas. La información de las cookies no es visible directamente por el usuario en el navegador, pero puede acceder fácilmente a los archivos de almacenamiento. Además, el usuario puede borrar las cookies y las opciones del navegador pueden prohibir su uso

- **Selección de una técnica**: La gestión del estado del lado cliente tiene las ventajas de no movilizar recursos en el servidor y de ser potente siempre que los volúmenes de información sean pequeños. Su nivel de seguridad es bajo, porque, aunque estén cifrados, los datos son fácilmente modificables. La tabla siguiente resume las características y restricciones de las distintas técnicas estudiadas:

Criterio		**Cadena de petición**	**Campo oculto**	**ViewState**	**Cookie**
Escenario	Multipágina	Si	Si	No	Si
	PostBack	No	Si	Si	Si
Petición HTTP	POST	No	Si	Si	Si
	GET	Si	No	No	Si
Volumen soportado		Muy pequeño	Pequeño	Pequeño	Pequeño
Dependencia opciones del navegador		No	No	No	Si
Seguridad		Débil	Débil	Media	Débil
Productividad para el programador		Buena	Media	Buena	Buena

3.5.6.6.2. Gestión del estado del lado servidor.

ASP.NET ofrece espacios de memoria específicos para almacenar información global, accesible por todas las páginas mientras dure la sesión del usuario o la propia aplicación. Estos espacios de memoria son gestionados por objetos que exponen un diccionario, que permite insertar parejas clave/valor:

- Objeto **Session** para la información relativa a un usuario (una instancia por usuario).
- Objeto **Application** para la información compartida por todos los usuarios (una sola instancia por directorio virtual).
- Objeto **Cache**: es una alternativa al objeto **Application**, con una gestión mejorada del ciclo de vida de las informaciones que contiene.

Un archivo concreto, **global.asax**, contiene el código de respuesta a los eventos globales del objeto **Session** y del objeto **Application**.

Visual Studio crea automáticamente un archivo **global.asax** por proyecto.

3.5.6.7. Resumen

En manera de resumen del ciclo de vida de una aplicación ASP.NET se muestran las siguientes tablas:

Fases generales del ciclo de vida de una página:

Fase	Descripción
Solicitud de página	Antes de que comience el ciclo de vida de la página ha de ser efectuada la solicitud de la misma. Es en este momento cuando ASP.NET determina si ésta se debe analizar y compilar o si se puede enviar una versión en caché como respuesta sin ejecutar la páginas.
Inicio	En esta fase se establecen las propiedades de la página además de determinar si la solicitud es una devolución de datos o una nueva solicitud.
Inicialización de la página	Durante esta fase los controles incluidos en la página están disponibles. También se aplican los temas correspondientes a la página.
Carga	Durante la carga, si la solicitud actual es una devolución de datos, las propiedades del control se cargan con información recuperada del estado de vista y del estado de control.
Validación	Durante la validación, se llama al método *Validate* de todos los controles de validación, que establece la propiedad *IsValid* de cada uno de los controles de validación y de la página.
Control de eventos de devolución de datos	Si la solicitud es una devolución de datos, se llama a los controladores de eventos.
Representación	Durante la representación, el estado vista se guarda en la página y, a continuación, ésta llama a cada uno de los controles para que aporte su salida representada al valor *OutputStream* de la propiedad *Response* de la página.
Descarga	Se llama descarga cuando la página se ha representado completamente, se ha enviado al cliente y está lista para ser descartada.

Eventos del ciclo de vida:

Evento de página	Uso Típico
Page_PreInit	• Utilizamos la propiedad *IsPostBack* para determinar si ésta es la primera vez que se procesa la página. • Crear o volver a crear controles dinámicos. • Establecer una propiedad *Theme* de forma dinámica. • Leer o establecer los valores de las propiedades de perfil.
Page_Init	• Leer o inicializar las propiedades de los controles.
Page_Load	• Leer y actualizar las propiedades de los controles.
Control events	Realizar el procesamiento específico de la aplicación: • Si la página contiene controles de validación, comprobamos la propiedad *IsValid* de la página y de cada uno de los controles de validación antes de realizar cualquier procesamiento. • Controlar un evento específico; por ejemplo, para el control *Button*, el evento *Clic*.
Page_PreRender	• Realizar los cambios finales del contenido de la página.
Page_Unload	Llevar a cabo el trabajo de limpieza final, que podría incluir: • El cierre de los archivos abiertos y de las conexiones a bases de datos. • La finalización del registro o de otras tareas específicas de cada solicitud.

3.5.6.8. Relación con la aplicación.

Una vez explicados los conceptos relacionados al ciclo de vida de una página ASP.NET hay que indicar la relación de éstos con respecto a la aplicación.

Es importante indicar que el tipo de ciclo de vida que ejecutará la aplicación será el de cualquier aplicación creada con páginas con archivos de código (*Codebehind*, apartado 3.5.6.1) que conlleva a diferencia de la ejecución de aplicaciones sin código *Codebehind* una compilación previa de este código. El código utilizado es C#.

A lo largo de esta aplicación también se referencia a los eventos de los componentes utilizados en cualquier aplicación Web como puede ser un **Button,** eventos que también se producirán en la ejecución de la aplicación ya que han sido utilizados a lo largo del desarrollo de la misma.

Se han explicado también otros conceptos referentes al ciclo de vida de las aplicaciones ASP.NET como pueden ser el *Objeto Server*, el *Objeto Context*, el *Obejto Request*, el *Objeto Response*, etc… algunos de ellos también son utilizados, como por ejemplo el caso del Objeto *Reponse* empleado para recoger variables de la URL.

Por último se muestra un resumen referente a las distintas *Fases* y *Eventos* (apartado *3.5.6.7*) que se producen a lo largo de la ejecución de una aplicación y que por lo tanto también son característicos de la aplicación implementada.

3.5.7. Seguridad en ASP.NET.

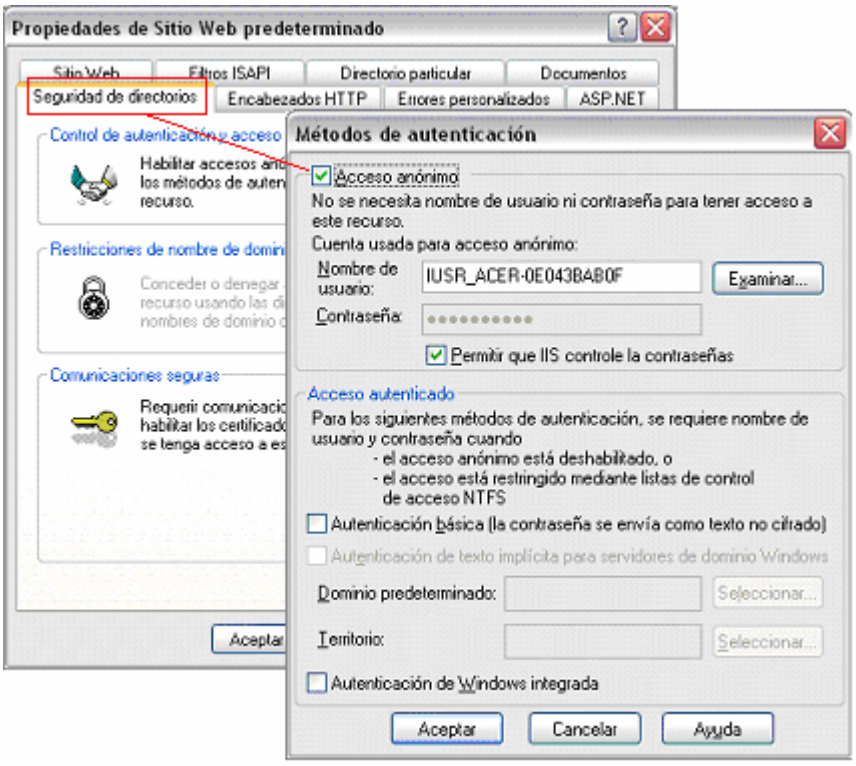
Prácticamente cualquier aplicación que creemos va a necesitar seguridad en el acceso a sus diversas secciones. Lo habitual es tener apartados en los que sólo pueden acceder miembros registrados o una zona de administración restringida a los gestores del sistema. ASP.NET ofrece multitud de novedades en este aspecto que nos va a simplificar mucho el trabajo.

3.5.7.1. Autenticación de usuarios.

Disponemos de diversas barreras que podemos utilizar a la hora de ejercer un control sobre el acceso a los recursos en nuestras aplicaciones Web. La primera de ellas la constituye el propio servidor de aplicaciones que en entornos de producción normalmente es Internet Information Server (IIS).

3.5.7.2. Autenticación IIS/Windows.

IIS permite definir mediante su configuración tanto el modo en el que se autentica a los usuarios como los permisos de acceso genéricos que se otorgan a cada uno de los recursos que sirve. La figura siguiente muestra el aspecto de la ventana que permite ajustar los métodos de autentificación que solicitará a los usuarios IIS.



Configuración de la Seguridad

3.5.7.3. Autenticación Forms de ASP.NET

Desde la primera versión de ASP.NET existe un método de autenticación conocido genéricamente como *Forms* que permite a los programadores crear un sistema propio de autenticación de usuarios del sistema (IIS debe permitir el acceso anónimo a los recursos), en este caso es el programador el que debe establecer sus propio mecanismo de autenticación, por ejemplo lanzando consultas contra una base de datos.

El funcionamiento es el siguiente:

Cuando un usuario solicita una página protegida ASP.NET comprueba si éste se encuentra ya autenticado o no. Lo sabe porque cuando un usuario se ha autenticado previamente, esta información se almacena en una *cookie* cifrada. Si está autenticado se le permite el acceso al recurso siendo responsabilidad del programador definir cualquier comprobación de autorización posterior. En caso de no estar autenticado se le dirige de forma automática a una página que le solicita las credenciales de acceso.

Activar la autenticación *Forms* en una aplicación es muy sencillo. Basta con editar el archivo *Web.config* y añadir la siguiente línea en la configuración de <system.web>:

```
<authentication mode="Forms">
```

Por defecto la página a la que se redirige a los usuarios anónimos para que se autentiquen se llama *login.aspx* pero se puede especificar cualquier cosa usando el atributo *loginUrl*, por ejemplo:

```
<authentication mode="Forms">
      <forms loginUrl="Default.aspx"/>
</authentication>
```

A partir de ahora todos los usuarios anónimos se redirigirán a *Default.aspx* para solicitarle las credenciales de acceso. Una vez en esta página se debe solicitar al usuario unas credenciales y comprobarlas en una base de datos o cualquier otro medio de almacenamiento.

En ASP.NET 1.x había que crear toda la lógica desde cero y apoyarse en la API de autenticación *Forms* para hacerlo. En ASP.NET 2.0 todo esto nos lo ahorraremos.

3.5.7.4. Autorización de usuarios.

Una vez que un usuario está autenticado, el proceso que regula a qué recursos tendrá acceso se denomina autorización.

Para realizar el control de acceso no sólo basta con indicar que método de autenticación se usará (en este caso *Forms*). Además hay que especificar a qué recursos se debe controlar el acceso y de qué manera. En ASP.NET existen diversos modos de definir la autorización de acceso.

3.5.7.5. Autorización de URL.

Consiste en la autorización de acceso a determinados recursos de la aplicación Web (páginas ASPX, carpetas, archivos…) refiriéndose a ellos a través de su ruta relativa en el archivo de configuración de la aplicación.

Se especifica el nivel de acceso requerido mediante una o varias entradas en el nodo raiz de *Web.config* (nodo <configuration>) que tiene un aspecto análogo al siguiente:

```
<location path="facturas.aspx">
  <system.web>
    <authorization>
      <allow users="jose,antonio,maria"/>
      <allow roles="clientes,administradores"/>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
```

Como se puede observar, la sintaxis es muy sencilla y bastante evidente:

- El nodo location especifica la ruta relativa al directorio actual cuyo acceso se desea controlar.
- Los nodos allow y deny se emplean para permitir o denegar el acceso a determinados usuarios o roles de usuario. Se pueden especificar nombres concretos para éstos o utilizar comodines especiales, como en el caso del nodo deny del ejemplo:

    - *?*: Representa a los usuarios anónimos. En el ejemplo anterior se deniega el acceso a todos los usuarios que no se hayan autenticado.

    - *: Indica que el criterio se aplica a todos los usuarios, tanto anónimos como autenticados, independientemente de su perficl de usuario. Por ejemplo, para permitir el acceso a todos los usuarios autenticados (o sea, a todos excepto a los anónimos) se escribiría:

```
<location path="facturas.aspx">
  <system.web>
    <authorization>
      <allow users="*"/>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
```

Pueden incluirse tantos nodos *location* como sea necesario dentro del archivo de configuración.

Lo habitual es incluir junto con el nodo de autenticación un nodo de autorización explícito que deniegue el acceso a todos los recursos a los usuarios anónimos, así:

```
<authentication mode="Forms">
  <authorization>
    <deny users="?"/>
  </authorization>
```

Dado que puede existir un archivo *Web.config* por cada carpeta de la aplicación se pueden establecer distintos criterios de acceso a cada una de ellas, agrupando en ellas los archivos con el mismo nivel de acceso.

3.5.7.6. Los controles Web de seguridad.

ASP.NET nos facilita todavía más el trabajo referente a la seguridad de nuestra aplicación gracias a la inclusión de los nuevos controles Web de seguridad. Los podemos encontrar en el grupo “Login” del cuadro de herramientas de Visual Studio, tal y como se muestra en la siguiente figura:

Controles Web de Seguridad

Estos controles nos dan ya hechas multitud de operaciones comunes de seguridad relacionadas con la interfaz de usuario. Por ejemplo, el control *Login* permite disponer de un completo diálogo de autenticación con sólo arrastrarlo sobre un formulario Web. El *CreateUserWizard* es un asistente con varios pasos que nos permite la creación automática de nuevos usuarios. Todos ellos permiten la personalización, tanto parcial por medio de propiedades, como absoluta usando plantillas. En el caso de los asistentes tenemos libertad de añadir nuevos pasos o modificar los predeterminados a voluntad.

3.5.7.6.1. El control Login

Este control permite definir un completo diálogo de autenticación en cualquier página, que además soporta el uso de temas. Además de los elementos obvios permite configurar enlaces para acceder a la creación de nuevos usuarios, muestra mensajes de fallo de autenticación, redirige automáticamente a otras páginas al autenticar si es necesario, etc…

Todo ello se controla con facilidad desde la ventan “Propiedades del entorno”.

Para validar a los usuarios internamente utiliza el método *ValidateUser* de la clase *Membership*. Por defecto si ya hay un usuario autenticado, este control se oculta automáticamente salvo cuando se ubica en la página principal. Este comportamiento se cambia con la propiedad *AutoHide* en caso de necesitarlo. Así se permite cambiar la sesión de usuario.

3.5.7.6.2. El control LoginStatus.

Se utiliza para mostrar el estado actual de conexión de un usuario y permitir su desconexión. Cuando hay un usuario autenticado el control muestra por defecto un enlace que permite desconectarse. Lo que ello provoca es que se elimine la referencia al usuario actual en la propiedad *User* de la clase *Page* y que se reenvíe a la página de autenticación especificada en *Web.config* para iniciar una nueva sesión.

Cuando no hay usuario alguno autenticado en el sistema el enlace apunta automáticamente a la página de autenticación definida. Se puede personalizar por completo para mostrar cualquier otra cosa en la superficie de control.

3.5.7.6.3. El control LoginName.

Muestra el nombre del usuario actualmente autenticado.

3.5.7.6.4. El control LoginView.

Se trata de un completo control que permite definir el contenido de una zona de la página en función de si el usuario está o no autenticado, y en caso de estarlo incluso en función del rol o roles a los que está asociado.

3.5.7.6.5. Controles Restantes.

PasswordRecovery, *ChangedPassword* y *CreateUserWizard* nos facilitan la recuperación de claves, el cambio de clave y la creación de usuarios respectivamente.

Al igual que los anteriores ofrecen una altísima capacidad de personalización que en el caso del control de creación de usuarios permite incluso añadir nuevos pasos en el asistente (se trata de un control heredado del control de tipo *Wizard* que nos permite crear asistentes con facilidad).

Todos ellos utilizan la API de *Membership* para trabajar y responden a los ajustes impuestos en la configuración de la aplicación. Así pues, por ejemplo, el control de cambio de contraseña o el de creación de usuarios exigirán a las contraseñas la complejidad indicada en la configuración, y el de recuperación de clave le hará una pregunta de seguridad si así está definido.

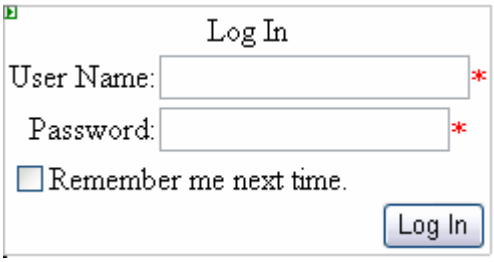
Gracias a estos controles y a la utilidad de administración de seguridad de ASP.NET podemos construir la seguridad completa de una aplicación Web en cuestión de minutos, cuando lo habitual sería que tardásemos horas o días.

3.5.7.7. Seguridad implementada.

En la implementación de la aplicación *Tu Servidor de Video* se ha optado por una seguridad basada en Formulario (*Forms)*, elección que se ha de indicar en el fichero *Web.config* de tal forma como la indicada en el apartado 3.5.7.3. Basta con escribir la siguiente línea:

```
<authentication mode="Forms">
```

Para completar la implantación de esta seguridad se ha utilizado un componente de tipo **Login:**



Componente Login

Es en este componente donde el usuario, previamente registrado en la base de datos, se debe autentificar ante la aplicación para poder tener acceso a la misma.

Una vez que el usuario introduce la URL de la aplicación, ésta debe de saber hacía donde redirigir dicha petición. En este caso deberá aparecer como página de Inicio donde se encuentre este componente, esto se indica también en el *Web.config*:

```
<authentication mode="Forms">
      <forms loginUrl="Default.aspx"/>
</authentication>
```

Por último se ha de ejecutar la consulta a las base de datos que compruebe el nombre y la contraseña del usuario. Esta consulta y su posterior tratamiento se realiza en el fichero de código C# asociado a la página donde se localiza el componente **Login:**

```
//Consulta ejecutada.

"SELECT count(*) FROM usuario WHERE usuario ='" + Login1.UserName + "' and
clave='" + Login1.Password+"'"

//Ejecución de la consulta.

contador = Convert.ToInt32(comando.ExecuteScalar());
   if (contador == 1)
     {
        //Damos acceso al usuario

        FormsAuthentication.RedirectFromLoginPage(Login1.UserName, false);

     }
```

3.6. MySQL

En pleno apogeo de la Era de la Información, donde cada vez es mayor el volumen de datos que se almacena en computadoras, la necesidad de bases de datos fiables y de alta velocidad ha aumentado drásticamente. Durante muchos años, determinadas compañías, coma Oracle, han venido ofreciendo aplicaciones específicas que iban destinadas principalmente a grandes organizaciones, capaces de asumir el coste y la demanda de personal que requiere un software diseñado para trabajar con información crítica. Entre tanto, dentro del seno de la comunidad *open source* -que ha florecido especialmente en la última década- ha visto la luz una nueva generación de aplicaciones de bases de datos pequeñas, fiables y poco costosas. El software de este tipo, como MySQL, ha proporcionado a los usuarios y desarrolladores una opción práctica muy asequible para satisfacer sus demandas en materia de bases de datos.

MySQL, afortunadamente, está ya muy lejos de sus modestos orígenes y ha pasado a convertirse en una aplicación de bases de datos robusta, fiable y fácil de manejar. Y lo más sorprendente es que MySQL se las ha ingeniado para conservar sus raices *open source* y sigue estando disponible para ser usada o modificada sin tener que pagar nada por ello. Las características que ofrece MySQL en la actualidad explican por qué la mayoría de las organizaciones de mayor peso mundial, como Yahoo y la NASA, la utilizan en sus operaciones diarias. Su bajo coste y su alta accesibilidad explican también por qué el lector puede incorporar MySQL en sus proyectos personales o profesionales.

3.6.1. ¿Qué es MySQL?

MySQL es la base de datos *open source* más popular y, posiblemente, mejor del mundo. De hecho, MySQL es un competidor cada vez más directo de los gigantes en esta materia como Oracle o SQL Server de Microsoft.

MySQL fue creado y sigue siendo desarrollado en la actualidad por MySQL AB, una compañía radicada en Suecia. Hasta cierto punto, MySQL creció a partir de una base de datos *open source* existente, mSQL, la cual sigue en fase de desarrollo, aunque su popularidad ha decaído. Irónicamente no resabe lo que significa exactamente MySQL. La última parte, SQL, se refiere a *Structured Query Language* (Lenguaje de consultad estructurado) –el lenguaje usado para interactuar con la mayoría de bases de datos-, aunque el prefijo “my” sigue siendo un misterio, imposible de desvelar incluso en el seno de la compañía MySQL AB.

MySQL es un sistema de administración de bases de datos (*Database Management System*, DBMS) para bases de datos relacionales (por tanto, MySQL es un RDBMS). Técnicamente, MySQL es una aplicación que permite administrar archivos llamados bases de datos. Una base de datos no es más que una colección de datos interrelacionados, ya sean éstos texto, números o archivos binarios, los cuales se almacenan y se mantienen debidamente organizados por medio del DBMS.

Hay muchos tipos de bases de datos, desde un simple archivo hasta sistemas relacionados orientados a objetos. Una base de datos relacional utiliza múltiples tablas para almacenar información, clasificándola en sus componentes más elementales. Antes de principios de los años setenta, cuando se desarrolló este concepto, las bases de datos

se parecían más a hojas de cálculo que contenían una simple tabla donde se metía toda la información. Aunque las bases de datos relacionales implican un mayor esfuerzo de diseño y programación, ofrecen una fiabilidad e integridad de datos muy superior, que compensan con creces el esfuerzo adicional que requiere su manejo.

MySQL es una aplicación *open source*, al igual que PHP y algunas variantes de Unix, lo que significa que se puede utilizar gratuitamente e incluso modificar con total libertad (se puede descargar incluso el propio código fuente). No obstante, hay ocasiones en que es necesario pagar para conseguir una licencia de MySQL, sobre todo si se obtienen beneficios de sus ventas o de su inclusión en algún otro producto.

El software de MySQL, consta de varios elementos, entre los que figuran el servidor MySQL (**mysqld**, que se encarga de ejecutar y administrar las bases de datos), el cliente MySQL (**mysql**, que proporciona una interfaz para el servidor) y numerosas utilidades de mantenimiento de otra índole. Se puede interactuar con MySQL usando lenguajes de programación más populares, como PHP, Perl, Java y C# tal y como hemos hecho en la realización de este proyecto.

MySQL fue escrito en C y C++ y funciona exactamente igual en distintos sistemas operativos. Se sabe que MySQL es capaz de manejar bases de datos de hasta 60.000 tablas y más de cinco mil millones de filas. MySQL puede trabajar con tablas de hasta ocho millones de terabytes (a partir de la versión 3.23) en algunos sistemas operativos y hasta 4 GB en otros.

MySQL, en este momento va por la versión 5 y la versión utilizada para el desarrollo de este proyecto es MySQL 4.1.15-nt via TCP/IP.

3.6.2. El lenguaje estándar SQL.

El lenguaje SQL (*Structured Query Language*) es el lenguaje actual para los sistemas de bases de datos relacionales. Fue desarrollado originalmente por IBM a mediados de la década de los setenta, e implementado por primera vez en un prototipo de IBM, el System R.

En el año 1986, el lenguaje SQL fue propuesto por ANSI como lenguaje relacional, y fue aceptado en 1987 por ISO como lenguaje estándar. Versiones posteriores de este lenguaje han aparecido en 1989, 1992 y 1999.

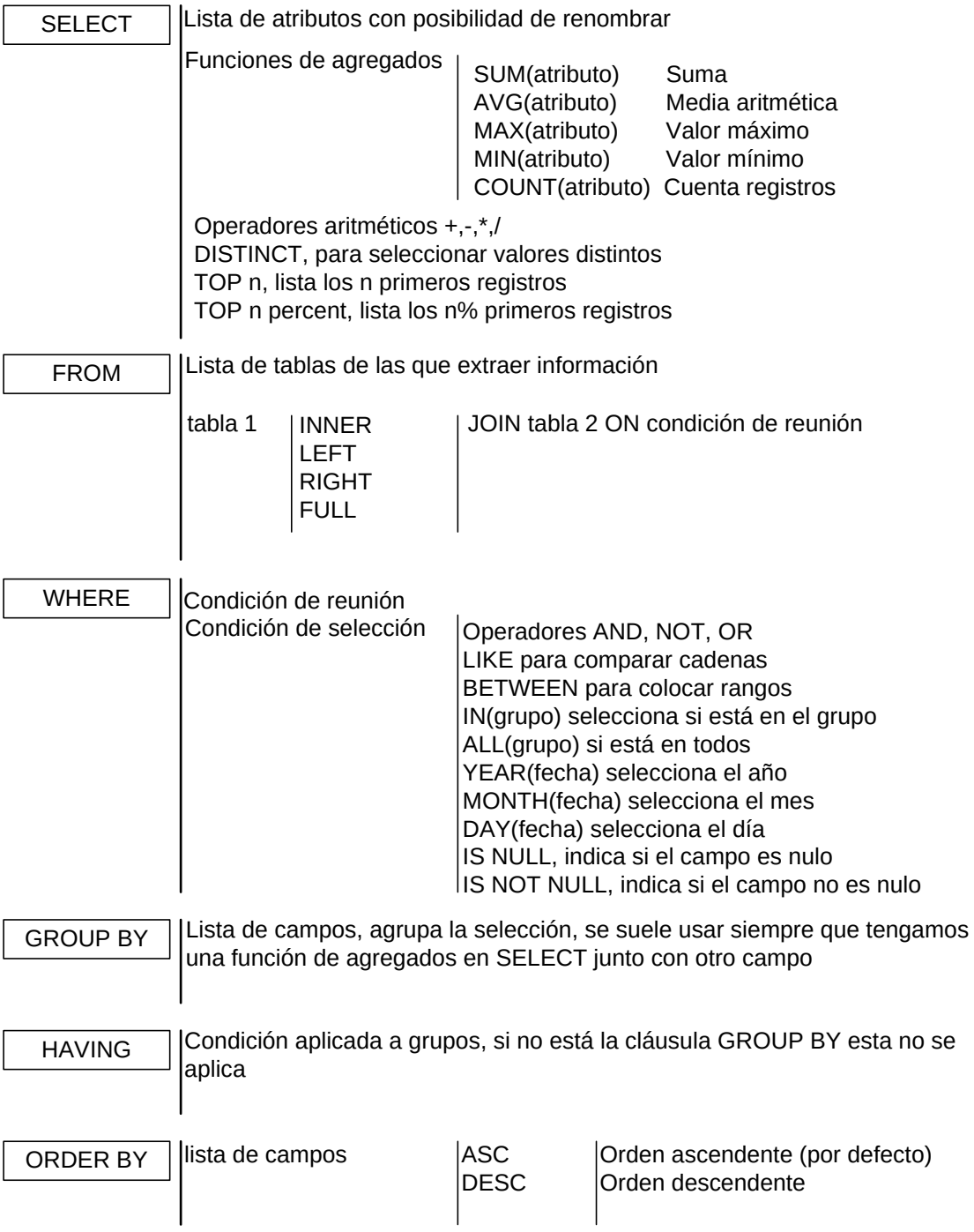
El lenguaje SQL proporciona un sublenguaje de definición de datos (DDL) y un sublenguaje de manipulación de datos (DML), así como otros componentes de control de datos que no pueden ser considerados ni de definición ni de manipulación: control de transacciones, control de usuarios, etc. El lenguaje puede ser utilizado también en modo incrustado (o embebido), es decir, en un programa escrito en un lenguaje de alto nivel (C, Java, etc). Para ello, introduce algunos componentes auxiliares que permiten manipular las tablas de una relación individualmente.

Este lenguaje fue pensado inicialmente como un “lenguaje de datos”; de hecho, SQL, no es computacionalmente completo, en el sentido de que no permite la definición de cualquier función computable. Posteriores desarrollos han consistido en incorporarle un sublenguaje que permite almacenar módulos de programas, con lo el lenguaje se hace más completo.

A continuación se va a realizar un análisis del lenguaje SQL.

3.6.3. Sintaxis SQL.

A continuación se muestran dos esquemas que resumen el conjunto de operaciones que se puede realizar sobre una base de datos.



Esquema sentencia SELECT SQL

Un tipo característico se consultas SQL son las consultas anidadas, como la que se muestra a continuación:

```
SELECT * FROM empleado
WHERE empleado NOT IN (SELECT empleado FROM nomina)
```

Otro tipo de sentencias son las referentes a la manipulación de datos:

INSERT INTO	nombre de tabla VALUES (colocar un valor para cada campo) nombre de tabla (lista de campos a insertar) VALUES (lista de valores)* para realizar esto hay que tener VALUES el mismo número de valores que campos y se tienen que relacionar en el mismo tipo de dato SELECT …, esto indica que podemos complementar esta con una consulta y de esta forma introducir tantos registros como devuelva la sentencia de consulta
DELETE	* FROM nombreTabla WHERE condición de eliminación
UPDATE	

Esquema sentencias de Manipulación de Datos

3.7. DirectShow

La implementación software del dispositivo de captura se ha realizado mediante un proyecto denominado *DShowNET* basado en *DirectX*. Dicho proyecto contiene todas las interfaces y clases necesarias para detectar las posibles interfaces gráficas del sistema en que se ejecuta la aplicación. Además permite ir obteniendo el flujo de video necesario para transmitir. A continuación se realiza una breve descripción de la arquitectura *DirectShow*.

Microsoft DirectShow es una arquitectura, incluida dentro del grupo del *DirectX*, que se creó para ayudar a la manipulación de archivos multimedia dentro de la plataforma Windows. *DirectShow* proporciona funciones para la captura de multimedia con buena calidad, además de reproducción de streams multimedia.

Soporta gran variedad de formatos, incluyendo:

- **ASF:** Advanced Streaming Format.
- **MPEG:** Motion Picture Experts Group.
- **AVI:** Audio-Video Interleaved.
- **MP3:** MPEG Audio-Layer 3.
- **WAV:** WAVEform audio format.

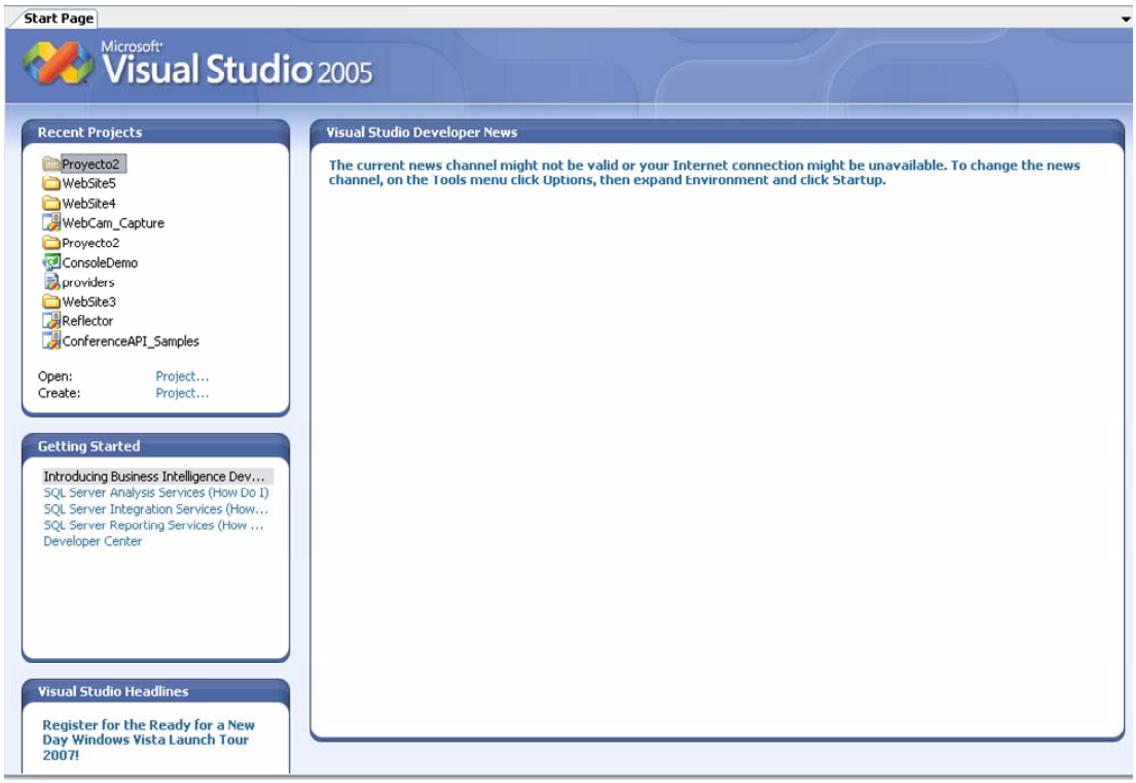
Soporta también la captura de usando los dispositivos WDM (*Windows Driver Model*) o incluso los más viejos “Video for Windows”.

La forma de trabajar de esta arquitectura se basa en la utilización del hardware de aceleración de vídeo y audio, cuando detecta que está disponible, aunque también existe la posibilidad de trabajar sin estos dispositivos. *DirectShow* simplifica todas las tareas de tratamiento multimedia, proporcionando acceso a la arquitectura de control de flujos, para que las aplicaciones puedan crear sus propias soluciones.

En el núcleo del *DirectX* están sus interfaces de aplicación a la programación, o API’s. Las API’s actúan como un tipo de puente entre el hardware y el software para “hablar” entre ellos. Las API’s *DIrectX* dan a las aplicaciones multimedia acceso a las características avanzadas del hardware de alto nivel, tal como chips de aceleración de gráficos tridimensionales y tarjetas de sonido. Controlan además las funciones de bajo nivel, incluyendo la aceleración de gráficos dimensionales; dan soporte a los dispositivos de entrada, como teclado y micrófono; y controlan además el tratamiento del sonido y su salida. Gracias al *DirectX* el trabajo con los gráficos 3D y la creación de efectos de música y audio son mucho más fáciles y productivos.

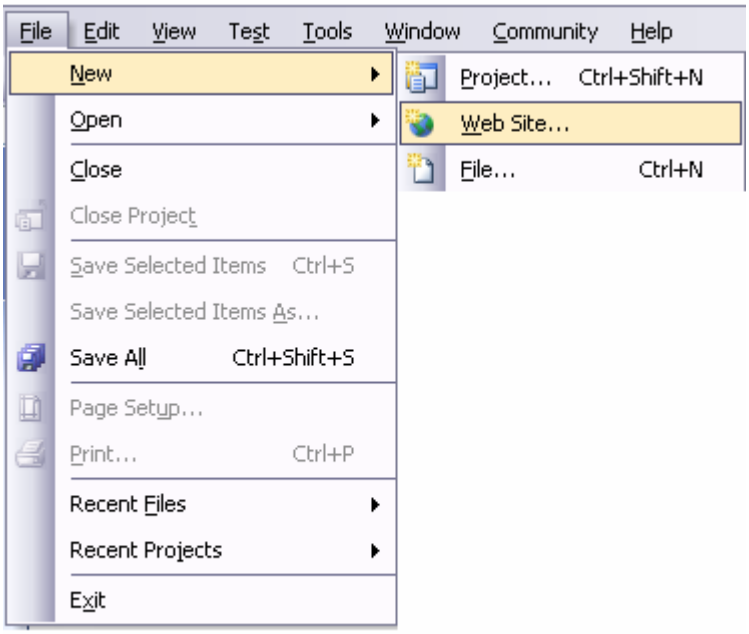
4. Creación de una aplicación con ASP.NET.

Al iniciar Visual Studio 2005, se muestra la página de inicio, como muestra la figura 1. Desde esta página, se pueden abrir proyectos existentes y crear nuevos proyectos, incluidos los proyectos de ASP.NET.



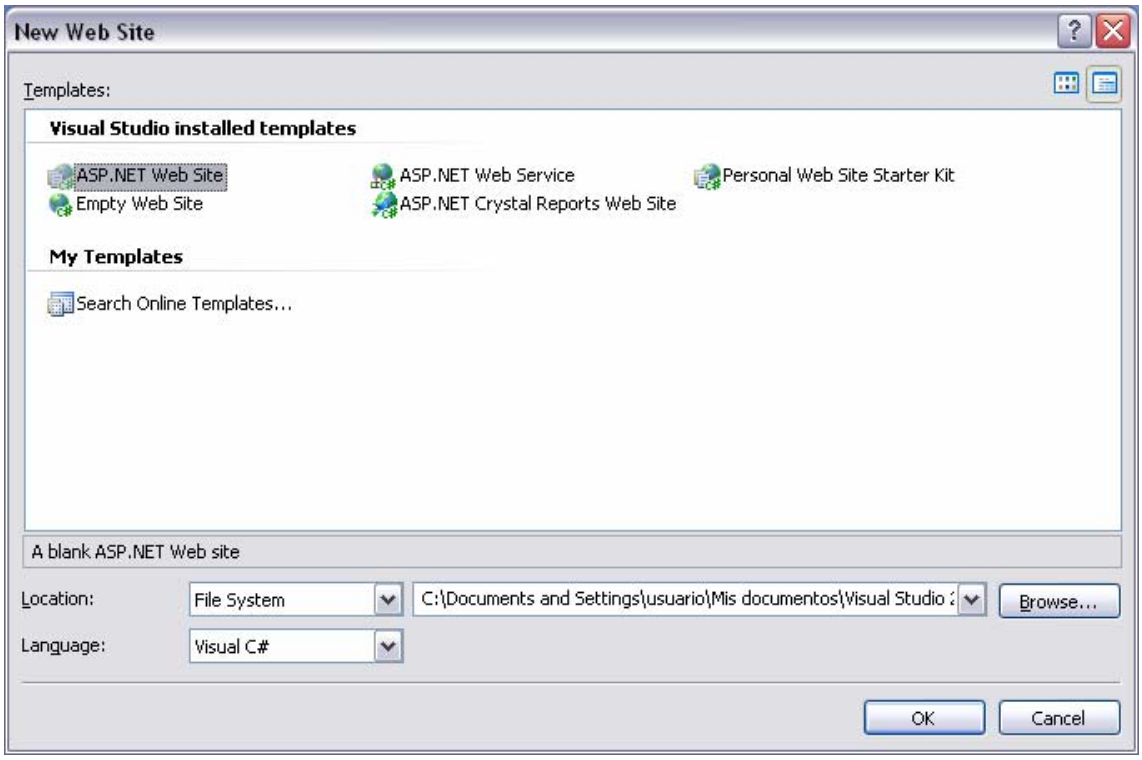
Página de Inicio de VS2005

Para crear una aplicación ASP.NET, hay que ir a la barra de menús y dentro del menú "*File*" e indicar que se va a crear un nuevo "*Web Site*":



Menú para crear nuevo Web Site 1

Una vez seleccionada esta opción, aparecerá la siguiente ventana:

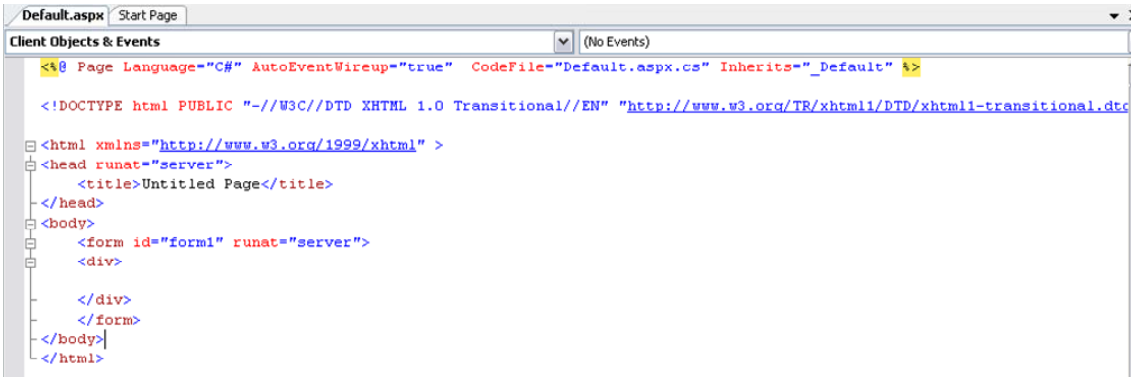


Menú para crear nuevo Web Site 2

Tal y como se puede apreciar, es en esta ventana donde se puede elegir el código en el que se programará el *Codebehind* (ver apartado **3.4.6.1.2**) de la aplicación ASP.NET. Las opciones para el *Codebehind* son hacerlo en *Visual Basic*, *Visual J#* o *Visual C#.*

Una vez elegido el lenguaje de programación a utilizar, se confirma mediante el botón "*OK*".

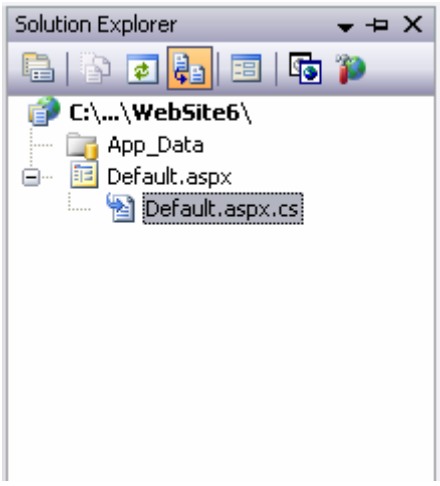
Tras crear el nuevo *Web Site* aparece un fichero llamado por defecto *Default.aspx* con un formato tal y como se muestra a continuación:



Página Default.aspx

En esta primera página aparece lo que es la estructura básica de una página aspx con su directiva *Page* además del código html correspondiente.

Por defecto, además del documento *Default.aspx*, se crea el fichero de *Codebehind* asociado a la página de aspx, tal y como se puede observar en el "*Solution Explorer*":

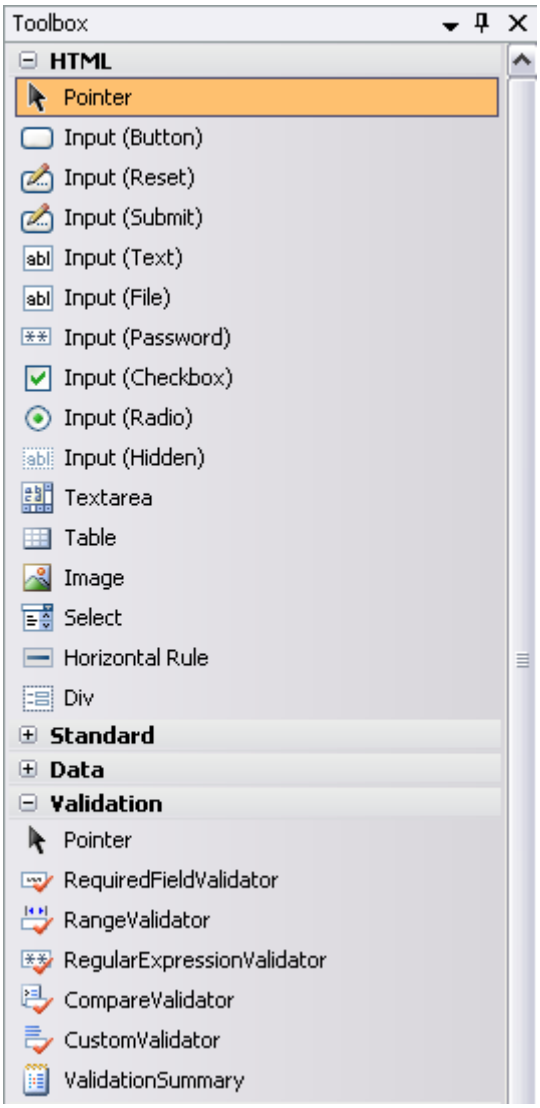


Solution Explorer

Es en este documento donde se podrá programar todos las acciones asociadas a los eventos tales como el evento *Load* del fichero aspx o el evento *OnClick* de algún botón de la aplicación ASP.NET.

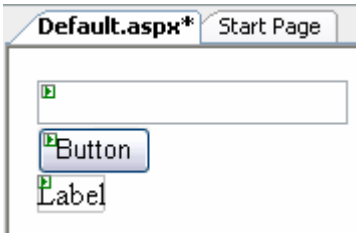
Volviendo al fichero *Default.aspx* hay que destacar las dos posibilidades de trabajo, o bien se trabaja en modo "*Source*", que es como lo mostrado anteriormente, o en modo "*Design*".

En el modo *Design* se ha de tener en cuenta una barra de herramientas nueva, esta barra de herramientas, llamada "*Toolbox*" proporciona una serie de componentes ASP.NET que están creados por *Visual Studio 2005* y que el programador deberá de olvidarse de crear, la preocupación de éste se basará en el aspecto del mismo que modificará por mediación de las propiedades de cada componente:



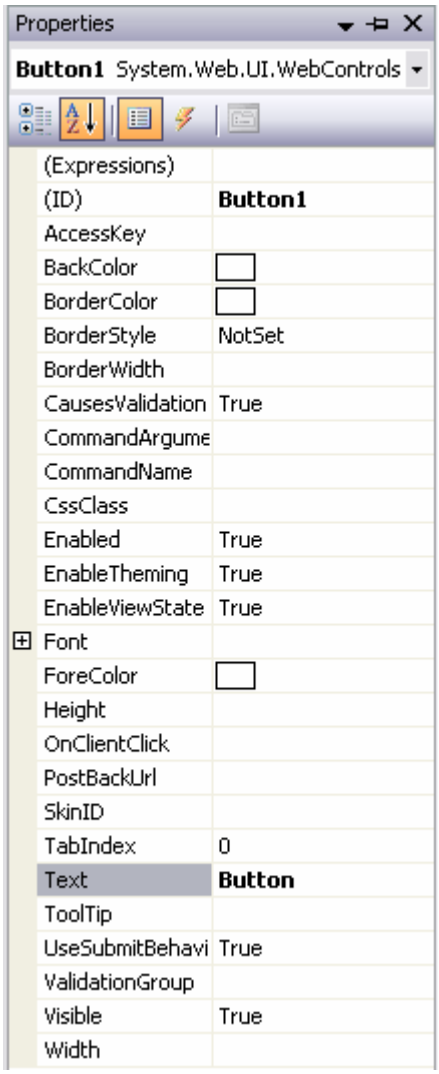
ToolBox

La forma de emplear estos componentes es sencilla, basta con arrastrar el componente deseado a la vista de diseño, por ejemplo, un "*TexBox*",un "*Button*" y un "*Label*". La intención de este Sitio Web es que el usuario introduzca su nombre en el *TextBox* cuando pulso el *Button* se muestre un saludo con su nombre en el *Label*.



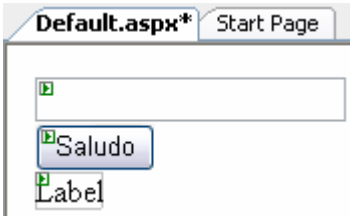
Página en Vista de Diseño 1

Tal y como se ha indicado anteriormente, el programador modificará los componentes mediante las propiedades, por ejemplo, se desea modificar el texto del botón, para ello, por mediación del botón derecho del ratón se elegirán las propiedades y aparecerá un menú como el que sigue:



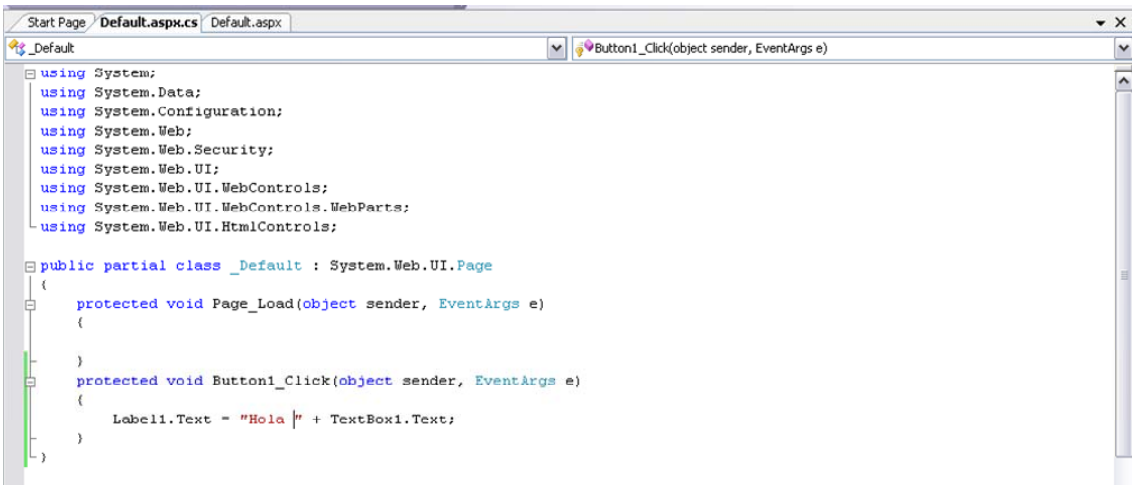
Properties

Dentro de este menú, y concretamente en la propiedad de "*Text*" se modificará el texto que aparece sobre el botón, indicando ahora "Saludo":



Página en Vista de Diseño 2

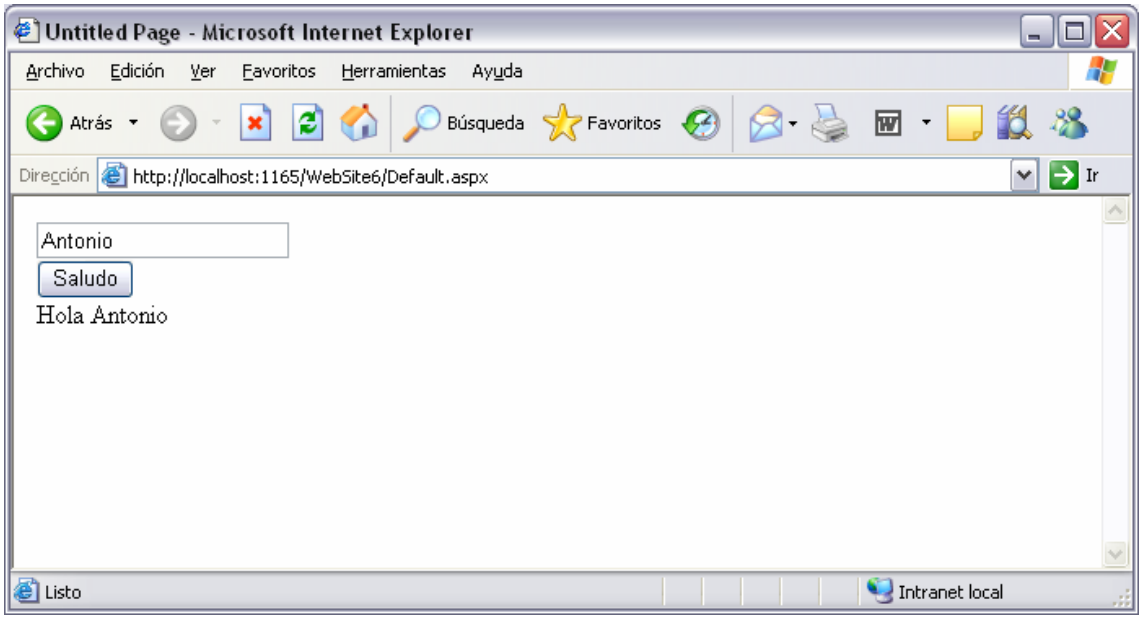
A continuación se debe programar el evento mediante el cual de mostrará al usuario un saludo con el nombre introducido en el *TextBox* y que se programará haciendo doble clic sobre el botón:



Página Default.aspx.cs

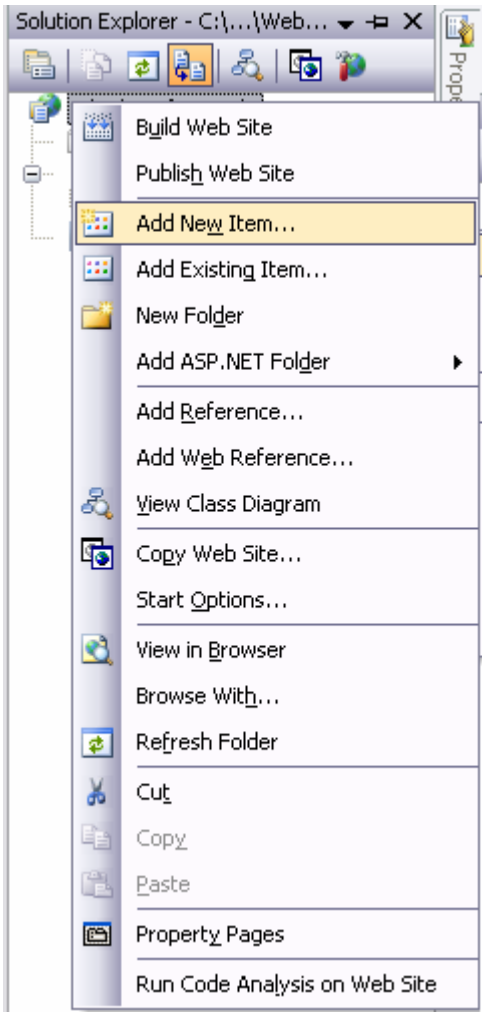
Tal y como se puede observar, el evento programado es el evento "*Clik*" del botón insertado en el fichero *Default.aspx*. Este evento esta programado en el fichero *Default.aspx.cs* de tal forma que asigna a la propiedad *Text* del elemento *Label* un texto compuesto por "Hola" más el texto introducido en el *TextBox*.

Para ejecutar la aplicación basta con pulsar F5, de esta forma, la aplicación se ejecutará en nuestro navegador:



Navegador ejecutando la aplicación

Por otro lado, si la aplicación consta de más de un fichero *aspx* con sus correspondientes ficheros *Codebehind*, el programador los podrá añadir fácilmente a través del "*Solution Explorer*":



Añadir Nuevo Fichero

Al seleccionar esta opción, aparecerá un menú con los distintos tipos de archivos que se pueden agregar al Sitio Web y además la opción de elegir el lenguaje en el que se programará, por lo tanto existe la posibilidad de crear una aplicación ASP.NET en la que las distintas páginas estén programadas en distintos lenguajes sin que el programador se preocupen del entendimiento entre las mismas. Estas es una de las grandes ventajas de esta herramienta.

5. Conexión de una Base de Datos MySQL a una aplicación ASP.NET.

A continuación se desarrolla un ejemplo de conexión de una base de datos creada en MySQL con una aplicación ASP.NET.

La base de datos que se va a utilizar para el desarrollo de este ejemplo va a ser una sencilla tabla con los datos de los usuarios, la cuál se conecta con la aplicación ASP.NET que se trata de un sencillo formulario que los usuarios realizarán. La tabla utilizada es:

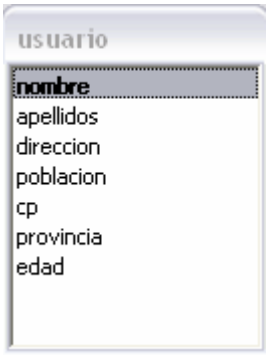


Tabla Usuario

A continuación se muestra el aspecto del Formulario Web:

Formulario Web

Tal y como se puede observar, el usuario introducirá todos los campos que hay en la base de datos y cuando pulse “Enviar datos”, éstos se guardarán en la misma.

Es importan, antes de continuar con el ejemplo, ver la librería utilizada para conectar una base de datos MySQL. Esta librería se encuentra en **System.Data.Odbc**:

Clases

- **OdbcCommand:** Representa una sentencia SQL a ejecutar sobre una base de datos MySQL.
- **OdbcCommandBuilder:** Genera automáticamente comandos de tabla-sencilla usados para aplicar los cambios hechos a un dataset (información codificada en una estructura definida como puede ser una tablas, una base de datos, etc.) con la base de datos asociada.
- **OdbcConnection:** Representa una conexión a un servidor de base de datos.
- **OdbcDataAdapter:** Representa un conjunto de comandos y una conexión que son usados para llenar un dataset y actualizar una base de datos.
- **OdbcDataReader:** Provee un medio de lectura de filas.
- **OdbcException:** La excepción que es lanzada cuando MySQL devuelve un error.
- **OdbcTransaction:** Representa una transacción SQL que se hará en una base de datos.

Otro concepto importante ha tener en cuenta, ya que sin el no se podría conectar la aplicación con la base de datos, es que hay que instalar el conector de ODBC, en este caso el conector es el **MySQL ODBC 3.51 Driver** que se le deberá indicar a la aplicación por medio de nodo <connectionStrings> en el fichero *Web.config***:**

```
<connectionStrings>
      <add name="BaseDatos" connectionString="DRIVER={MySQL ODBC 3.51
      Driver};SERVER=localhost;DATABASE=usuario;USER=root;
      PASSWORD=1234;"/>
</connectionStrings>
```

Entre los parámetros que hay que indicar, cabe destacar el parámetro **name** el cual se utilizará posteriormente a la hora de crear la conexión a la base de datos.

El procesamiento de la conexión a la base de datos y envío de los datos a la misma se realiza en el fichero de *Codebehind* asociado al formulario, más concretamente en el evento *OnClick* del botón de "Enviar Datos".

Lo primero que se realiza es la conexión a la base de datos:

```
OdbcConnection conexion = new OdbcConnection
(ConfigurationManager.ConnectionStrings["BaseDatos"].ConnectionString);
```

Tal y como se puede observar, en la conexión se indica el nombre del connectionString que previamente ha sido definido en el fichero *Web.Config.*

Una vez realizada la conexión hay que crear la cadena que se ejecutará sobre la base de datos, esta cadena puede ser una consulta, una modificación, etc. En este caso, puesto que se trata de introducir datos, la cadena utilizada es:

```
string cadena1 = "INSERT INTO usuario
      (nombre,apellidos,direccion,poblacion,cp,provincia,edad) VALUES
      ('" + TextBoxNombre.Text + "','" + TextBoxApellidos.Text + "','"
      + TextBoxDireccion.Text + "','" + TextBoxPoblacion.Text + "','" +
      TextBoxCP.Text + "','" + TextBoxProvincia.Text + "','" +
      TextBoxEdad.Text  + "')";
```

Una vez realizadas la conexión y la cadena, se debe crear el comando de conexión por medio de la clase **OdbcCommand**. Este comando es el que se utiliza conectar y abrir la base de datos:

```
OdbcCommand comando1 = new OdbcCommand(cadena1,conexion);
comando1.Connection.Open();
```

A continuación hay que ejecutar el comando, este comando puede lanzar una serie de excepciones, por lo que se debe introducir en un bloque try/catch:

```
try
        {
            comando1.ExecuteNonQuery();
            MessageBox.Show("DATOS ENVIADOS");
            TextBoxNombre.Text = "";
            TextBoxApellidos.Text = "";
            TextBoxDireccion.Text = "";
            TextBoxPoblacion.Text = "";
            TextBoxCP.Text = "";
            TextBoxProvincia.Text = "";
            TextBoxEdad.Text = "";

        }
catch
        {
            MessageBox.Show("NO SE HAN PODIDO ENVIAR LOS DATOS");
        }
```

El funcionamiento del código anterior es tal que si la ejecución del comando es correcta se le informa al usuario de ello además de poner en blanco todo el conjunto de TerxtBox del Formulario Web. Por el contrario, si la ejecución del comando no ha sido satisfactoria también se le informa al usuario de ello.

Por último, una vez ejecutado el comando y al no ser necesario que la conexión a la base de datos se mantenga abierta, ésta es cerrada:

```
comando1.Connection.Close();
```

6. Instalación y configuración de Internet Information Server.

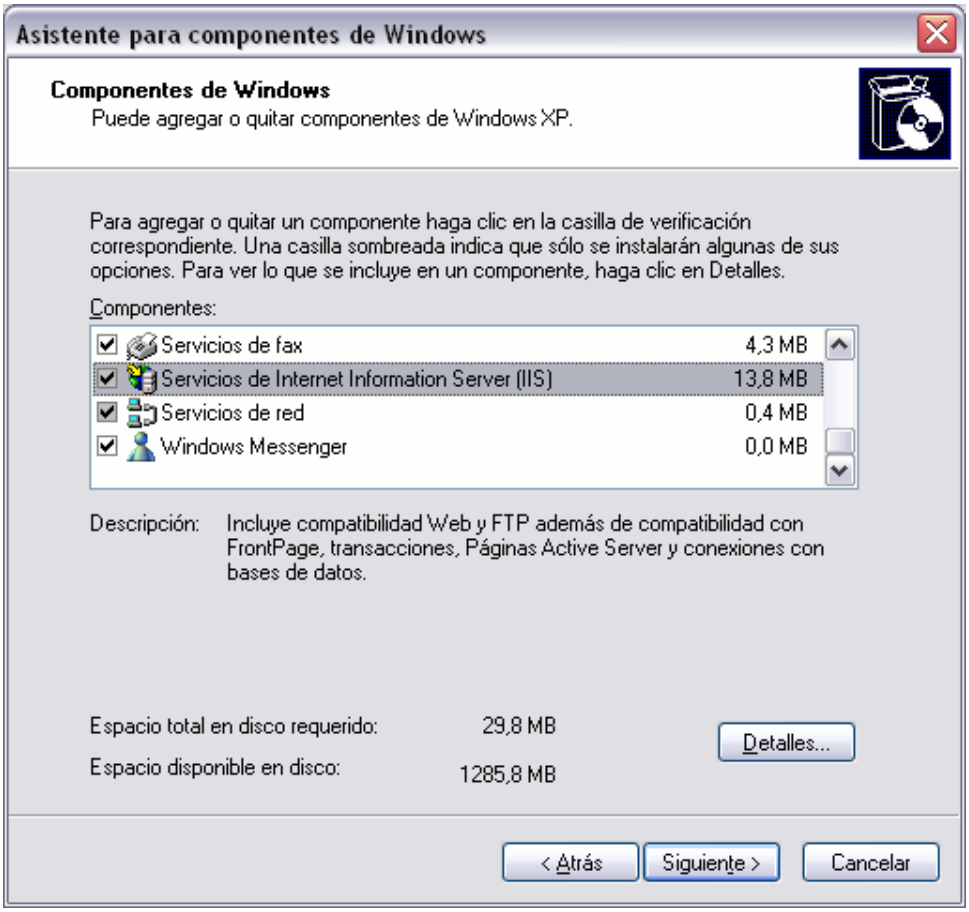
Las aplicaciones ASP.NET se suelen ejecutar con IIS (*Internet Information Server*) como servidor Web.

6.1. Instalación de IIS.

Para instalar IIS en el sistema operativo Windows XP hay que ir a la aplicación **Agregar o quitar programas** del Panel de Control.

Los pasos a seguir son:

1. Hacer clic en **Inicio**, seleccionar **Configuración**, y una vez dentro del **Panel de Control** hacer doble clic en **Agregar o quitar programas**.
2. Seleccionar **Agregar o quitar componentes de Windows** y seleccionar la opción de IIS. Insertar el CD de instalación de Windows y se instalará.



Ventana Componentes de Windows

6.2. Crear y configurar directorios virtuales en IIS.

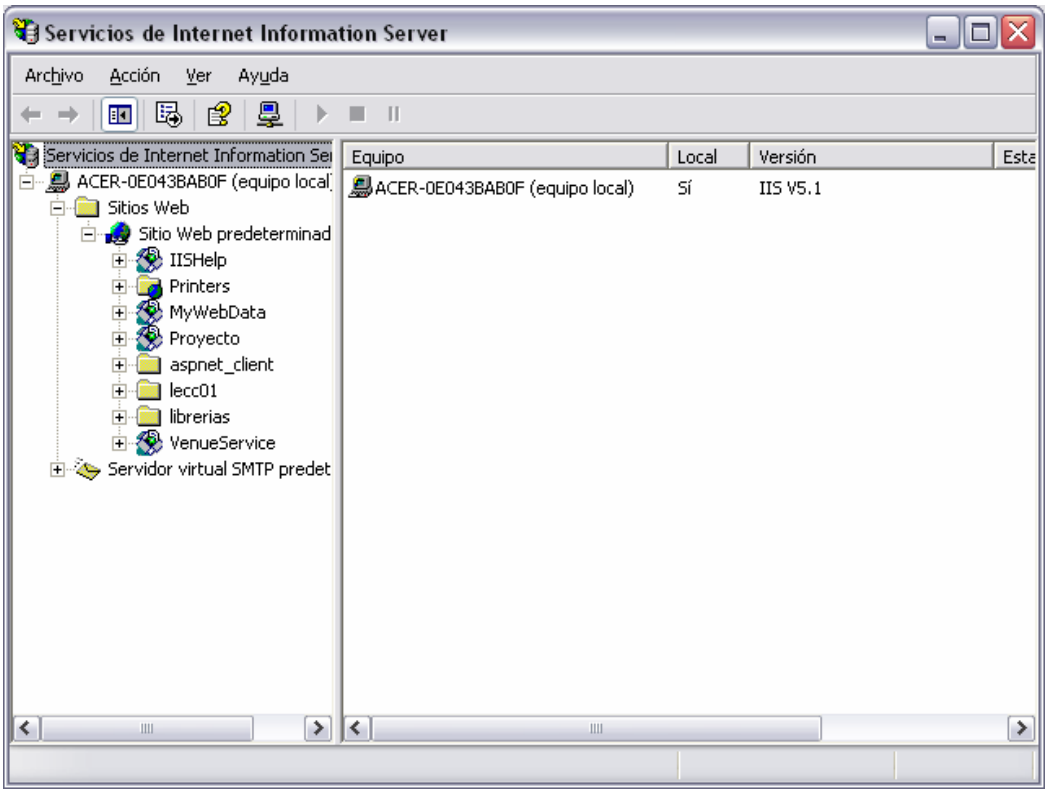
Con el Administrador de Servicios de Internet Información Server (IIS) se puede crear un directorio virtual para una aplicación Web ASP.NET. Un directorio virtual es un directorio que se encuentra en una dirección distinta de la que se mapea en la URL. El directorio virtual no tiene por que encontrarse dentro de los directorios que se crean en el PC, se puede encontrar en cualquier otra ubicación, incluso en otro ordenador distinto. Es en estos directorios virtuales donde se debe de localizar la aplicación para su correcto funcionamiento.

Para crear un directorio virtual, se debe crear un sitio Web en el servidor . IIS crea un sitio Web predeterminado en el equipo durante la instalación.

6.2.1 Iniciar el Administrador IIS.

Para iniciar el Administrador de IIS desde la consola Servicios Administrativos:

1. Ir al **Panel del Control** dentro de la opción de **Configuración** del menú **Inicio**.
2. En la ventana del **Panel de Control**, acceder a **Herramientas administrativas**.
3. En la ventana de **Herramientas administrativas**, hacer clic en S**ervicios de Internet Information Server.**



Servicios de Internet Information Server

6.2.2 Crear el Directorio Virtual.

Una vez iniciado el Administrador IIS, ya se puede crear un directorio virtual:

1. En el sitio Web en el que se desea agregar un directorio virtual hay que hacer clic con el botón secundario del ratón, señalar **Nuevo** y a continuación, clic en **Directorio virtual.**
2. A continuación se muestra un asistente para crear el directorio virtual.
3. En el cuadro de **Alias** se introduce el nombre del directorio virtual.
4. En el siguiente pasa se debe indicar el **Directorio** físico que contendrá el directorio virtual.
5. A continuación aparece una serie de opciones, entre ellas las casillas de **Lectura** y **Ejecutar secuencias de comandos** que aparecen seleccionadas por defecto.
6. Por último, se finaliza el asistente.

6.2.3 Configurar el Directorio Virtual.

Después de crear un directorio virtual, se pude configurar para que ejecute páginas ASP.NET y definir su seguridad.

Para configurar la seguridad y la autenticación para un directorio virtual:

Propiedades del Directorio Virtual

1. En el Administrador IIS, con el botón derecho sobre el directorio virtual que se desea configurar, se accede a la opción de **Propiedades.**
2. Dentro de la ficha **Seguridad de Directorios** se puede editar la Autenticación y control de acceso.
3. Entre otras muchas opciones, se pueden configurar opciones tales como los **Permisos de ejecución**, **Protección de la aplicación**, etc.

7. Implementación del Servidor Web.

7.1. El escenario: Tu Servidor de Video

"Tu Servidor de Video" se acerca al usuario para mostrar una serie de videos de toda índole, desde puramente de ocio hasta videos referentes a temas de educación. Este servidor de video proporcionará todas las especificaciones necesarias para que los clientes dispongan de opciones tales como un registro en una base de datos, una autenticación o bien una lista con el número de visitas de cada vídeo.

7.2. Diseño de la aplicación.

El paso más importante al desarrollar una aplicación es el diseño. Sin un diseño adecuado, las cosas se pueden poner muy feas más adelante. Por cada minuto que se invierta en esta etapa se ahorran 10 minutos durante el desarrollo. Por lo tanto, antes de comenzar a escribir código, se deberá sentar una base sólida de diseño.

7.2.1. Los datos de la aplicación.

Primero, se deben analizar los almacenes de datos requeridos para la aplicación. Obviamente, esta aplicación va a necesitar de una base de datos. Específicamente, necesitará rastrear dos cosas: Los clientes y su información y los datos referentes a los vídeos que éstos podrán visitar.

La información que mantiene el Servidor de Vídeos sobre sus clientes es estándar, así que se puede utilizar una pequeña base de datos en la que se incluya información de los clientes, como su nombre, nombre de usuario y contraseña, etc.

Al tener también un registro sobre los vídeos que el cliente visitará, se hace necesario contar con otra base de datos (en este caso otra tabla) que contenga información referente a estos.

También habrá que crear cierta cantidad de procedimientos almacenados. De los datos que se han analizado anteriormente se puede inferir que será necesario procedimientos para ejecutar las siguientes tareas:

- Validar la identificación de un usuario.
- Agregar usuarios a la tabla de clientes.
- Obtener información de la tabla de los vídeos almacenados.

Después de analizar los requisitos para los datos, terminará con algo similar a la figura 1. Existirán dos tablas, que contienen las diferentes categorías que se discutieron anteriormente: usuarios y vídeos. Estas dos tablas no contienen ningún

tipo de relación entre ellas puesto que únicamente son usadas para almacenar la información referente a usuarios y vídeos.

Tablas de la Base de Datos

7.2.2. La interfaz de usuario.

Los requisitos para la interfaz de usuario se derivan directamente de lo expuesto anteriormente. Se necesitará una interfaz en el que el usuario se valide para comprobar que se registro anteriormente, un registro que hizo en un formulario de la aplicación.

Por otro lado se encuentran las interfaces referentes a lo que el usuario podrá realizar en esta aplicación, es decir, visitar una serie de vídeos dispuesto en otra interfaz, y una interfaz adicional en la que podrá obtener una información sobre éstos.

Resumiendo, las páginas que se necesitan para el desarrollo de la aplicación son:

- Página de entrada.
- Página de registro.
- Página para mostrar los diferentes videos.
- Página para visualizar el vídeo elegido por el usuario.
- Página que contenga la información referente a los vídeos.

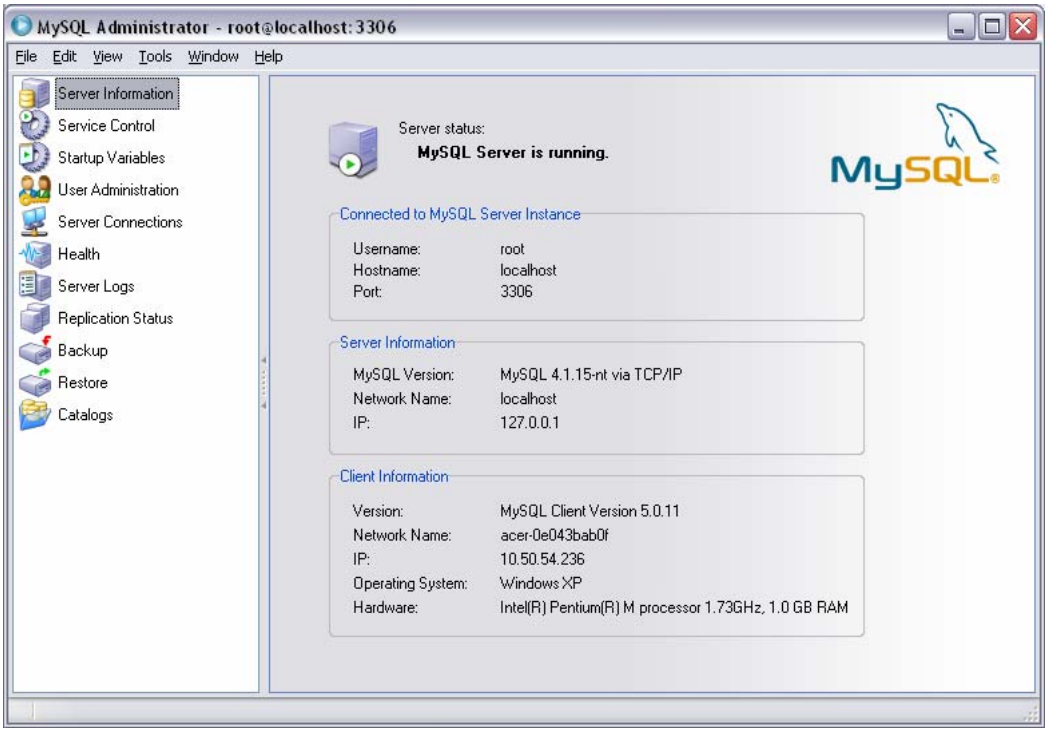
7.3. Creación del Servidor de Vídeo.

7.3.1. Creación de la base de datos.

Lo primero que se ha de elegir, es el sistema gestor de base de datos que se desea utilizar, existiendo varias opciones para ello. Entre las más utilizadas están SQL Server, Oracle y MySQL, entre otros muchos. Los tres sistemas gestores de base de datos mencionados anteriormente permiten el correcto funcionamiento y operabilidad de la

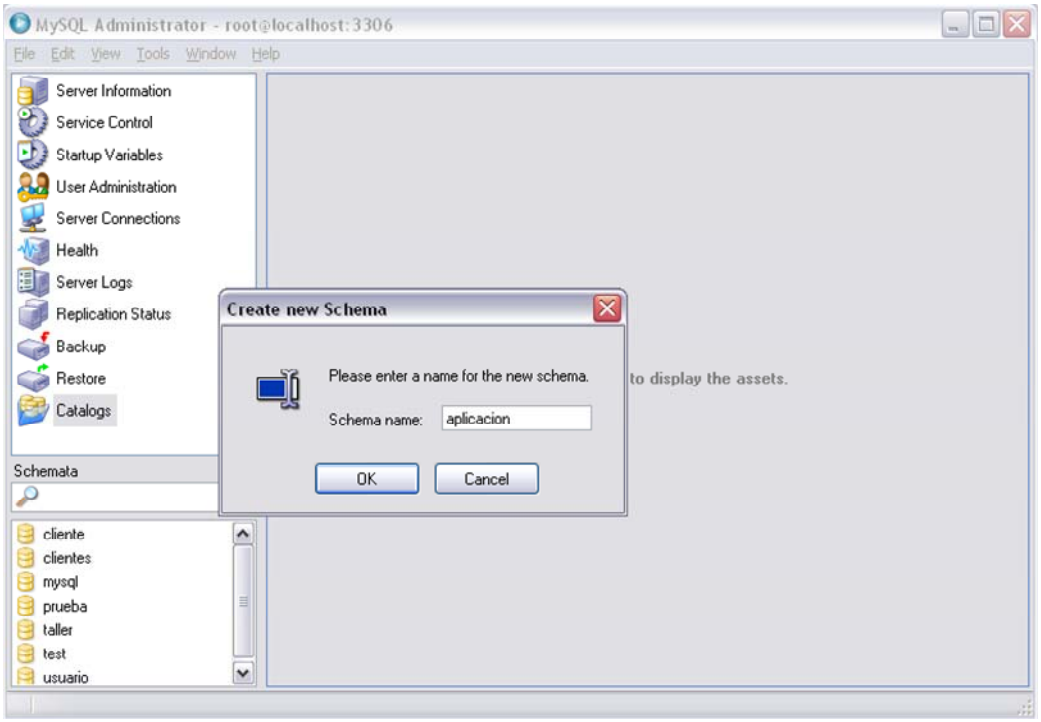
aplicación, pero se ha optado por utilizar MySQL puesto que se trata de un sistema gestor muy extendido en Internet, además de ser una aplicación *open source* como se ha indicado anteriormente.

Una vez, elegido el sistema gestor de da base de datos se ha de tener en cuenta que para crear la base de datos es necesario la aplicación MySQL Administrador que es la que permitirá la creación de la base de datos.



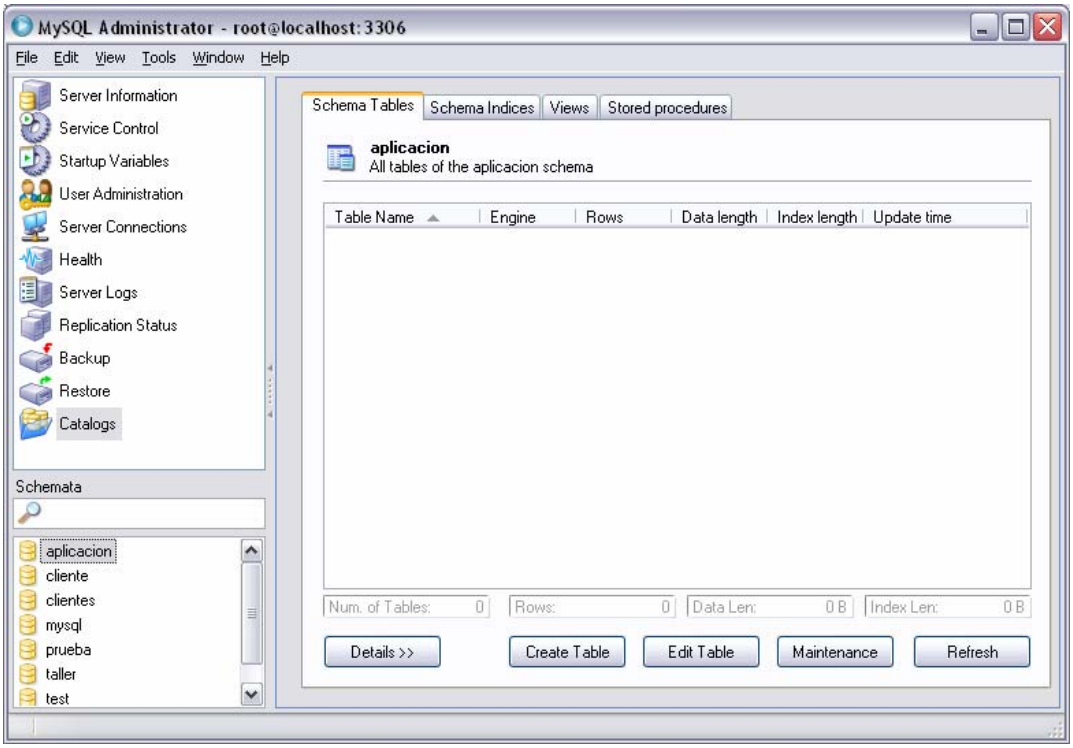
Vista General de MySQL Admistrator

Desde esta vista es donde se creará la base de datos, concretamente, desde el menú *Catalogs* donde se indicará que se desea crear una nueva base de datos a partir de la opción *Create a New Schema*, en este caso, se le dará el nombre de *aplicacion.*



Creación de la Base de Datos

Una vez creada la nueva base de datos, ésta se mostrará sobre la lista de todas las bases de datos creadas. Para crear las tablas, se ha de acceder a ella y aparecerá el menú correspondiente a la figura siguiente:



Menú para la creación de las Tablas

Es desde esta ventana, donde se crearán las dos tablas necesarias para la aplicación:

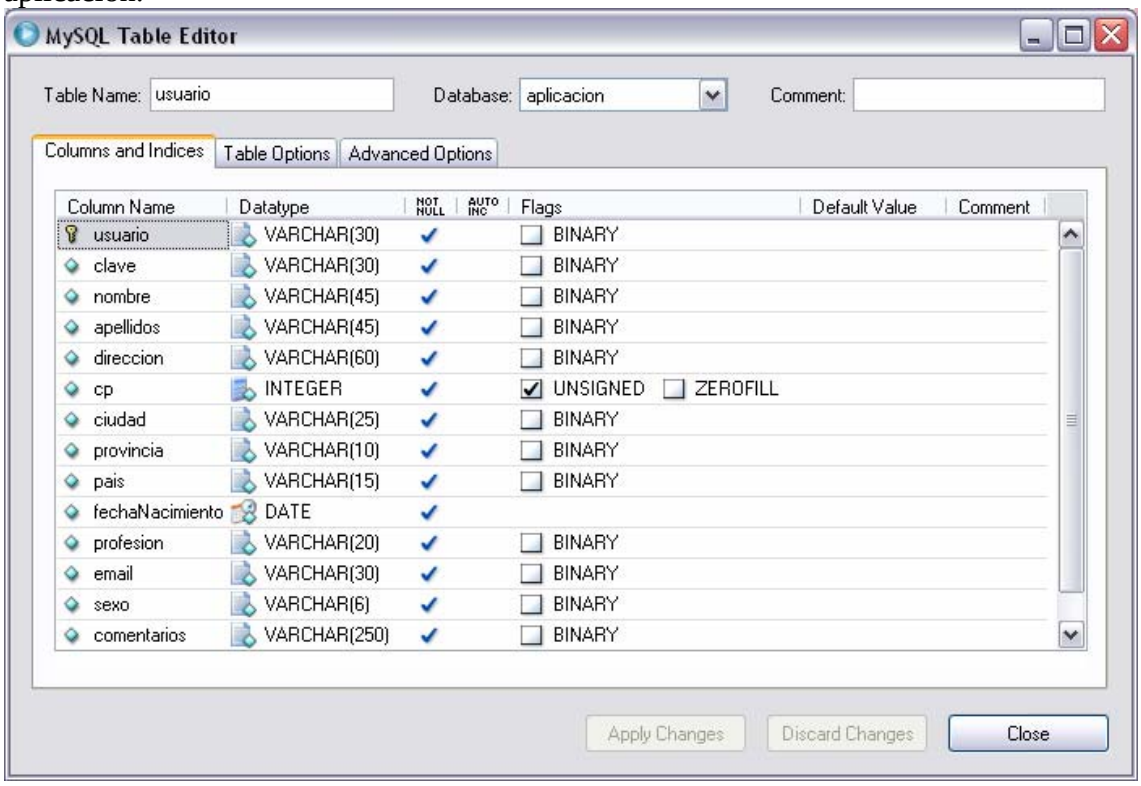


Tabla usuario

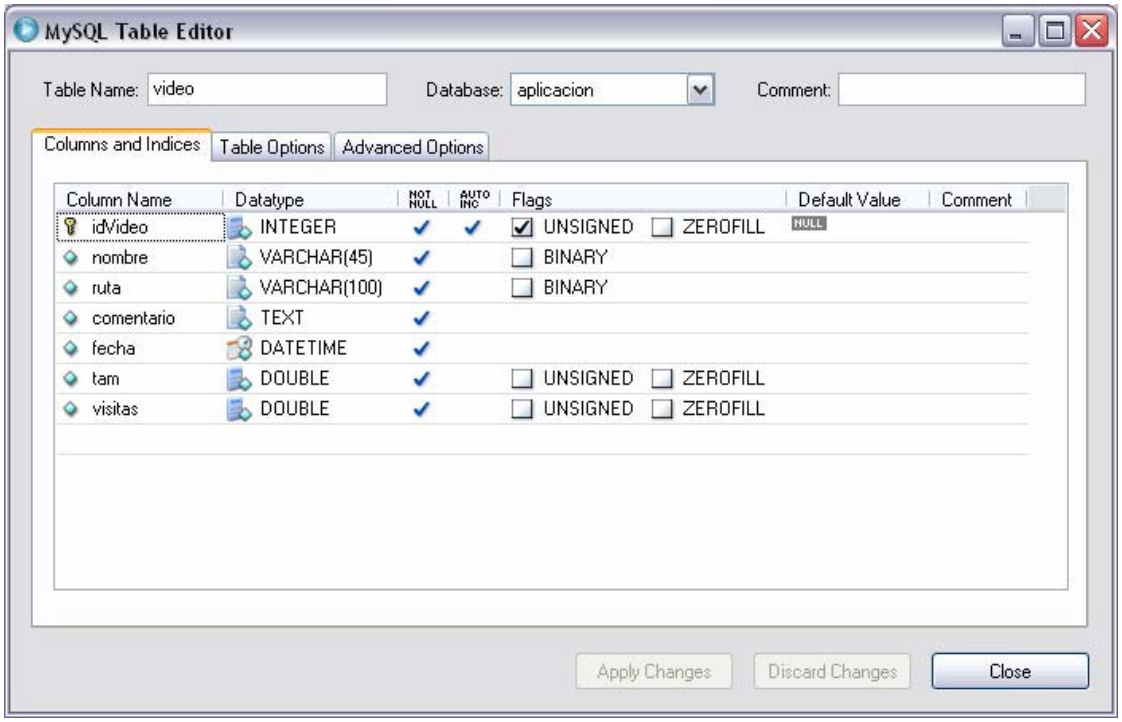


Tabla video

Estos son los datos que permiten almacenar la información referente tanto a los usuarios como a los videos. Es necesario tener en cuenta, que cada tabla creada necesita establecer una clave principal que para la tabla usuario será el nombre de *usuario* y para

la tabla vídeo será el identificador *idVideo*. Esta clave principal es un autonumérico, es decir, se irá incrementando cada vez que se introduzca un nuevo vídeo en la tabla.

A continuación se analizan las diferentes consultas que la aplicación necesitará realizar sobre la base de datos:

Tal y como se ha indicado anteriormente, los diferentes usuarios se deben autentificar ante el sistema, de forma que el usuario y la contraseña sean correctas. Para ello se utilizan tres sentencias **SELECT**, una que comprueba que hay un usuario registrado con el nombre de usuario introducido, otra que comprueba si el password introducido es correcto y otra que comprueba la combinación de ambas, es decir, que el nombre de usuario y password introducido corresponde con un cliente registrado:

```
"SELECT count(*) FROM usuario WHERE usuario='" + Login1.UserName + "'";

"SELECT count(*) FROM usuario WHERE clave='" + Login1.Password + "'";

"SELECT count(*) FROM usuario WHERE usuario ='" + Login1.UserName + "'
and clave='" + Login1.Password+"'"
```

Referente al registro de los distintos usuarios se debe realizar una inserción de datos sobre la tabla *usuario*, dicha inserción de datos se realiza con una sentencia **INSERT** que insertará todos los datos recogidos del formulario Web rellenado por el usuario:

```
"INSERT INTO usuario
(usuario,clave,nombre,apellidos,direccion,cp,ciudad,provincia,pai
s,fechaNacimiento,profesion,email,sexo,comentarios)VALUES('"+Text
BoxNombreUsuario.Text+"','"+TextBoxClaveAcceso.Text+"','"+TextBox
Nombre.Text+"','"+TextBoxApellidos.Text+"','"+TextBoxDireccion.Te
xt+"','"+TextBoxCPostal.Text+"','"+TextBoxCiudad.Text+"','"+TextB
oxProvincia.Text+"','"+DropDownList1.Text+"','"+fechaNacimiento+"
','"+DropDownList6.Text+"','"+TextBoxEmail.Text+"','"+DropDownLis
t5.Text+"','"+TextBox1.Text+"')";
```

En cuanto a los vídeos, el registro de los mismos se hará por el administrador del sistema, por lo que el usuario no influirá para nada en la información de estos, salvo en el campo de visitas que contiene cada uno de los videos, incrementándose su valor. Para incrementarlo se ha de obtener previamente el valor anterior mediante una sentencia **SELECT** y seguidamente modificar el valor de visitas mediante una sentencia **UPDATE:**

```
"SELECT visitas FROM video where idVideo='1'"

"UPDATE video SET visitas='"+visitasNuevas+"' where idVideo='1'"
```

Una característica de estas sentencias es que se realizan con una condición (*where*) ya que la modificación se realizará solamente sobre el vídeo visitado por el usuario.

Por último, referente a las sentencias utilizadas en la aplicación, se ha de rellenar un *GridView* con la información más característica de los videos, está información será, el nombre, el comentario y el número de visitas, por lo tanto la sentencia **SELECT** utilizada es:

```
"SELECT nombre,comentario,visitas FROM video ORDER BY visitas DESC"
```

La característica principal de esta consulta, es que además de devolver los tres campos solicitados para cada video, los devuelve en orden descendente según el número de visitas.

7.3.2. Implementación de la página de inicio.

En la página de inicio se mostrará un componente *Login* en el que el usuario debe introducir su nombre de usuario y su contraseña para comprobar que está registrado. La comprobación del registro se realiza en el fichero *Codebehind* asociado a esta página y que esta programado, al igual que todas las páginas, en C#. Para realizar la comprobación basta con lanzar una consulta **SELECT** sobre la base de datos, esta consulta devolverá un 1 si encuentra un usuario con tal contraseña, si esto es así el usuario quedará autenticado ante la aplicación:

```
contador = Convert.ToInt32(comando.ExecuteScalar());
  if (contador == 1)
    {
       //Damos acceso al usuario

       FormsAuthentication.RedirectFromLoginPage(Login1.UserName,false);

    }
```

La página a la que se redirecciona al usuario se configura en el fichero *Web.config* mencionado a lo largo de esta memoria:

```
<authentication mode="Forms">
      <forms loginUrl="Default.aspx" defaultUrl="Catalogo.aspx">
      </forms>
</authentication>
```

Aquí se le está indicando a la aplicación que todo usuario que acceda a la misma debe pasar primero por la página de inicio (*Default.aspx*) y una vez autenticado se le redirecciona a *Catalogo.aspx.*

En esta página de inicio también se muestra el video de la cámara además de los enlaces a las páginas de Registro, Catálogo y Estadísticas.

7.3.3. Implementación del Formulario Web de Registro.

En esta página se muestra una serie de campos que el usuario ha de rellenar para su posterior registro en la base de datos.

Los elementos utilizados van desde simples *Label*, pasando por los *TextBox* y *DropDownList* que deberá rellenar el usuario y varios controles *RequiredFieldValidator* para comprobar que el usuario ha rellenado correctamente todos los datos solicitados. La vista generar de este formulario es:

Formulario Web

Tal y como se puede observar, los *Label* utilizados le indican al usuario que debe de introducir en cada caso en cada uno de los *TextBox* o *DropDownList*.

Para la correcta introducción de los datos por parte del usuario se han utilizado dos componentes:

- *RequiredFieldValidator*: Este componente obliga al usuario a introducir el dato indicado.
- *CompareValidator*: Este componente compara que la clave de acceso introducida por el usuario coincida con la repetida en el *TextBox* inferior.

La forma que tienen estos controles de obligar al usuario a introducir los distintos datos, es que mientras no se hayan rellenado todos los datos correctamente, éstos no podrán ser enviados por el usuario.

A continuación se desarrolla la programación de los enventos *Click* de cada uno de los botones presentes en este formulario.

Botón "Enviar Datos"

El funcionamiento de este botón, tal y como indica su nombre, consiste en enviar los datos a la base de datos.

Algunos de estos datos deben ser tratados previamente de forma que cumplan el correcto formato exigido por la base de datos, como puede ser el caso de la fecha de nacimiento que ha de tener un formato del tipo 0000-00-00.

También se ha de tener en cuenta de que no pueden existir dos usuarios distintos con el mismo nombre de usuario, para lo cual se realiza una consulta **SELECT** sobre la base de datos y en caso de coincidir con algún usuario registrado se le informará al cliente:

```
string cadena1= "SELECT count(*) FROM usuario WHERE usuario ='" +
TextBoxNombreUsuario.Text +"'";

if (contador == 1)
  {
    //Nombre de usuario ya existente

    LabelUsuario.Text="Nombre de usuario ya existente";

   }
```

Objetivo principal de este botón es almacenar los datos, por lo tanto se deben enviar dichos datos a la base de datos e informar al usuario de la correcta inserción o no de los mismos:

```
string cadena2 = "INSERT INTO usuario
(usuario,clave,nombre,apellidos,direccion,cp,ciudad,provincia,pais
,fechaNacimiento,profesion,email,sexo,comentarios) VALUES ('"+
TextBoxNombreUsuario.Text+"','"+TextBoxClaveAcceso.Text+"','"+Text
BoxNombre.Text+"','"+TextBoxApellidos.Text+"','"+TextBoxDireccion.
Text+"','"+TextBoxCPostal.Text+"','"+TextBoxCiudad.Text+"','"+Text
BoxProvincia.Text+"','"+DropDownList1.Text+"','"+fechaNacimiento+"
','"+DropDownList6.Text+"','"+TextBoxEmail.Text+"','"+DropDownList
5.Text+"','"+TextBox1.Text+"')";

try
  {
    comando2.ExecuteNonQuery();
    LabelInformacion.Text="DATOS ENVIADOS";
  }
catch
  {
    LabelInformacion.Text="NO SE HAN PODIDO ENVIAR LOS DATOS";
  }
```

Botón “Recibir Correo”

Mediante este botón se le da la opción al usuario de recibir un correo electrónico en la dirección introducida por él, que le enviará la confirmación de su registro con el nombre de usuario y contraseña elegido por el. La implementación de esta funcionalidad es:

```
protected void Button2_Click(object sender, EventArgs e)
  {
    //Envio de Correo al usuario registrado

    MailMessage correo = new MailMessage();

    correo.From = new MailAddress("amartinez@jnc.es");
    correo.To.Add(TextBoxEmail.Text);
    correo.Subject = "Confirmacion de Registro";
    correo.Body ="Ha sido registrado con exito en Tu Servidor de
                  Video" +
                  "\nSu nombre de usuario es: " +
                  TextBoxNombreUsuario.Text+"\nSu Clave es:
                  "+TextBoxClaveAcceso.Text+
                  "\nPara cualquier duda, notifiquelo al
                  administrador: amartinez@jnc.es";
    correo.IsBodyHtml = false;
    correo.Priority = MailPriority.Normal;

    SmtpClient smtp = new SmtpClient();
    smtp.Host = "mail.jnc.es";
    string usuario = "usuario";
    string clave = "clave";
    smtp.Credentials = new System.Net.NetworkCredential(usuario,
                       clave);
    try
      {
        smtp.Send(correo);
        LabelCorreo.Text = "Mensaje enviado satisfactoriamente";
      }
    catch (SmtpException ex)
      {
        LabelCorreo.Text="ERROR: " + ex.Message;
      }
  }
```

Lo primero que se ha de hacer es crear el mensaje que se va a enviar. Esto se realiza mediante la clase **MailMessage** del espacio de nombres **System.Net.Mail**, a la que se le indicarán los datos habituales como el origen, el destino, el cuerpo, etc.

Una vez creado el objeto del mensaje, hay que enviarlo, para lo cual se usa una instancia a la clase **SmtpClient** de la que se usarán tres propiedades: **Host** para indicar el nombre del servidor de correo por el que se enviará el mensaje, **Credentials** para autenticarse ante el servidor de correo y **Send** para enviar el correo.

Botón “Ir al Inicio”

Mediante este botón, se le permite al usuario ir de nuevo a la página de inicio. Este se programa fácilmente mediante el objeto **Response**:

```
protected void Button3_Click(object sender, EventArgs e)
 {
    Response.Redirect("Default.aspx");
 }
```

7.3.4. Implementación de la página de Catálogo.

Es esta página donde el usuario puede ver una lista con los diferentes videos disponibles en la aplicación. Al usuario se le muestra una imagen representativa del video, una pequeño comentario sobre el mismo, y se le dan dos opciones, o bien puede verlo directamente en la aplicación o bien descargarlo.

Botón "Ver"

Mediante el evento de este botón se redirige al usuario a una nueva página *vervideo.aspx* en la cual se mostrará el video solicitado.

Se ha de tener en cuenta que para obtener el video solicitado en la página *vervideo.aspx*, se le debe mandar la información necesaria a esta página a través de la URL, para lo cual, la redirección se realiza de la forma siguiente:

```
video = @"Videos\MOV00041.MPG";
Response.Redirect("vervideo.aspx?video="+video);
```

Es decir, se le concatena a la dirección de la página, el lugar donde se encuentra el video dentro del directorio virtual de la aplicación. Posteriormente se mostrará como recuperar dicha información.

Otra funcionalidad que hay que implementar en este botón es la de actualizar el campo "*visitas*" del video solicitado, para lo cual, tal como se ha indicado anteriormente, se utilizan dos sentencias SQL, un **SELECT** para recoger el valor del campo, y un **UPDATE** para modificarlo:

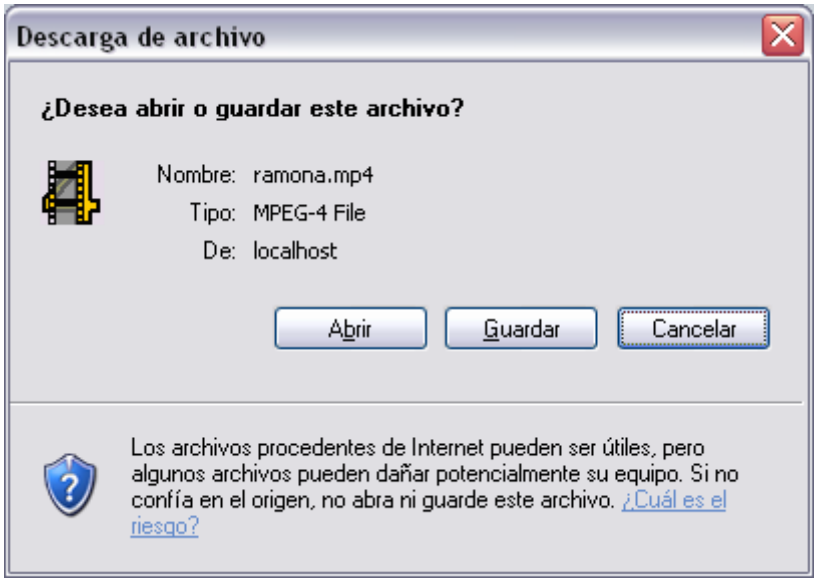
```
"SELECT visitas FROM video where idVideo='1'"

"UPDATE video SET visitas='"+visitasNuevas+"' where idVideo='1'"
```

Botón "Descargar"

En cuanto a la descarga de los ficheros, se utiliza el protocolo HTTP gracias a la siguiente función:

```
public void descarga(string filepath, string filename)
 {
   Response.Clear();
   Response.ContentType = "application/octet-stream";
   Response.AddHeader("Content-Disposition", "attachment;
                      filename=" + filename);
   Response.Flush();
   Response.WriteFile(filepath);
   Response.End();
 }
/*Esta función se puede ejecutar desde donde queramos (lo típico
es llamarlo cuando el usuario presiona un botón). básicamente, lo
que hace es:
1.- Limpia el contenido de salida.
2.- Le cambia el contentType a tipo octet... aquí es donde
      "engañamos al navegador".
3.- Le añadimos la cabecera Content-Disposition y le damos un
      nombre al fichero. Esto es opcional, y lo que hace es dar el
      nombre que queremos que aparezca si el usuario decide
      guardar el fichero.
4.- Manda la info que tenemos hasta ahora (la única cabecera que
      hemos añadido) a la salida hacia el usuario.
5.- Mandamos el fichero en sí desde Response.WriteFile(filepath),
      donde, obviamente, filepath es el path interno del fichero
      en nuestro servidor.
6.- Enviamos todo y terminamos la ejecución de la página.*/
```

Habitualmente, cuando se programa en HTML para forzar la descarga de un archivo basta con poner un enlace al mismo. Cuando el navegador no puede abrir ese tipo de archivo, fuerza la descarga mediante el cuadro de diálogo siguiente:



Descarga de archivo

Esta función implementa exactamente lo mismo. Por lo tanto a la hora de forzar la descarga se utilizará esta función.

7.3.5. Implementación de la página de Visualización de los Vídeos

En esta página se debe mostrar el vídeo solicitado por el usuario, un vídeo cuya información será recogida de la URL. La programación del objeto para mostrar el vídeo es:

```
<object id="MediaPlayer" classid = "CLSID:6BF52A52-394A-11d3-B153-
00C04F79FAA6" width="375" height="260">
      <param name="URL"
      value="<%Response.Write(Request.QueryString.Get("video")); %>" />
      <param name="uiMode" value="full" />
      <param name="stretchToFit" value="false" />
      <param name="windowlessVideo" value="false" />
      <param name="enabled" value="true" />
      <param name="enableContextMenu" value="false" />
      <param name="fullScreen" value="false" />
</object>
```

Es decir *object* dependerá de una serie de parámetros en el que se indicarán las diversas opciones elegidas. Los parámetros más interesantes, ya que de ellos dependerá el tipo de reproductor, son los parámetros *id* y *classid*. Estos parámetros dan la opción de utilizar un reproductor *MediaPlayer* o *QuickTime* entre otros muchos.

Además de los parámetros, cabe destacar el método para recoger la información de la URL:

```
Response.Write(Request.QueryString.Get("video"));
```

De una forma sencilla, se pueden recoger los parámetros pasados por la URL. La URL mandada a esta página es de la forma:

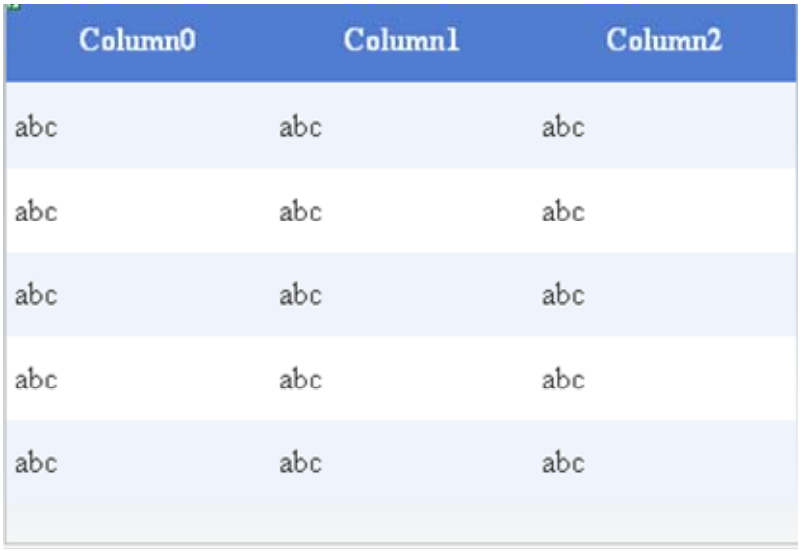
```
http://localhost:2488/Proyecto2/vervideo.aspx?video=Videos\MOV00041.MPG
```

Basta con indicar que parámetro se quiere obtener, en este caso “video”, para recogerlo mediante la función Request.QueryString.Get().

7.3.6. Implementación de la página de Estadísticas.

En esta página se muestra una información referente a todos los videos registrados en la base de datos de la aplicación.

Para mostrar esta información se ha utilizado un componente *GridView* del siguiente formato:



Componente GridView

En este componente se mostrarán tantas columnas como se desee. En este caso se van a mostrar las columnas referentes a nombre, comentario y visitas de la tabla de *video* de la base de datos. Para ello se debe realizar una consulta **SELECT** sobre esta tabla, además de ordenarlo según el número de visitas:

```
"SELECT nombre,comentario,visitas FROM video ORDER BY visitas DESC";
```

Esta sentencia SQL se ejecutará sobre la base de datos y una vez obtenida la información, ésta se mostrará en el componente *GridView*.

```
OdbcConnection conexionGrid = new
      OdbcConnection(ConfigurationManager.ConnectionStrings["BaseDatos].
      ConnectionString);

string cadena="SELECT nombre,comentario,visitas FROM video ORDER BY
               visitas DESC";

OdbcDataAdapter da;

DataTable dt = new DataTable();

try
  {
    da =new OdbcDataAdapter(cadena, conexionGrid);
    da.Fill(dt);

    this.GridView1.DataSource = dt;
    this.GridView1.DataBind();
    Label1.Text = String.Format("Total videos en la tabla: {0}",
            dt.Rows.Count);

  }
  catch (Exception ex)
  {
    Label1.Text = "Error: " + ex.Message;
  }
```

El objeto *OdbcDataAdapter* es el que ejecuta la consulta. Mediante el método *Fill ()* se asocia la consulta con el *DataTable*, posteriormente lo muestra en el *GridView*. Por último se muestra una información referente al número total de videos almacenados.

7.3.7. Visualización de la cámara remota.

Uno de los objetivos de este proyecto es la visualización, a través de la aplicación Web, de una cámara remota conectada al servidor donde se ejecutará la aplicación.

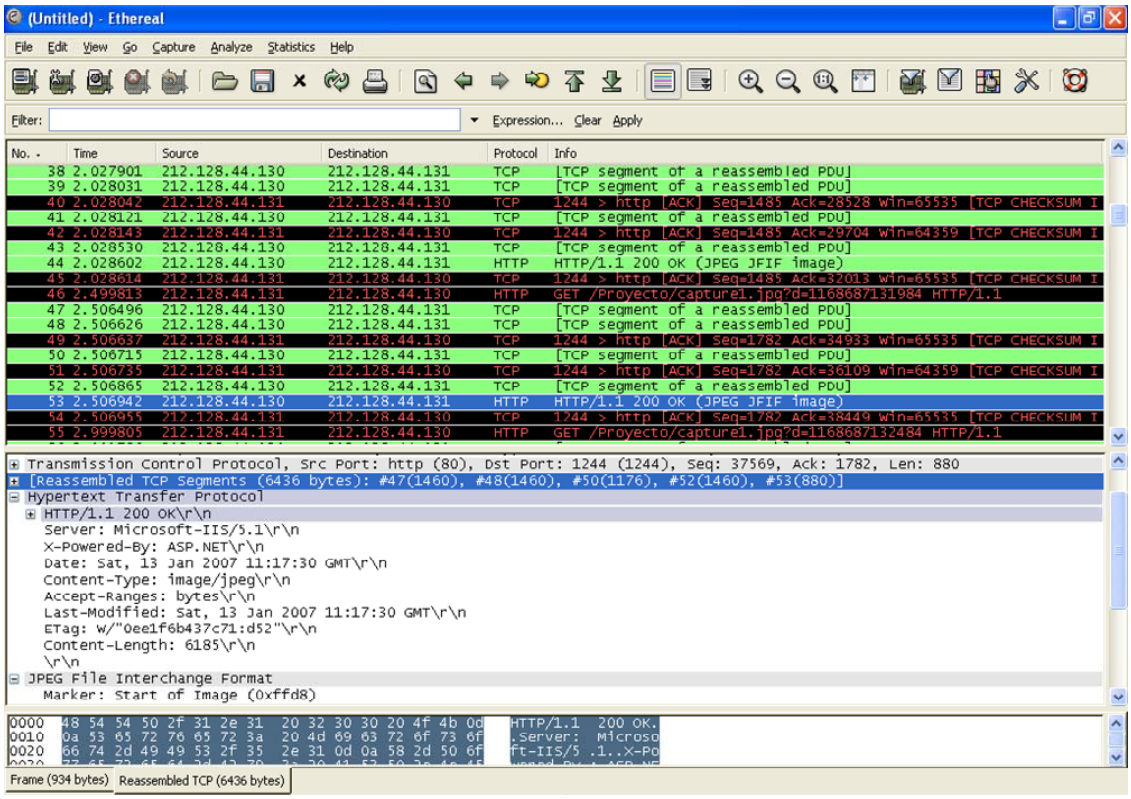
La solución aportada ha sido el desarrollo de un programa a partir de un software que permite la captura de imágenes de la cámara Web de forma periódica. El método para realizar la captura está basado en los filtros *DirectShow* (ver apartado 3.7.).

Estas imágenes son guardadas en el directorio virtual de la aplicación. Es la aplicación, la que va refrescando las imágenes de forma periódica. Las funciones utilizadas son:

```
<img name="webcam" alt="" src="capture1.jpg"/>
periodo = 1;
  imgsrc = document.webcam.src;
  buffer = new Image();
  buffer.onload = imageChange;
  buffer.onerror = imageError;
  setTimeout("imageReload()", periodo * 1000);

  function imageReload() {
    buffer.src = imgsrc + "?d=" + new Date().getTime();
      setTimeout("imageReload()", periodo * 1000);
  }

  function imageChange() {
    document.webcam.src = buffer.src;
  }

  function imageError() {
      setTimeout("imageReload()", 1000);
  }
```

El código anterior hace lo siguiente: crea el objeto imagen al que se le llama *buffer* y le asigna a su evento *onload* una función que copia la imagen recargada en el buffer a la imagen visible. Por otro lado, pone un temporizador que se ejecuta cada *periodo* segundos, y que llama a una función que comienza la carga de la nueva imagen en el buffer. Para que en cada ocasión que se ejecute cargue una imagen distinta se pone el un tiempo que garantiza que cambie entre recargas.

Por último hay que indicar que la transferencia de estas imágenes se realiza a través de HTTP, tal y como se muestra en la siguiente captura de Ethereal:



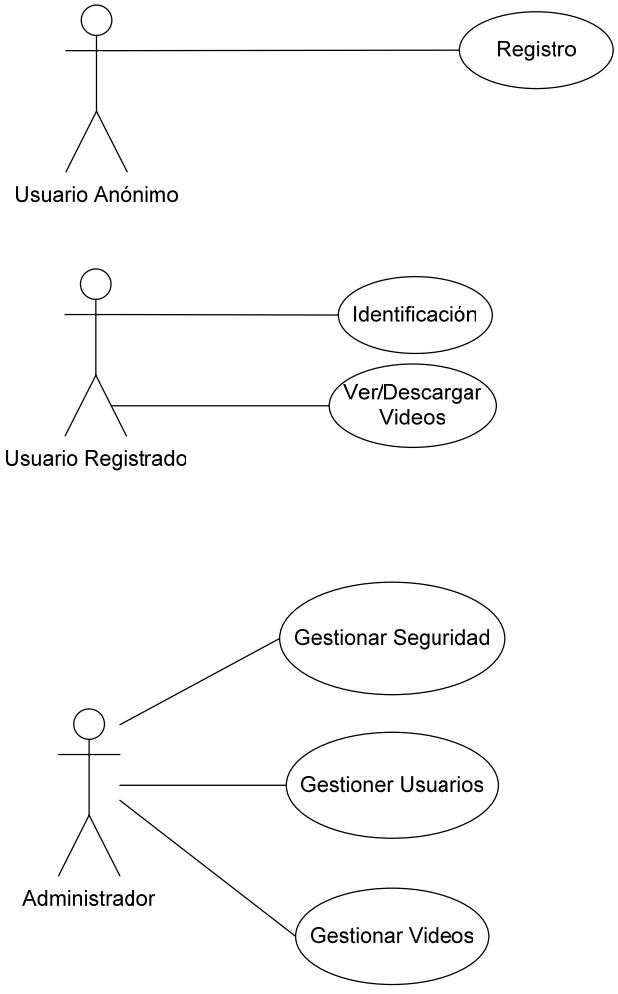
Captura Ethereal

8. Desarrollo de los Diagramas UML de la aplicación.

Uno de los aspectos más importantes en el proceso de desarrollo de software es determinar que necesidades del cliente debe cubrir el sistema. Las capacidades que los clientes exigen a las aplicación tradicionales (no Web) cubren necesidades referentes básicamente a funcionalidad (qué debe hacer el sistema), almacenamiento de la información (que información se va a manejar el sistema) u otras características no funcionales como la facilidad de uso y el rendimiento.

Una aplicación Web no deja de ser una aplicación software, y por tanto, debe dar soporte también requisitos funcionales, de almacenamiento de la información o de calidad.

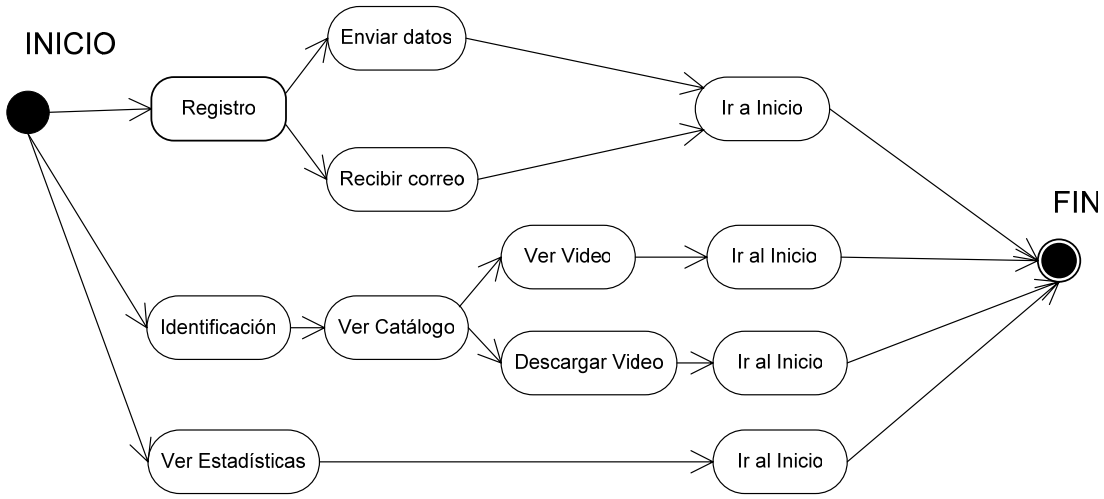
Para representar estas capacidades, se utilizan los Diagramas de Casos de Uso, donde se identifican los diferentes "Actores" y se le asigna a cada uno de ellos un "Caso de Uso". De esta forma, los *Actores* y *Casos de Uso* identificados en esta aplicación son:



Casos de Uso

En el proceso de desarrollo de aplicaciones, el concepto de caso de uso se utiliza para la especificación de requisitos funcionales. Los casos de uso describen un

determinado comportamiento del sistema a partir de las acciones que este lleva a cabo. En el desarrollo de aplicaciones Web este tipo de descripciones necesita ser ampliado con el fin de especificar todos los requisitos. Para ello se ha de obtener la secuencia de acciones que realiza el sistema para alcanzar el objetivo de cada tarea y la interacción que lleva a cabo el usuario y el sistema.



Desarrollo de Casos de Uso

En la figura anterior se observar las diversas acciones que puede realizar un usuario en el momento de acceder a la aplicación.

En el caso del administrador la determinación de las actividades no condiciona la creación de esta aplicación, puesto que Gestionar la seguridad, los usuarios y los videos se hace con anterioridad a la puesta en funcionamiento de la página Web.

A continuación se detalla el Diagrama UML que más información relevante mostrará del desarrollo y posterior funcionamiento de la aplicación. Se trata de un Diagrama UML de Secuencia, en el que se mostrarán todas las páginas que conforman la aplicación y el movimiento del usuario en cada una de ellas:

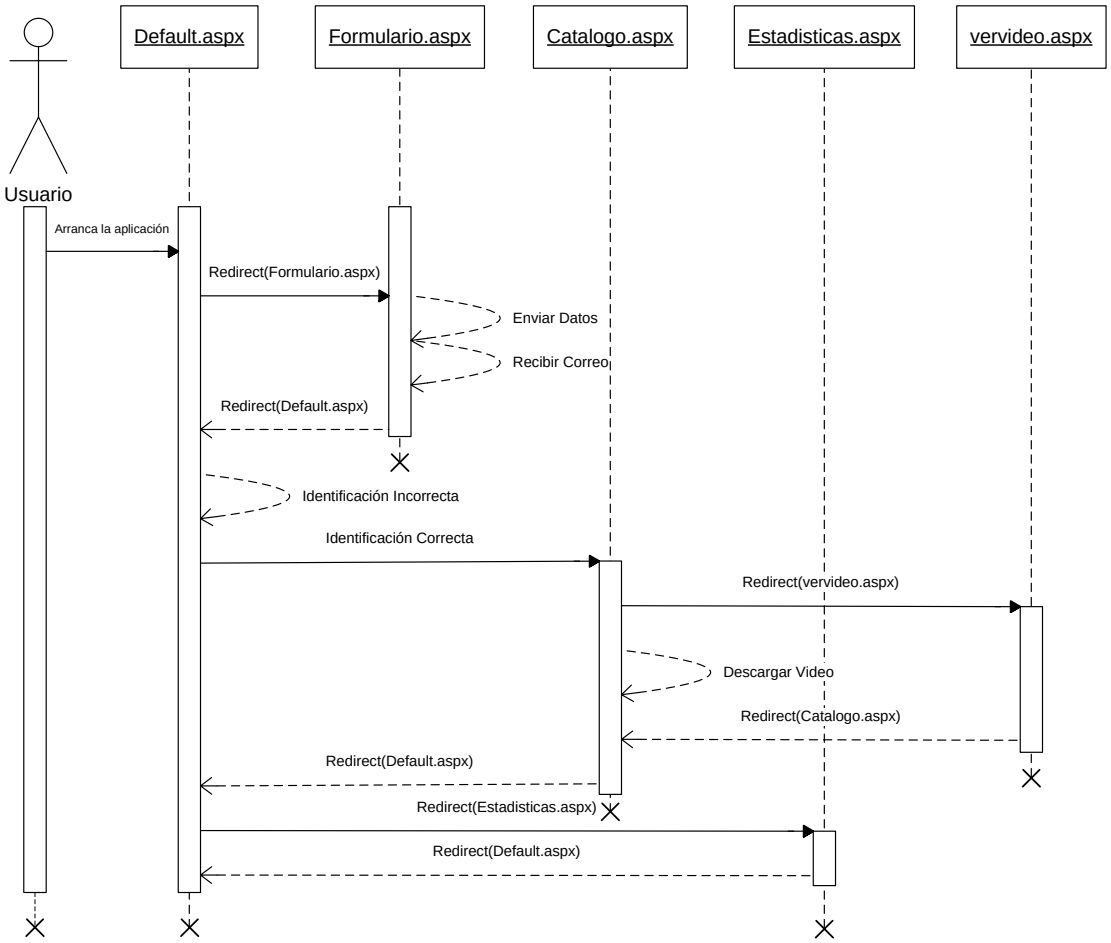


Diagrama UML de Secuencia 1

Es necesario tener en cuenta en este tipo de diagramas, la conexión a la base de datos. Esta conexión no se realiza en cada una de las acciones del usuario, sino que depende de la utilización que haga el usuario de la aplicación, por ejemplo, será necesario conectarse a la base de datos cuando se envíen los datos del usuario, se vean las estadísticas de los vídeos, etc. En el siguiente diagrama no se tendrán en cuenta aquellas páginas en las que no se hace uso de la base de datos:

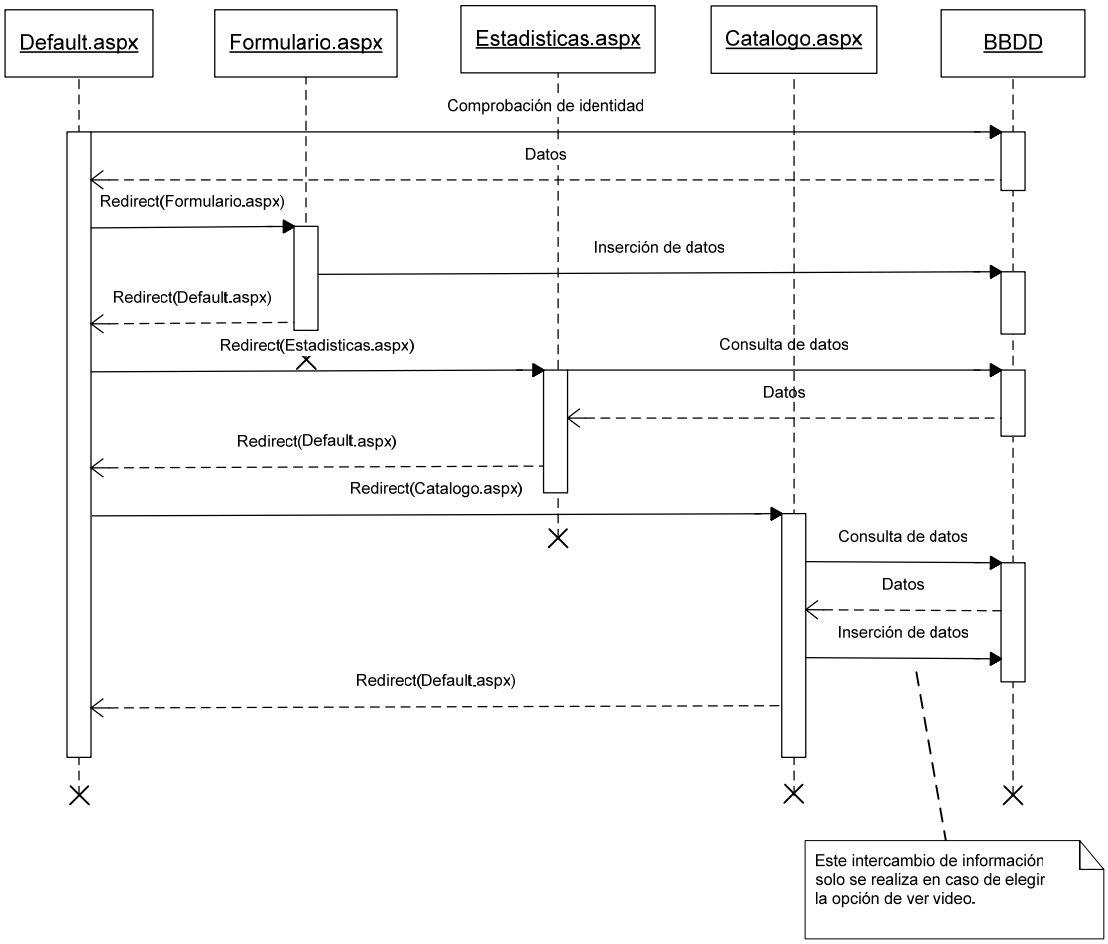


Diagrama UML de Secuencia 2

9. Conclusiones, mejoras y trabajos futuros.

En este proyecto final de carrera se ha realizado un estudio sobre la creación de aplicaciones Web a partir de una tecnología novedosa como es ASP.NET. Cada día son más las aplicaciones Web creadas con esta tecnología.

La tecnología ASP.NET esta cada vez más usada debido a dos ventajas principales: (i) facilidad de desarrollo de componentes .NET proporcionado por la propia arquitectura .NET y su entorno de desarrollo Visual Studio (ii) integración de diferentes lenguajes de programación.

Por otra parte, el gestor de base de datos empleado en el desarrollo de la aplicación, MySQL, ofrece una gran ventaja debido a que hoy en día es la base de datos más empleada en Internet. MySQL es una aplicación *opensource*, concepto muy a tener en cuenta en el desarrollo de cualquier herramienta.

La última aportación ofrecida en el desarrollo de este PFC, es la transmisión de video desde una cámara remota. Referente a ello cabe destacar la gran utilidad que puede tener en numerosas aplicaciones, tales como domótica, seguridad, etc.

Como trabajos futuros se pueden destacar:

- Creación de componentes en ASP.NET que permitan la transmisión de video de una manera más eficaz y sin depender de otras aplicaciones.
- Mejora de la transacción de datos entre aplicación-base de datos y base de datos-aplicación.
- Optimización del código.
- Conexión de cámaras IP al Servidor Web.

Además se pueden añadir más funcionalidades respecto a la aplicación Web como puede ser una mayor interacción del usuario con respecto a la inserción de vídeos por parte de éste o la conexión de varias cámaras remotas.

10. Apéndice.

10.1. Apéndice 1: Manual de Usuario.

En este apartado se va a explicar gráficamente el uso de la aplicación multimedia.

Una forma de representar el conjunto de opciones de la aplicación se muestra a en forma de figura que se irán comentando:

Uno de los casos es en el cuál, el usuario no se identifica ante la aplicación he intenta navegar por ella.

Como se puede observar si accede al enlace de "*Registro"* se le redirige hacía un formulario Web donde deberá introducir sus datos correctamente. Una vez introducidos tiene la opción de recibir un correo electrónico en la dirección introducida por parte del administrador que le enviará su nombre de usuario y contraseña. Para que los datos sean guardados en la base de datos debe pulsar el botón de "*Enviar Datos"*. Por último se le da la opción de volver al inicio.

Si el usuario accede a "*Estadísticas*" puede observar los videos disponibles en la Web además de una pequeña información de cada uno de ellos.

Por último, y es aquí donde radica la diferencia con respecto al usuario identificado, si opta por la opción de visualizar el catálogo, le aparece un página de "*Notificación*" en la que se le informa de que ha de identificarse para acceder al catálogo. En esta "*Notificación*" se le dan dos opciones, o bien volver al inicio o bien ir al *Registro* de usuarios.

Usuario No Identificado

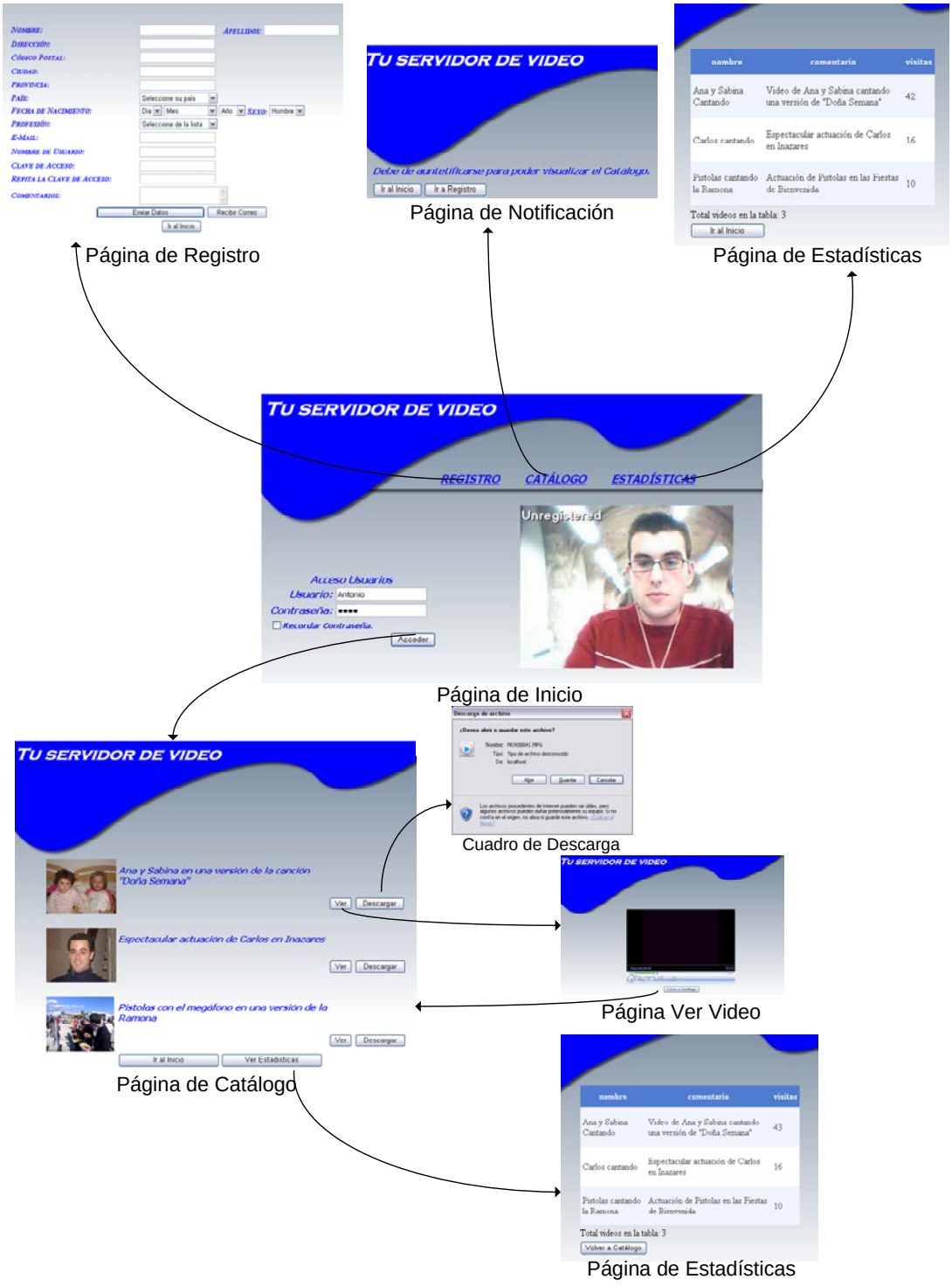
El segundo caso es el del usuario registrado. Tal y como se puede observar el manejo de los enlaces de la parte superior de la página de inicio es la misma que para el usuario no identificado.

La diferencia está en el momento en el que el usuario se identifica. Es en este momento cuando la aplicación redirige al usuario identificado hacía la página de "*Catálogo"*. En esta página el usuario puede *"Ver"* o "*Descargar*" los videos que se le muestran.

Si el usuario opta por visualizar los videos desde la aplicación, se abre una nuevo documento .aspx, "*vervideo.aspx"*, en el que se reproduce automáticamente el video seleccionado.

Por el contrario, si el usuario prefiere descargarse el vídeo, le aparece el cuadro de dialogo para dicha descarga (ver apartado 7.3.4).

Otra opción que tiene el usuario desde el *"Catálogo"* es poder ver las estadísticas de los vídeos al igual que desde la página de "*Inicio*", la diferencia entre ambas está en que en este caso el usuario si puede regresar a "*Catálogo*" opción que no podrá realizar desde el "*Inicio*" por no haberse identificado.



Usuario Identificado

10.2. Apéndice 2: Descripción de Software utilizado.

10.2.1. Visual Studio 2005

El entorno de desarrollo para ASP.NET y C# ha sido Visual Studio 2005 (ver apartado 3.2.3.3). Este entorno permita la creación de aplicaciones con diferentes lenguajes, es decir, en el caso de esta aplicación, el entorno y diseño Web ha sido programada en ASP.NET mientras que la funcionalidad de muchos de los componentes empleados han sido programados en C# (ver apartado 3.3).

Este entorno de desarrollo ha sido creado por Microsoft y se encuentra un programa de prueba que se puede descargar en http://www.microsoft.com/spanish/msdn/spain/beta2vs05/default.asp y es de fácil instalación.

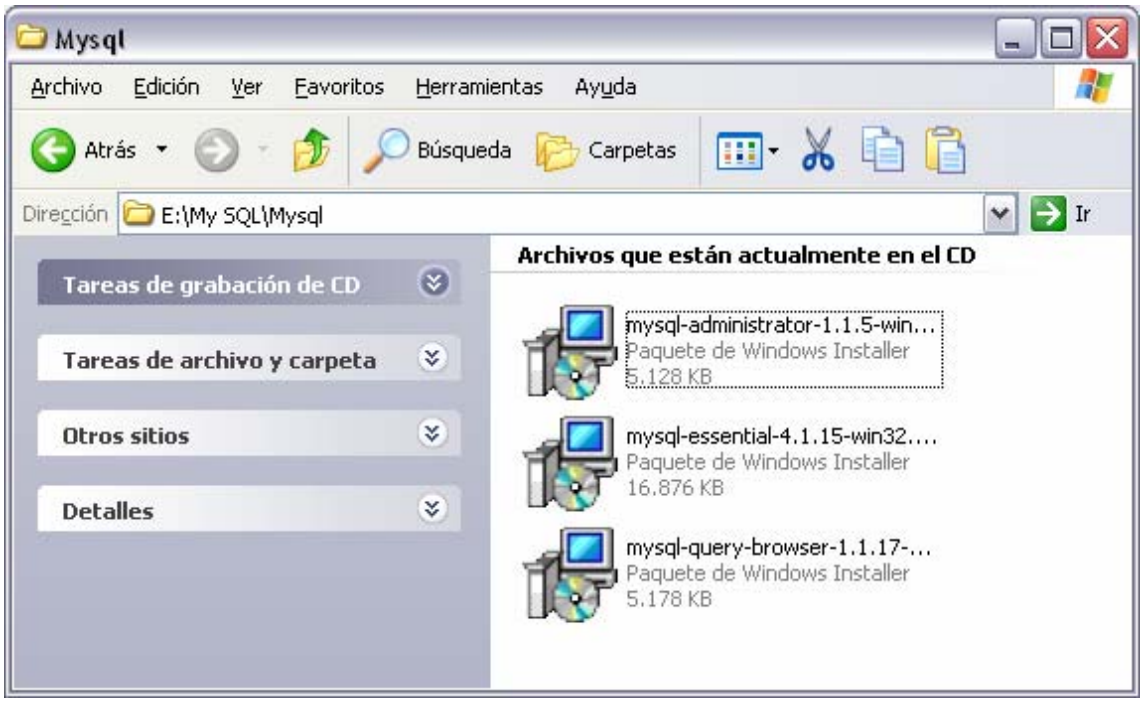
La realización de aplicaciones bajo este entorno se realiza de forma sencilla gracias un entorno de desarrollo muy intuitivo y con gran disponibilidad de diversos componentes.

10.2.2. MySQL.

El gestor de base de datos elegido para la creación de la base de datos de la aplicación ha sido MySQL (ver apartado 3.6). Se trata de una aplicación *opensource* y la versión empleada ha sido MySQL 4.1.15-nt via TCP/IP.

10.2.2.1. Instalación de MySQL.

La aplicación MySQL empleada consta de los siguientes asistentes de instalación:



Asistentes de Instalación

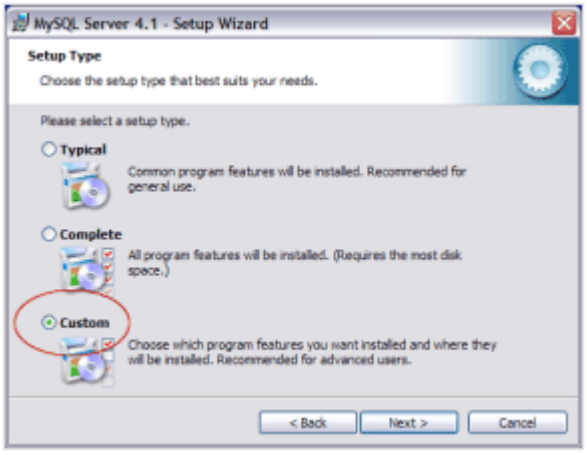
Tal y como se puede observar, están *mysql-administrator*, *mysql-essetial* y *mysql-query-browser*.

El primero que se ha de instalar será el *mysql-essential*:

Se inicia el asistente de instalación, se ha de pulsar *Next:*

Inicio del asistente de instalación

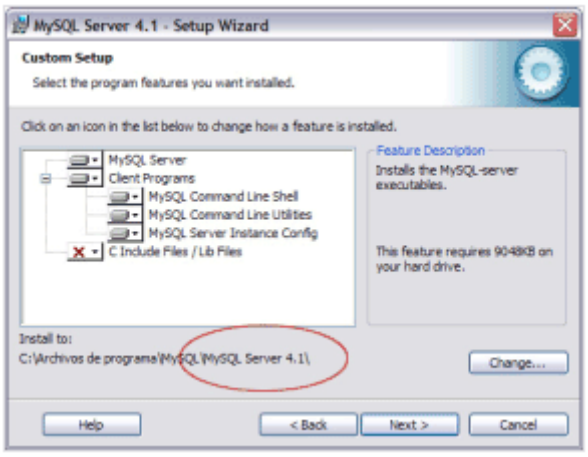
A continuación se debe seleccionar el tipo de instalación (*custom*):



Selección del Tipo de Instalación

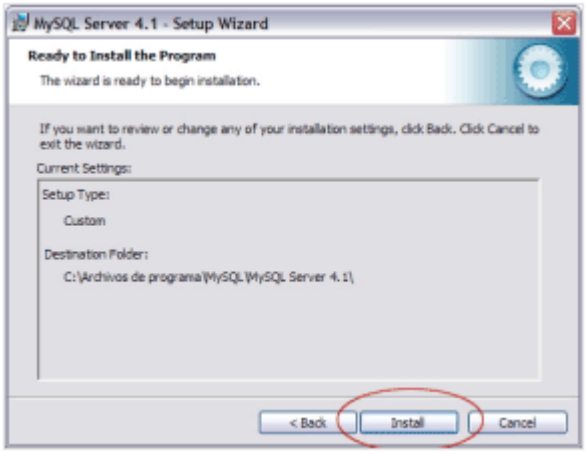
Lo que será instalado es: el servidor MySQL, el cliente modo texto para MySQL, algunos otros programas que se ejecutan de la línea de comandos y el asistente de configuración.

A continuación se ha de indicar el directorio de instalación:



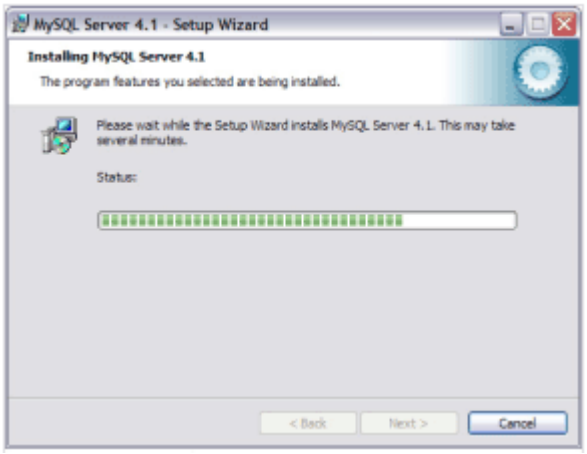
Directorio de Instalación

Seguidamente se comienza con la instalación:



Inicio de la Instalación

Se inicia la copia de archivos al sistema:

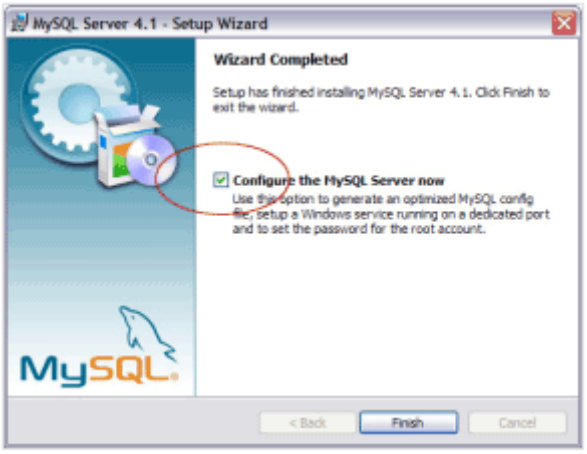


Copia de archivos al sistema

Se pide dar de alta una cuenta en el sitio MySQL.com. Este proceso se puede omitir seleccionando la opción de omitir registro (*Skip Sign-up*):

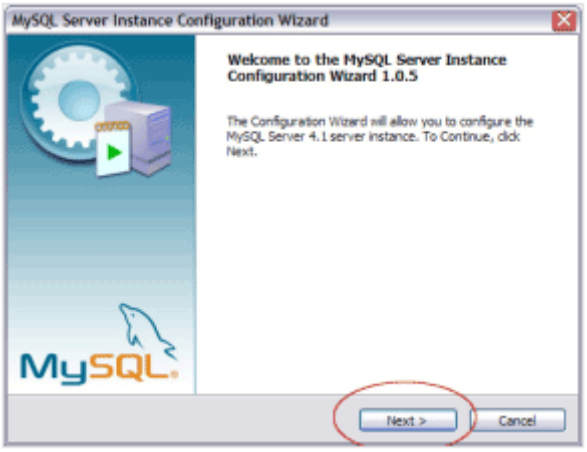
Registro de cuenta

Una vez que la instalación ha finalizado, hay una opción de ejecutar el asistente de instalación de MySQL. Se selecciona esta opción y se presiona el botón Finalizar (*Finish*):



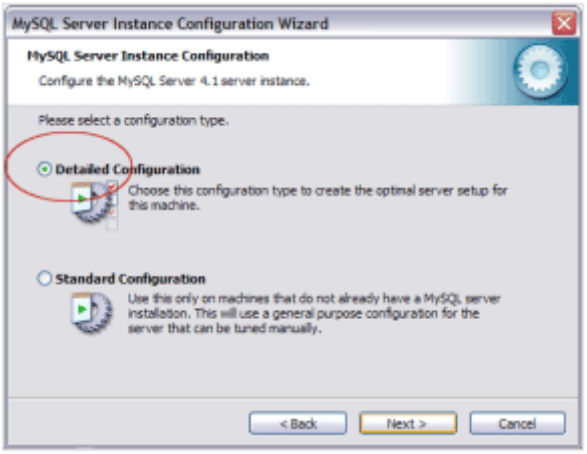
Fin de Instalación

Se muestra la pantalla de bienvenida al asistente de configuración:



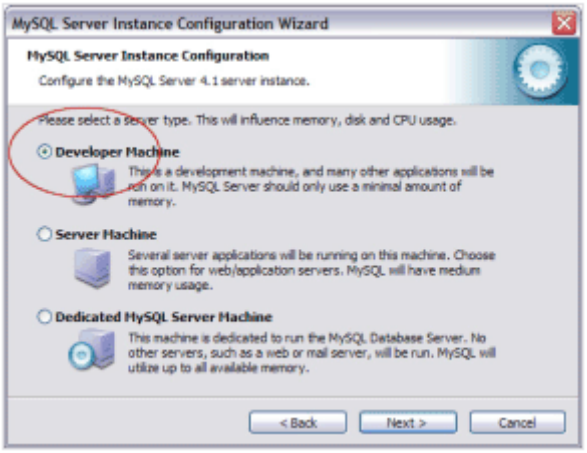
Asistente de Configuración

Se selecciona el tipo de configuración detallada:



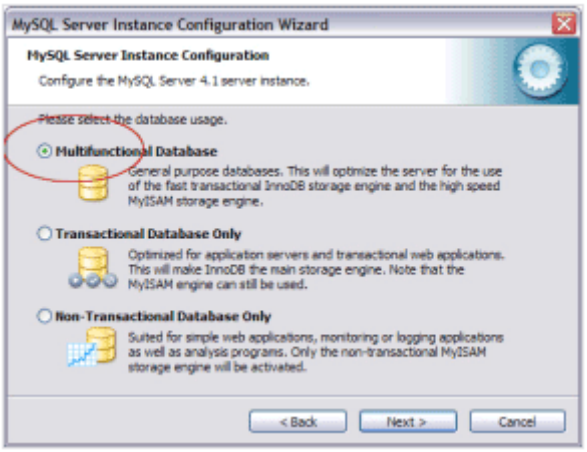
Configuración Detallada

Se selecciona el servidor de tipo desarrollo (*Developer machine)*, de esta forma la configuración de MySQL quedará de tal manera que ocupe la menor cantidad de recursos:



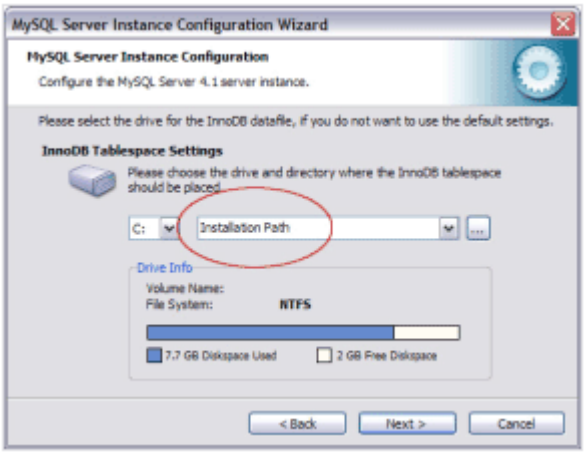
Tipo de Servidor

A continuación se muestra la opción de base de datos que se desea utilizar, en este caso base de datos multifuncional (*Muntifuctional Database*):



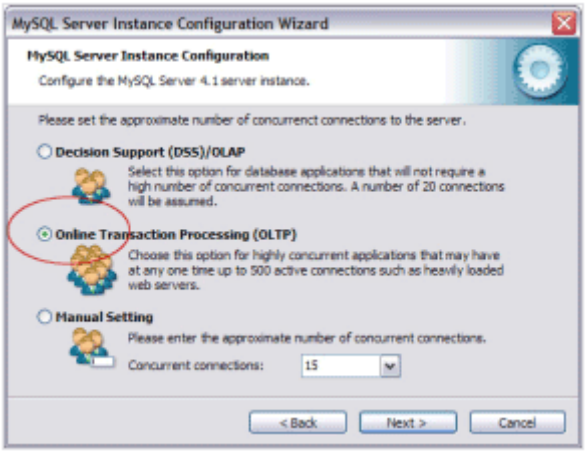
Tipo de Base de Datos

Se puede determinar cuál será el directorio donde se ubicarán los archivos referentes a las tablas creadas, sin embargo, es recomendable dejar éstos en el mismo directorio de instalación de MySQL (*Installation Path*):



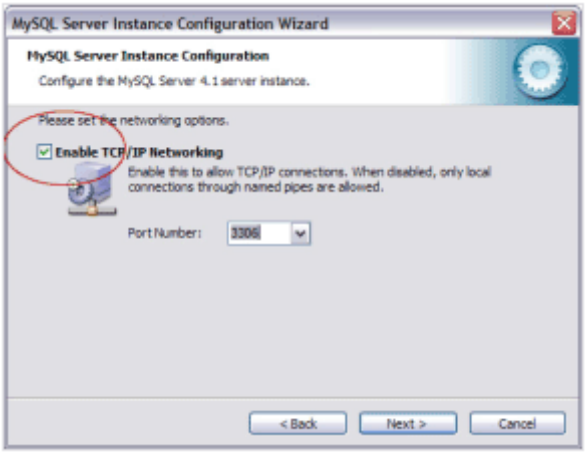
Directorio de Almacenamiento

En el siguiente paso se deberá especificar cuál será el máximo número conexiones concurrentes que aceptará el servidor. Con la segunda opción habrá habilitadas por lo menos 100 conexiones concurrentes, suficiente para muchas situaciones:



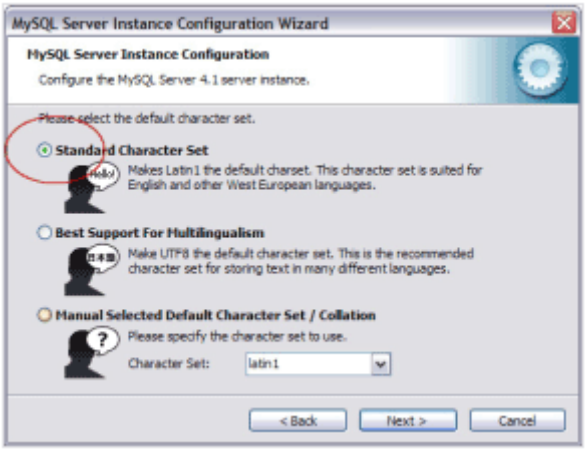
Número de Conexiones Concurrentes

Si se desea conectar de manera remota al servidor que se está instalando, hay que dejar habilitada la opción de red TCP/IP (*TCP/IP Networking*):



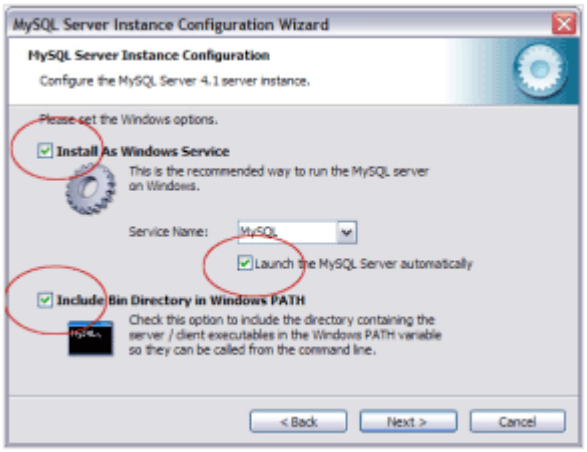
Conexión Remota al Servidor

En este paso se determina cuál será el conjunto de caracteres por omisión. *Latin1* debe ser la primera opción ha considerar, puesto que, no se tendrán problemas para manejar palabras acentuadas y algunos otros caracteres especiales:



Conjunto de Caracteres

Es recomendable que se deje habilitada la opción de instalación como servicio de Windows, y que además el servidor sea iniciado automáticamente. La otra opción presente es la de habilitar que todos los programas todos los programas cliente que se distribuyen con MySQL estén disponibles desde la línea de comandos:

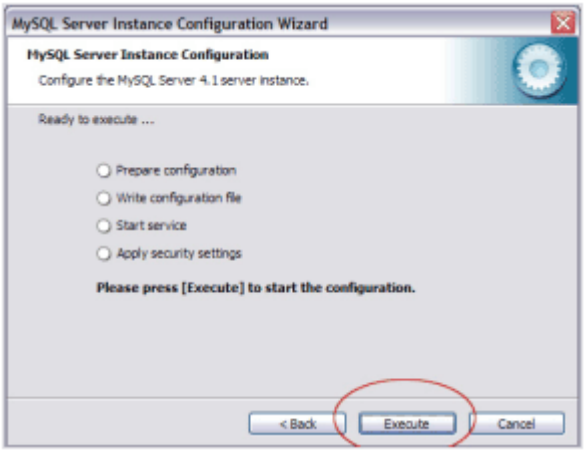


Servicio de Windows

Ahora se debe determinar cuál será la contraseña del usuario administrador de la base de datos de MySQL:

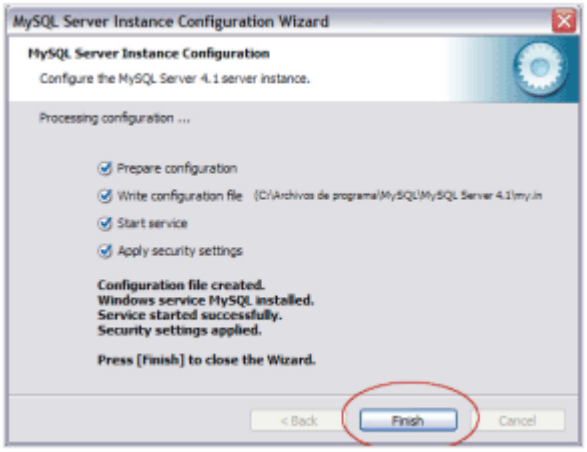
Contraseña del Administrador

Para terminar se presiona el botón Ejecutar (*Execute*):



Ejecutar

Y por último terminamos con el botón finalizar (*Finish*):



Finalizar

Al finalizar el asistente, el archivo de configuración de MySQL (*my.ini*) será generado y almacenado en el directorio de instalación.

El asistente instalará MySQL como servicio de Windows e intentará iniciarlo por primera vez. Si el inicio del servidor llegara a fallar se pueden revisar los archivos de registros de errores localizados en el directorio de instalación.

10.2.2.2. Creación de una base de datos.

Una vez instalado MySQL se debe de crear la base de datos deseada. Para ver la creación de una base de datos ver apartado 7.3.1.

10.2.3. Instalación de IIS.

Para poner en funcionamiento cualquier aplicación ASP.NET es necesario la instalación de IIS (*Internet Information Server*) y la creación de un directorio virtual en el ordenador Servidor. Todo ello se encuentra explicado en el apartado 5.

10.3. Apéndice 3: Código de la aplicación

A continuación se expone el código empleado en la implementación de la aplicación.

10.3.1. Página Default.

10.3.1. Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>:::Tu Servidor Web:::</title>
<script language="javascript" type="text/javascript">
</script>
</head>
<body style="background-position: center; background-image: url(fondo
principal2.png); background-repeat: no-repeat">
    <form id="form1" runat="server">
    <div style="left: 100px; top: 0px; position: relative;"
      title=":::Tu Servidor de Video:::">
        <br />
        <br />
        <br />
                         
                 
                         
                   
                         
                
                     
           
                         
                 
                        
           
                         
                 
                         
                 
                        
           
                         
                 
                         
                 
                         
                <br />
          
        <table id="TABLE1" style="width: 805px" onclick="return
TABLE1_onclick()">
```

```
        <tr>
            <td style="width: 234px; height: 24px;">
            </td>
            <td style="height: 24px;" colspan="3">
  <asp:HyperLink ID="HyperLink1" runat="server"
BackColor="Transparent" ForeColor="Blue"
EnableTheming="True" Font-Bold="True" Font-Italic="True"
Font-Size="Larger" Font-Underline="True"
NavigateUrl="~/Formulario.aspx" Font-
Names="Tahoma">REGISTRO</asp:HyperLink>
                        
  <asp:HyperLink ID="HyperLink2" runat="server" Font-Bold="True"
      Font-Italic="True" Font-Size="Larger" Font-Underline="True"
      ForeColor="Blue" Font-Names="Tahoma"
NavigateUrl="~/Notificacion.aspx">CATÁLOGO</asp:HyperLink>
                        
  <asp:HyperLink ID="HyperLink3" runat="server" Font-Bold="True"
      Font-Italic="True" Font-Size="Larger" Font-Underline="True"
      ForeColor="Blue" Font-Names="Tahoma" Width="8px"
NavigateUrl="~/Estadisticas.aspx">ESTADÍSTICAS
</asp:HyperLink> </td>
        </tr>
        <tr>
            <td style="width: 234px; height: 21px;">
            </td>
            <td style="width: 31px; height: 21px">
            </td>
            <td style="width: 20px; height: 21px;">
            </td>
            <td style="width: 86px; height: 21px;">
            </td>
        </tr>
        <tr>
            <td style="width: 234px; height: 257px;">
                <br />
                <br />
                <br />
                <br />
  <asp:Login ID="Login1" runat="server" Font-Bold="True" Font-
Italic="True" Font-Names="Tahoma"
      ForeColor="Blue" LoginButtonText="Acceder"
OnAuthenticate="Login1_Authenticate"
      PasswordLabelText="Contraseña:"
PasswordRequiredErrorMessage="Debe Introducir su
Contraseña."
      RememberMeText="Recordar Contraseña." TitleText="Acceso
Usuarios" UserNameLabelText="Usuario:"
      UserNameRequiredErrorMessage="Debe Intruducir su Usuario."
      FailureText="">
      <TitleTextStyle Font-Bold="True" Font-Italic="True" Font-
Names="Tahoma" ForeColor="Blue" />
      <CheckBoxStyle Font-Bold="True" Font-Italic="True" Font-
Names="Tahoma" Font-Size="Smaller"
      ForeColor="Blue" />
      <TextBoxStyle Font-Bold="False" Font-Italic="False" Font-
Names="Tahoma" ForeColor="Black" />
      <FailureTextStyle Wrap="True" />
  </asp:Login>
  <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-
Italic="True" Font-Names="Tahoma"
ForeColor="Blue"></asp:Label></td>
```

```
                <td colspan="1" style="width: 31px; height: 257px">
                </td>
                <td style="height: 257px; width: 152px;" colspan="3">


<img name="webcam" alt="" src="capture1.jpg"/>
 <script type="text/javascript" language="JavaScript"><!--
  periodo = 0.5;
  imgsrc = document.webcam.src;
  buffer = new Image();
  buffer.onload = imageChange;
  buffer.onerror = imageError;
  setTimeout("imageReload()", periodo * 1000);

  function imageReload() {
    buffer.src = imgsrc + "?d=" + new Date().getTime();
      setTimeout("imageReload()", periodo * 1000);
  }

  function imageChange() {
    document.webcam.src = buffer.src;
  }

  function imageError() {
      setTimeout("imageReload()", 1000);
  }
  // --></script>

                </td>
            </tr>
        </table>
        <br />
                     
                         
                 
                         
                 
                       
                     
                        
           
                         
                 
                         
                <br />

        <br />
        <br />
    </div>
    </form>
</body>
</html>
```

10.3.2. Default.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.Odbc;
using System.Net.Sockets;
using MSR.LST.Net.Rtp;
using System.Windows.Forms;



public partial class _Default : System.Web.UI.Page
{

    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void Login1_Authenticate(object sender,
      AuthenticateEventArgs e)
    {
        //Con la ayuda de Configuration Manager recogemos la conexión
      de Web.Config
        OdbcConnection conexion = new
      OdbcConnection(ConfigurationManager.
      ConnectionStrings["BaseDatos"].ConnectionString);

        string cadena = "SELECT count(*) FROM usuario WHERE usuario
      ='" + Login1.UserName + "' and clave='" +
      Login1.Password+"'";
        string cadenaNombre = "SELECT count(*) FROM usuario WHERE
      usuario='" + Login1.UserName + "'";
        string cadenaClave = "SELECT count(*) FROM usuario WHERE
      clave='" + Login1.Password + "'";

        //Declaramos el comando SQL
        OdbcCommand comando = new OdbcCommand();

        //Abrimos la conexión
        conexion.Open();

        //Asignar la conexión al comando
        comando.Connection = conexion;

        //Asignamos al comando SQL la sentencia SQL que ejecutará
        comando.CommandText = cadenaNombre;

        //Preparamos la consulta para que nos indique si hay o no
      coincidencia
        int contador=0;
        bool usuario = true;
        bool clave = true;
```

```
        contador = Convert.ToInt32(comando.ExecuteScalar());
        if (contador == 0)
        {
                usuario = false;
                Label1.Text= "No está registrado como usuario en este
            servicio";
        }
        else
        {
            comando.CommandText = cadenaClave;
            contador = Convert.ToInt32(comando.ExecuteScalar());
            if (contador == 0)
            {
                clave = false;
                Label1.Text="La Contraseña Introducida es Incorrecta";
            }
        }

        comando.CommandText = cadena;
        contador = Convert.ToInt32(comando.ExecuteScalar());
        if (contador == 1)
        {
            //Damos acceso al usuario

            FormsAuthentication.RedirectFromLoginPage(Login1.UserName,
                  false);


        }


    }

}
```

10.3.2. Página Formulario.

10.3.2.1. Formulario.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Formulario.aspx.cs" Inherits="Formulario" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body style="background-image: url(fondo básico.png); background-
position-x: center; background-repeat: no-repeat; background-
attachment: scroll;">
    <form id="form1" runat="server">
    <div>
        <br />
        <br />
```

```
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
    <table style="font-weight: bold; color: blue; font-style:
italic; font-variant: small-caps; left: 126px; position:
relative; top: 15px;">
        <tr>
            <td style="width: 260px; height: 26px">
                <asp:Label ID="LabelUsuario"
              runat="server"></asp:Label></td>
            <td style="width: 436px; height: 26px">
            </td>
        </tr>
        <tr>
            <td style="width: 260px; height: 26px">
                <asp:Label ID="LabelCorreo"
              runat="server"></asp:Label></td>
            <td style="width: 436px; height: 26px">
            </td>
        </tr>
        <tr>
            <td style="width: 260px; height: 26px">
                <asp:Label ID="LabelInformacion"
              runat="server"></asp:Label></td>
            <td style="width: 436px; height: 26px">
            </td>
        </tr>
        <tr>
            <td style="width: 260px; height: 26px;">
                <asp:Label ID="Label1" runat="server"
              Text="Nombre:"></asp:Label></td>
            <td style="width: 436px; height: 26px;">
                <asp:TextBox ID="TextBoxNombre" runat="server"
              Width="150px"></asp:TextBox> 
     <asp:RequiredFieldValidator
                    ID="RequiredFieldValidator1" runat="server"
              ControlToValidate="TextBoxNombre"
                    ErrorMessage="*"></asp:RequiredFieldValidator>
                <asp:Label ID="Label12" runat="server"
                    Text="Apellidos:"></asp:Label>
                <asp:TextBox ID="TextBoxApellidos" runat="server"
              Width="150px"></asp:TextBox>
                <asp:RequiredFieldValidator
              ID="RequiredFieldValidator2" runat="server"
         ControlToValidate="TextBoxApellidos"
                    ErrorMessage="*">
              </asp:RequiredFieldValidator></td>
        </tr>
        <tr>
            <td style="width: 260px; height: 21px">
                <asp:Label ID="Label2" runat="server"
              Text="Dirección:"></asp:Label></td>
            <td style="width: 436px; height: 21px">
                <asp:TextBox ID="TextBoxDireccion" runat="server"
              Width="149px"></asp:TextBox>
```

```
            <asp:RequiredFieldValidator
          ID="RequiredFieldValidator3" runat="server"
ControlToValidate="TextBoxDireccion"
              ErrorMessage="*">
      </asp:RequiredFieldValidator></td>
</tr>
<tr>
    <td style="width: 260px">
        <asp:Label ID="Label3" runat="server" Text="Código
            Postal:"></asp:Label></td>
    <td style="width: 436px">
        <asp:TextBox ID="TextBoxCPostal" runat="server"
      Width="150px"></asp:TextBox>
          <asp:RequiredFieldValidator
      ID="RequiredFieldValidator4" runat="server"
ControlToValidate="TextBoxCPostal"
              ErrorMessage="*">
   </asp:RequiredFieldValidator></td>
</tr>
<tr>
    <td style="width: 260px">
        <asp:Label ID="Label4" runat="server"
      Text="Ciudad:"></asp:Label></td>
    <td style="width: 436px">
        <asp:TextBox ID="TextBoxCiudad" runat="server"
      Width="150px"></asp:TextBox>
          <asp:RequiredFieldValidator
      ID="RequiredFieldValidator5" runat="server"
ControlToValidate="TextBoxCiudad"
              ErrorMessage="*">
  </asp:RequiredFieldValidator></td>
</tr>
<tr>
    <td style="width: 260px">
        <asp:Label ID="Label5" runat="server"
      Text="Provincia:"></asp:Label></td>
    <td style="width: 436px">
        <asp:TextBox ID="TextBoxProvincia" runat="server"
      Width="150px"></asp:TextBox>
          <asp:RequiredFieldValidator
      ID="RequiredFieldValidator6" runat="server"
ControlToValidate="TextBoxProvincia"
              ErrorMessage="*">
   </asp:RequiredFieldValidator></td>
</tr>
<tr>
    <td style="width: 260px">
        <asp:Label ID="Label13" runat="server"
            Text="País:"></asp:Label></td>
    <td style="width: 436px">
        <asp:DropDownList ID="DropDownList1"
      runat="server" Width="160px">
            <asp:ListItem>Seleccione su
            pa&#237;s</asp:ListItem>
            <asp:ListItem>Afganist&#225;n</asp:ListItem>
            <asp:ListItem>Albania</asp:ListItem>
            <asp:ListItem>Andorra</asp:ListItem>
            <asp:ListItem>Angola</asp:ListItem>
            <asp:ListItem>Alemania</asp:ListItem>
            <asp:ListItem>Arab&#237;a
            Saud&#237;</asp:ListItem>
```

```
<asp:ListItem>Argelia</asp:ListItem>
<asp:ListItem>Argentina</asp:ListItem>
<asp:ListItem>Australia</asp:ListItem>
<asp:ListItem>Austria</asp:ListItem>
<asp:ListItem>B&#233;lgica</asp:ListItem>
<asp:ListItem>Bielorrusia</asp:ListItem>
<asp:ListItem>Bolivia</asp:ListItem>
<asp:ListItem>Brasil</asp:ListItem>
<asp:ListItem>Camer&#250;n</asp:ListItem>
<asp:ListItem>Canad&#225;</asp:ListItem>
<asp:ListItem>Chile</asp:ListItem>
<asp:ListItem>China</asp:ListItem>
<asp:ListItem>Chipre</asp:ListItem>
<asp:ListItem>Colombia</asp:ListItem>
<asp:ListItem>Corea</asp:ListItem>
<asp:ListItem>Corea del Norte</asp:ListItem>
<asp:ListItem>Costa de Marfil</asp:ListItem>
<asp:ListItem>Costa Rica</asp:ListItem>
<asp:ListItem>Croacia</asp:ListItem>
<asp:ListItem>Cuba</asp:ListItem>
<asp:ListItem>Dinamarca</asp:ListItem>
<asp:ListItem>Ecuador</asp:ListItem>
<asp:ListItem>Egipto</asp:ListItem>
<asp:ListItem>Eslovaquia</asp:ListItem>
<asp:ListItem>Eslovenia</asp:ListItem>
<asp:ListItem>Espa&#241;a</asp:ListItem>
<asp:ListItem>Estados Unidos</asp:ListItem>
<asp:ListItem>Finlandia</asp:ListItem>
<asp:ListItem>Francia</asp:ListItem>
<asp:ListItem>Ghana</asp:ListItem>
<asp:ListItem>Grecia</asp:ListItem>
<asp:ListItem>Ghinea</asp:ListItem>
<asp:ListItem>Holanda</asp:ListItem>
<asp:ListItem>Honduras</asp:ListItem>
<asp:ListItem>Hungr&#237;a</asp:ListItem>
<asp:ListItem>India</asp:ListItem>
<asp:ListItem>Indonesia</asp:ListItem>
<asp:ListItem>Irak</asp:ListItem>
<asp:ListItem>Ir&#225;n</asp:ListItem>
<asp:ListItem>Irlanda</asp:ListItem>
<asp:ListItem>Israel</asp:ListItem>
<asp:ListItem>Italia</asp:ListItem>
<asp:ListItem>Jamaica</asp:ListItem>
<asp:ListItem>Jap&#243;n</asp:ListItem>
<asp:ListItem>Kenia</asp:ListItem>
<asp:ListItem>Kuwait</asp:ListItem>
<asp:ListItem>Laos</asp:ListItem>
<asp:ListItem>Letonia</asp:ListItem>
<asp:ListItem>L&#237;bano</asp:ListItem>
<asp:ListItem>Liechtenstein</asp:ListItem>
<asp:ListItem>Lituania</asp:ListItem>
<asp:ListItem>Luxemburgo</asp:ListItem>
<asp:ListItem>Malasia</asp:ListItem>
<asp:ListItem>Mali</asp:ListItem>
<asp:ListItem>Malta</asp:ListItem>
<asp:ListItem>Marruecos</asp:ListItem>
<asp:ListItem>Mauritania</asp:ListItem>
<asp:ListItem>M&#233;xico</asp:ListItem>
<asp:ListItem>Moldavia</asp:ListItem>
<asp:ListItem>M&#243;naco</asp:ListItem>
<asp:ListItem>Namibia</asp:ListItem>
```

```
                <asp:ListItem>Nicaragua</asp:ListItem>
                <asp:ListItem>Nigeria</asp:ListItem>
                <asp:ListItem>Noruega</asp:ListItem>
                <asp:ListItem>Nueva Zelanda</asp:ListItem>
                <asp:ListItem>Om&#225;n</asp:ListItem>
                <asp:ListItem>Paquist&#225;n</asp:ListItem>
                <asp:ListItem>Paraguay</asp:ListItem>
                <asp:ListItem>Per&#250;</asp:ListItem>
                <asp:ListItem>Polonia</asp:ListItem>
                <asp:ListItem>Portugal</asp:ListItem>
                <asp:ListItem>Puerto Rico</asp:ListItem>
                <asp:ListItem>Qatar</asp:ListItem>
                <asp:ListItem>Reino Unido</asp:ListItem>
                <asp:ListItem>Rep&#250;blica
               Checa</asp:ListItem>
                <asp:ListItem>Republica
               Dominicana</asp:ListItem>
                <asp:ListItem>Ruanda</asp:ListItem>
                <asp:ListItem>Rumania</asp:ListItem>
                <asp:ListItem>Rusia</asp:ListItem>
                <asp:ListItem>San Marino</asp:ListItem>
                <asp:ListItem>Senegal</asp:ListItem>
                <asp:ListItem>Serbia y
               Montenegro</asp:ListItem>
                <asp:ListItem>Sierra Leona</asp:ListItem>
                <asp:ListItem>Singapur</asp:ListItem>
                <asp:ListItem>Suecia</asp:ListItem>
                <asp:ListItem>Suiza</asp:ListItem>
                <asp:ListItem>Tailandia</asp:ListItem>
                <asp:ListItem>Tanzania</asp:ListItem>
                <asp:ListItem>Turqu&#237;a</asp:ListItem>
                <asp:ListItem>Ucrania</asp:ListItem>
                <asp:ListItem>Uganda</asp:ListItem>
                <asp:ListItem>Urugay</asp:ListItem>
                <asp:ListItem>Venezuela</asp:ListItem>
            </asp:DropDownList></td>
    </tr>
    <tr>
        <td style="width: 260px">
            <asp:Label ID="Label6" runat="server" Text="Fecha
          de Nacimiento:"></asp:Label></td>
        <td style="width: 436px">
            <asp:DropDownList ID="DropDownList2"
          runat="server">
                <asp:ListItem>Dia</asp:ListItem>
                <asp:ListItem>1</asp:ListItem>
                <asp:ListItem>2</asp:ListItem>
                <asp:ListItem>3</asp:ListItem>
                <asp:ListItem>4</asp:ListItem>
                <asp:ListItem>5</asp:ListItem>
                <asp:ListItem>6</asp:ListItem>
                <asp:ListItem>7</asp:ListItem>
                <asp:ListItem>8</asp:ListItem>
                <asp:ListItem>9</asp:ListItem>
                <asp:ListItem>10</asp:ListItem>
                <asp:ListItem>11</asp:ListItem>
                <asp:ListItem>12</asp:ListItem>
                <asp:ListItem>13</asp:ListItem>
                <asp:ListItem>14</asp:ListItem>
                <asp:ListItem>15</asp:ListItem>
                <asp:ListItem>16</asp:ListItem>
```

```
 <asp:ListItem>17</asp:ListItem>
 <asp:ListItem>18</asp:ListItem>
 <asp:ListItem>19</asp:ListItem>
 <asp:ListItem>20</asp:ListItem>
 <asp:ListItem>21</asp:ListItem>
 <asp:ListItem>22</asp:ListItem>
 <asp:ListItem>23</asp:ListItem>
 <asp:ListItem>24</asp:ListItem>
 <asp:ListItem>25</asp:ListItem>
 <asp:ListItem>26</asp:ListItem>
 <asp:ListItem>27</asp:ListItem>
 <asp:ListItem>28</asp:ListItem>
 <asp:ListItem>29</asp:ListItem>
 <asp:ListItem>30</asp:ListItem>
 <asp:ListItem>31</asp:ListItem>
</asp:DropDownList>
<asp:DropDownList ID="DropDownList3"
runat="server" Width="109px">
 <asp:ListItem>Mes</asp:ListItem>
 <asp:ListItem>Enero</asp:ListItem>
 <asp:ListItem>Febrero</asp:ListItem>
 <asp:ListItem>Marzo</asp:ListItem>
 <asp:ListItem>Abril</asp:ListItem>
 <asp:ListItem>Mayo</asp:ListItem>
 <asp:ListItem>Junio</asp:ListItem>
 <asp:ListItem>Julio</asp:ListItem>
 <asp:ListItem>Agosto</asp:ListItem>
 <asp:ListItem>Septiembre</asp:ListItem>
 <asp:ListItem>Octubre</asp:ListItem>
 <asp:ListItem>Noviembre</asp:ListItem>
 <asp:ListItem>Diciembre</asp:ListItem>
</asp:DropDownList>
<asp:DropDownList ID="DropDownList4"
runat="server">
 <asp:ListItem>A&#241;o</asp:ListItem>
 <asp:ListItem>2006</asp:ListItem>
 <asp:ListItem>2005</asp:ListItem>
 <asp:ListItem>2004</asp:ListItem>
 <asp:ListItem>2003</asp:ListItem>
 <asp:ListItem>2002</asp:ListItem>
 <asp:ListItem>2001</asp:ListItem>
 <asp:ListItem>2000</asp:ListItem>
 <asp:ListItem>1999</asp:ListItem>
 <asp:ListItem>1998</asp:ListItem>
 <asp:ListItem>1997</asp:ListItem>
 <asp:ListItem>1996</asp:ListItem>
 <asp:ListItem>1995</asp:ListItem>
 <asp:ListItem>1994</asp:ListItem>
 <asp:ListItem>1993</asp:ListItem>
 <asp:ListItem>1992</asp:ListItem>
 <asp:ListItem>1991</asp:ListItem>
 <asp:ListItem>1990</asp:ListItem>
 <asp:ListItem>1989</asp:ListItem>
 <asp:ListItem>1988</asp:ListItem>
 <asp:ListItem>1987</asp:ListItem>
 <asp:ListItem>1986</asp:ListItem>
 <asp:ListItem>1985</asp:ListItem>
 <asp:ListItem>1984</asp:ListItem>
 <asp:ListItem>1983</asp:ListItem>
 <asp:ListItem>1982</asp:ListItem>
 <asp:ListItem>1981</asp:ListItem>
```

```
<asp:ListItem>1980</asp:ListItem>
<asp:ListItem>1979</asp:ListItem>
<asp:ListItem>1978</asp:ListItem>
<asp:ListItem>1977</asp:ListItem>
<asp:ListItem>1976</asp:ListItem>
<asp:ListItem>1975</asp:ListItem>
<asp:ListItem>1974</asp:ListItem>
<asp:ListItem>1973</asp:ListItem>
<asp:ListItem>1972</asp:ListItem>
<asp:ListItem>1971</asp:ListItem>
<asp:ListItem>1970</asp:ListItem>
<asp:ListItem>1969</asp:ListItem>
<asp:ListItem>1968</asp:ListItem>
<asp:ListItem>1967</asp:ListItem>
<asp:ListItem>1966</asp:ListItem>
<asp:ListItem>1965</asp:ListItem>
<asp:ListItem>1964</asp:ListItem>
<asp:ListItem>1963</asp:ListItem>
<asp:ListItem>1962</asp:ListItem>
<asp:ListItem>1961</asp:ListItem>
<asp:ListItem>1960</asp:ListItem>
<asp:ListItem>1959</asp:ListItem>
<asp:ListItem>1958</asp:ListItem>
<asp:ListItem>1957</asp:ListItem>
<asp:ListItem>1956</asp:ListItem>
<asp:ListItem>1955</asp:ListItem>
<asp:ListItem>1954</asp:ListItem>
<asp:ListItem>1953</asp:ListItem>
<asp:ListItem>1952</asp:ListItem>
<asp:ListItem>1951</asp:ListItem>
<asp:ListItem>1950</asp:ListItem>
<asp:ListItem>1949</asp:ListItem>
<asp:ListItem>1948</asp:ListItem>
<asp:ListItem>1947</asp:ListItem>
<asp:ListItem>1946</asp:ListItem>
<asp:ListItem>1945</asp:ListItem>
<asp:ListItem>1944</asp:ListItem>
<asp:ListItem>1943</asp:ListItem>
<asp:ListItem>1942</asp:ListItem>
<asp:ListItem>1941</asp:ListItem>
<asp:ListItem>1940</asp:ListItem>
<asp:ListItem>1939</asp:ListItem>
<asp:ListItem>1938</asp:ListItem>
<asp:ListItem>1937</asp:ListItem>
<asp:ListItem>1936</asp:ListItem>
<asp:ListItem>1935</asp:ListItem>
<asp:ListItem>1934</asp:ListItem>
<asp:ListItem>1933</asp:ListItem>
<asp:ListItem>1932</asp:ListItem>
<asp:ListItem>1931</asp:ListItem>
<asp:ListItem>1930</asp:ListItem>
<asp:ListItem>1929</asp:ListItem>
<asp:ListItem>1928</asp:ListItem>
<asp:ListItem>1927</asp:ListItem>
<asp:ListItem>1926</asp:ListItem>
<asp:ListItem>1925</asp:ListItem>
<asp:ListItem>1924</asp:ListItem>
<asp:ListItem>1923</asp:ListItem>
<asp:ListItem>1922</asp:ListItem>
<asp:ListItem>1921</asp:ListItem>
<asp:ListItem>1920</asp:ListItem>
```

```
                <asp:ListItem>1919</asp:ListItem>
                <asp:ListItem>1918</asp:ListItem>
                <asp:ListItem>1917</asp:ListItem>
                <asp:ListItem>1916</asp:ListItem>
                <asp:ListItem>1915</asp:ListItem>
                <asp:ListItem>1914</asp:ListItem>
                <asp:ListItem>1913</asp:ListItem>
                <asp:ListItem>1912</asp:ListItem>
                <asp:ListItem>1911</asp:ListItem>
                <asp:ListItem>1910</asp:ListItem>
                <asp:ListItem>1909</asp:ListItem>
                <asp:ListItem>1908</asp:ListItem>
                <asp:ListItem>1907</asp:ListItem>
                <asp:ListItem>1906</asp:ListItem>
                <asp:ListItem>1905</asp:ListItem>
                <asp:ListItem>1904</asp:ListItem>
                <asp:ListItem>1903</asp:ListItem>
                <asp:ListItem>1902</asp:ListItem>
                <asp:ListItem>1901</asp:ListItem>
                <asp:ListItem>1900</asp:ListItem>
               </asp:DropDownList>
               <asp:Label ID="Label14" runat="server"
               Text="Sexo:"></asp:Label>
               <asp:DropDownList ID="DropDownList5"
               runat="server">
                <asp:ListItem>Hombre</asp:ListItem>
                <asp:ListItem>Mujer</asp:ListItem>
            </asp:DropDownList></td>
    </tr>
    <tr>
        <td style="width: 260px; height: 24px;">
            <asp:Label ID="Label7" runat="server"
          Text="Profesión:"></asp:Label></td>
        <td style="width: 436px; height: 24px;">
            <asp:DropDownList ID="DropDownList6"
          runat="server" Width="160px">
                <asp:ListItem>Seleccione de la
                lista</asp:ListItem>
                <asp:ListItem>Abogado</asp:ListItem>
                <asp:ListItem>Administrativo</asp:ListItem>
                <asp:ListItem>Ama de casa</asp:ListItem>
                <asp:ListItem>Arquitecto</asp:ListItem>
                <asp:ListItem>Cient&#237;fico</asp:ListItem>
                <asp:ListItem>Deportista</asp:ListItem>
                <asp:ListItem>Economista</asp:ListItem>
                <asp:ListItem>Estudiante</asp:ListItem>
                <asp:ListItem>Inform&#225;tico</asp:ListItem>
                <asp:ListItem>Ingeniero</asp:ListItem>
                <asp:ListItem>Operario</asp:ListItem>
                <asp:ListItem>Pensionista</asp:ListItem>
                <asp:ListItem>Periodista</asp:ListItem>
                <asp:ListItem>Profesiona de la
                salud</asp:ListItem>
                <asp:ListItem>Profesor</asp:ListItem>
                <asp:ListItem>Otra</asp:ListItem>
            </asp:DropDownList></td>
    </tr>
    <tr>
        <td style="width: 260px">
            <asp:Label ID="Label8" runat="server" Text="E-
          Mail:"></asp:Label></td>
```

```
            <td style="width: 436px">
                <asp:TextBox ID="TextBoxEmail" runat="server"
              Width="150px"></asp:TextBox>
                <asp:RequiredFieldValidator
              ID="RequiredFieldValidator7" runat="server"
        ControlToValidate="TextBoxEmail"
                    ErrorMessage="*">
           </asp:RequiredFieldValidator></td>
        </tr>
        <tr>
            <td style="width: 260px; height: 26px;">
                <asp:Label ID="Label15" runat="server"
                    Text="Nombre de Usuario:"></asp:Label></td>
            <td style="width: 436px; height: 26px;">
                <asp:TextBox ID="TextBoxNombreUsuario"
                    runat="server" Width="150px">
          </asp:TextBox></td>
        </tr>
        <tr>
            <td style="width: 260px">
                <asp:Label ID="Label9" runat="server" Text="Clave
              de Acceso:"></asp:Label></td>
            <td style="width: 436px">
                <asp:TextBox ID="TextBoxClaveAcceso"
              runat="server" Width="150px"
        TextMode="Password"></asp:TextBox>
                <asp:RequiredFieldValidator
              ID="RequiredFieldValidator8" runat="server"
        ControlToValidate="TextBoxClaveAcceso"
                    ErrorMessage="*">
          </asp:RequiredFieldValidator></td>
        </tr>
        <tr>
            <td style="width: 260px">
                <asp:Label ID="Label10" runat="server"
                    Text="Repita la Clave de
              Acceso:"></asp:Label></td>
            <td style="width: 436px">
                <asp:TextBox ID="TextBoxClaveAccesoR"
              runat="server" TextMode="Password"
        Width="150px"></asp:TextBox>
                <asp:CompareValidator ID="CompareValidator1"
              runat="server"
        ControlToCompare="TextBoxClaveAcceso"
                    ControlToValidate="TextBoxClaveAccesoR"
              ErrorMessage="Repita la clave correctamente.">
    </asp:CompareValidator></td>
        </tr>
        <tr>
            <td style="width: 260px; height: 21px;">
                <asp:Label ID="Label11" runat="server"
                    Text="Comentarios:"></asp:Label></td>
            <td style="width: 436px; height: 21px;">
                <asp:TextBox ID="TextBox1" runat="server"
              TextMode="MultiLine"></asp:TextBox></td>
        </tr>
        <tr>
            <td align="center" colspan="2" style="height: 21px">
                <asp:Button ID="Button1" runat="server"
              Text="Enviar Datos" Width="225px"
        OnClick="Button1_Click" />
```

```
                    <asp:Button ID="Button2" runat="server"
                  Text="Recibir Correo" OnClick="Button2_Click"
              /></td>
              </tr>
              <tr>
                  <td align="center" colspan="2" style="height: 21px">
                      <asp:Button ID="Button3" runat="server"
                    OnClick="Button3_Click" Text="Ir al Inicio"
                          UseSubmitBehavior="False"
                    CausesValidation="False" /></td>
              </tr>
          </table>

    </div>
    </form>
</body>
   </html>
```

10.3.2.2. Formulario.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.Odbc;
using System.Net.Sockets;
using MSR.LST.Net.Rtp;
using System.Windows.Forms;



public partial class _Default : System.Web.UI.Page
{

    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void Login1_Authenticate(object sender,
      AuthenticateEventArgs e)
    {
        //Con la ayuda de Configuration Manager recogemos la conexión
      de Web.Config
        OdbcConnection conexion = new OdbcConnection
      (ConfigurationManager. ConnectionStrings
      ["BaseDatos"].ConnectionString);

        string cadena = "SELECT count(*) FROM usuario WHERE usuario
      ='" + Login1.UserName + "' and clave='" +
      Login1.Password+"'";
        string cadenaNombre = "SELECT count(*) FROM usuario WHERE
      usuario='" + Login1.UserName + "'";
```

```
        string cadenaClave = "SELECT count(*) FROM usuario WHERE
      clave='" + Login1.Password + "'";

        //Declaramos el comando SQL
        OdbcCommand comando = new OdbcCommand();

        //Abrimos la conexión
        conexion.Open();

        //Asignar la conexión al comando
        comando.Connection = conexion;

        //Asignamos al comando SQL la sentencia SQL que ejecutará
        comando.CommandText = cadenaNombre;

        //Preparamos la consulta para que nos indique si hay o no
      coincidencia
        int contador=0;
        bool usuario = true;
        bool clave = true;
        contador = Convert.ToInt32(comando.ExecuteScalar());
        if (contador == 0)
        {
                usuario = false;
                Label1.Text= "No está registrado como usuario en este
            servicio";
        }
        else
        {
            comando.CommandText = cadenaClave;
            contador = Convert.ToInt32(comando.ExecuteScalar());
            if (contador == 0)
            {
                clave = false;
                Label1.Text="La Contraseña Introducida es Incorrecta";
            }
        }

        comando.CommandText = cadena;
        contador = Convert.ToInt32(comando.ExecuteScalar());
        if (contador == 1)
        {
            //Damos acceso al usuario

            FormsAuthentication.RedirectFromLoginPage(Login1.UserName,
                  false);


        }


    }

}
```

10.3.3. Página Notificación.

10.3.3.1. Notificacion.aspx.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Notificacion.aspx.cs" Inherits="Default3" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body style="background-position: center; background-image: url(fondo
básico.png); background-repeat: no-repeat">
    <form id="form1" runat="server">
    <div>
        <table style="width: 890px; height: 259px">
            <tr>
                <td style="width: 106px; height: 147px">
                </td>
                <td style="height: 147px">
                </td>
                <td style="width: 58px; height: 147px">
                </td>
            </tr>
            <tr>
                <td style="width: 106px">
                </td>
                <td>
                    <asp:Label ID="Label1" runat="server" Font-
                  Bold="True" Font-Italic="True" Font-
            Names="Tahoma"
                        ForeColor="Blue"
                  Text="Label"></asp:Label></td>
                <td style="width: 58px">
                </td>
            </tr>
            <tr>
                <td style="width: 106px">
                </td>
                <td>
                    <asp:Button ID="Button1" runat="server"
                  OnClick="Button1_Click" Text="Ir al Inicio" />
                    <asp:Button ID="Button2" runat="server"
                  OnClick="Button2_Click" Text="Ir a Registro"
            /></td>
                <td style="width: 58px">
                </td>
            </tr>
        </table>

    </div>
    </form>
</body>

</html>
```

10.3.3.2. Notificacion.aspx.cs.

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class Default3 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = "Debe de auntetificarse para poder visualizar el
Catálogo.";
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Redirect("Default.aspx");

    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        Response.Redirect("Formulario.aspx");
    }
}
```

10.3.4. Página Estadísticas

10.3.4.1. Estadisticas.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Estadisticas.aspx.cs" Inherits="Estadisticas" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body style="background-position: center; background-image: url(fondo
básico.png); background-repeat: no-repeat">
    <form id="form1" runat="server">
    <div>
         </div>
          
        <table style="width: 842px; height: 200px">
            <tr>
                <td style="width: 295px; height: 99px">
```

```
                </td>
                <td style="height: 99px">
                </td>
                <td style="height: 99px">
                </td>
            </tr>
            <tr>
                <td style="width: 295px; height: 94px">
                </td>
                <td style="height: 94px">
      <asp:GridView ID="GridView1" runat="server" CellPadding="4"
    ForeColor="#333333" GridLines="None" Height="254px"
    Width="394px">
          <FooterStyle BackColor="#507CD1" Font-Bold="True"
          ForeColor="White" />
          <RowStyle BackColor="#EFF3FB" />
          <EditRowStyle BackColor="#2461BF" />
          <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True"
          ForeColor="#333333" />
          <PagerStyle BackColor="#2461BF" ForeColor="White"
          HorizontalAlign="Center" />
          <HeaderStyle BackColor="#507CD1" Font-Bold="True"
          ForeColor="White" />
          <AlternatingRowStyle BackColor="White" />
      </asp:GridView>
                </td>
                <td style="height: 94px">
                </td>
            </tr>
            <tr>
                <td style="width: 295px; height: 21px">
                </td>
                <td style="height: 21px">
      <asp:Label ID="Label1" runat="server" Text="Label"
          Width="393px"></asp:Label></td>
                <td style="height: 21px">
                </td>
            </tr>
            <tr>
                <td style="width: 295px; height: 21px">
                </td>
                <td style="height: 21px">
                    <asp:Button ID="Button1" runat="server"
                  OnClick="Button1_Click" Text="Ir al Inicio"
                        Width="120px" /></td>
                <td style="height: 21px">
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

10.3.4.2. Estadisticas.aspx.cs.

```
using System;
using System.Data;
using System.Configuration;
```

```
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.Odbc;

public partial class Estadisticas : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        OdbcConnection conexionGrid = new
      OdbcConnection(ConfigurationManager.
      ConnectionStrings["BaseDatos"].ConnectionString);

        string cadena = "SELECT nombre,comentario,visitas FROM video
      ORDER BY visitas DESC";

        OdbcDataAdapter da;

        DataTable dt = new DataTable();

        try
        {
            da =new OdbcDataAdapter(cadena, conexionGrid);
            da.Fill(dt);

            this.GridView1.DataSource = dt;
            this.GridView1.DataBind();
            Label1.Text = String.Format("Total videos en la tabla:
            {0}", dt.Rows.Count);

        }
        catch (Exception ex)
        {
            Label1.Text = "Error: " + ex.Message;
        }
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Redirect("Default.aspx");
    }
}
```

10.3.5. Página EstadisticasUsuario

10.3.5.1. EstadisticasUsuario.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="EstadisticasUsuario.aspx.cs" Inherits="EstadisticasUsuario"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body style="background-position: center; background-image: url(fondo
básico.png); background-repeat: no-repeat">
    <form id="form1" runat="server">
    <div>
         </div>
          
        <table style="width: 842px; height: 200px">
            <tr>
                <td style="width: 295px; height: 99px">
                </td>
                <td style="height: 99px">
                </td>
                <td style="height: 99px">
                </td>
            </tr>
            <tr>
                <td style="width: 295px; height: 94px">
                </td>
                <td style="height: 94px">
      <asp:GridView ID="GridView1" runat="server" CellPadding="4"
     ForeColor="#333333" GridLines="None" Height="254px"
     Width="394px">
           <FooterStyle BackColor="#507CD1" Font-Bold="True"
           ForeColor="White" />
           <RowStyle BackColor="#EFF3FB" />
           <EditRowStyle BackColor="#2461BF" />
           <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True"
           ForeColor="#333333" />
           <PagerStyle BackColor="#2461BF" ForeColor="White"
           HorizontalAlign="Center" />
           <HeaderStyle BackColor="#507CD1" Font-Bold="True"
           ForeColor="White" />
           <AlternatingRowStyle BackColor="White" />
      </asp:GridView>
                </td>
                <td style="height: 94px">
                </td>
            </tr>
            <tr>
                <td style="width: 295px; height: 21px">
                </td>
                <td style="height: 21px">
      <asp:Label ID="Label1" runat="server" Text="Label"
          Width="393px"></asp:Label></td>
                <td style="height: 21px">
                </td>
            </tr>
            <tr>
                <td style="width: 295px; height: 21px">
                </td>
                <td style="height: 21px">
                    <asp:Button ID="Button1" runat="server"
                  OnClick="Button1_Click" Text="Volver a
           Catálogo"
                        Width="120px" /></td>
                <td style="height: 21px">
```

```
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

10.3.5.2. EstadisticasUsuario.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.Odbc;

public partial class EstadisticasUsuario : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        OdbcConnection conexionGrid = new
      OdbcConnection(ConfigurationManager.
      ConnectionStrings["BaseDatos"].ConnectionString);

        string cadena = "SELECT nombre,comentario,visitas FROM video
      ORDER BY visitas DESC";

        OdbcDataAdapter da;

        DataTable dt = new DataTable();

        try
        {
            da = new OdbcDataAdapter(cadena, conexionGrid);
            da.Fill(dt);

            this.GridView1.DataSource = dt;
            this.GridView1.DataBind();
            Label1.Text = String.Format("Total videos en la tabla:
      {0}", dt.Rows.Count);

        }
        catch (Exception ex)
        {
            Label1.Text = "Error: " + ex.Message;
        }
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Redirect("Catalogo.aspx");
    }

}
```

10.3.6. Página Catalogo

10.3.6.1. Catalogo.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Catalogo.aspx.cs" Inherits="Default2" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>:::Tu Servidor Web:::</title>
</head>
<body style="background-position: center; background-image: url(fondo
básico.png); background-repeat: no-repeat">
    <form id="form1" runat="server"   >
    <div>
        <br />
        <br />
        <br />
        <br />
        <br />

        <br />
        <br />
        <br />
         
    </div>
               
        <table style="left: 150px; position: relative; top: 0px">
            <tr>
                <td rowspan="1" style="width: 116px">
                </td>
                <td rowspan="1" style="width: 372px">
                     </td>
                <td style="width: 42px; height: 21px">
                </td>
                <td style="width: 125px; height: 21px">
                </td>
            </tr>
            <tr>
                <td rowspan="3" style="width: 116px">
                    <img src="DSC00043.JPG" alt="" height="100"
                  style="width: 125px" /></td>
                <td rowspan="2" style="width: 372px">
                    <asp:Label ID="Label2" runat="server" Font-
                  Bold="True" Font-Italic="True" Font-
            Names="Tahoma"
                        ForeColor="Blue" Text='Ana y Sabina en una
                  versión de la canción "Doña
            Semana"'></asp:Label></td>
                <td style="width: 42px; height: 21px">
                </td>
                <td style="width: 125px; height: 21px">
                </td>
            </tr>
            <tr>
```

```
        <td style="width: 42px; height: 21px">
        </td>
        <td style="width: 125px; height: 21px">
        </td>
    </tr>
    <tr>
        <td style="width: 372px; height: 21px">

        </td>
        <td style="width: 42px; height: 21px">
            <asp:Button ID="Button4" runat="server" Text="Ver"
                Width="40px" OnClick="Button4_Click"/></td>
        <td style="width: 125px; height: 21px">
            <asp:Button ID="Button1" runat="server"
          Text="Descargar" OnClick="Button1_Click"
/></td>
        </tr>
    <tr>
        <td style="width: 116px; height: 20px">
        </td>
        <td style="width: 372px; height: 20px">
        </td>
        <td style="width: 42px; height: 20px">
        </td>
        <td style="width: 125px; height: 20px">
        </td>
    </tr>
    <tr>
        <td rowspan="3" style="width: 116px">
            <img src="DSC00178.JPG" alt="" height="100"
          style="width: 125px" /></td>
        <td rowspan="2" style="width: 372px">
            <asp:Label ID="Label3" runat="server" Font-
          Bold="True" Font-Italic="True" Font-
Names="Tahoma"
                ForeColor="Blue" Text="Espectacular actuación
          de Carlos en Inazares"></asp:Label></td>
        <td style="width: 42px">
        </td>
        <td style="width: 125px">
        </td>
    </tr>
    <tr>
        <td style="width: 42px">
        </td>
        <td style="width: 125px">
        </td>
    </tr>
    <tr>
        <td style="width: 372px">
        </td>
        <td style="width: 42px">
            <asp:Button ID="Button5" runat="server" Text="Ver"
                Width="40px" OnClick="Button5_Click" /></td>
        <td style="width: 125px">
            <asp:Button ID="Button2" runat="server"
          Text="Descargar" Width="90px"
OnClick="Button2_Click" /></td>
    </tr>
    <tr>
        <td rowspan="1" style="width: 116px; height: 20px;">
```

```
                </td>
                <td style="width: 372px; height: 20px;">
                </td>
                <td style="width: 42px; height: 20px">
                </td>
                <td style="width: 125px; height: 20px;">
                </td>
            </tr>
            <tr>
                <td rowspan="3" style="width: 116px">
                    <img src="pistolas.jpg" alt="" height="100"
/></td>
                <td rowspan="2" style="width: 372px">
                    <asp:Label ID="Label1" runat="server" Font-
                  Bold="True" Font-Italic="True" Font-
            Names="Tahoma"
                        ForeColor="Blue" Text="Pistolas con el
                  megáfono en una versión de la Ramona "
                        Width="373px"></asp:Label></td>
                <td style="width: 42px">
                </td>
                <td style="width: 125px">
                </td>
            </tr>
            <tr>
                <td style="width: 42px">
                </td>
                <td style="width: 125px">
                </td>
            </tr>
            <tr>
                <td style="width: 372px">
                </td>
                <td style="width: 42px">
                    <asp:Button ID="Button6" runat="server" Text="Ver"
                        Width="40px" OnClick="Button6_Click" /></td>
                <td style="width: 125px">
                    <asp:Button ID="Button3" runat="server"
                  Text="Descargar" Width="90px"
            OnClick="Button3_Click" /></td>
            </tr>
            <tr>
                <td rowspan="1" style="width: 116px">
                </td>
                <td style="width: 372px">
                    <asp:Button ID="Button7" runat="server"
                  OnClick="Button7_Click" Text="Ir al Inicio"
                        Width="180px" />
                    <asp:Button ID="Button8" runat="server"
                  OnClick="Button8_Click" Text="Ver Estadísticas"
                        Width="180px" /></td>
                <td style="width: 42px">
                </td>
                <td style="width: 125px">
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

10.3.6.2. Catalogo.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.ComponentModel;
using System.Windows.Forms;
using System.Drawing;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Data.Odbc;



public partial class Default2 : System.Web.UI.Page
{
    OdbcConnection conexion = new OdbcConnection(ConfigurationManager.
      ConnectionStrings["BaseDatos"].ConnectionString);
    public static string video = null;
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        string filepath = @"Videos\MOV00041.MPG";
        string filename = "MOV00041.MPG";
        descarga(filepath, filename);

    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        string filepath = @"Videos\MOV00094.MPG";
        string filename = "MOV00094.MPG";
        descarga(filepath, filename);

    }
    protected void Button3_Click(object sender, EventArgs e)
    {
        string filepath = @"Videos\ramona.mp4";
        string filename = "ramona.mp4";
        descarga(filepath, filename);

    }

    public void descarga(string filepath, string filename)
    {
        Response.Clear();
        Response.ContentType = "application/octet-stream";
        Response.AddHeader("Content-Disposition", "attachment;
      filename=" + filename);
```

```
        Response.Flush();
        Response.WriteFile(filepath);
        Response.End();
    }
    /*Esta función se puede ejecutar desde donde queramos (lo típico
      es llamarlo cuando el usuario presiona un botón). básicamente,
      lo que hace es:

      1.- Limpia el contenido de salida.
      2.- Le cambia el contentType a tipo octet... aquí es donde
      "engañamos al navegador".
      3.- Le añadimos la cabecera Content-Disposition y le damos un
      nombre al fichero. Esto es opcional, y lo que hace es dar el
      nombre que queremos que aparezca si el usuario decide guardar el
      fichero.
      4.- Manda la info que tenemos hasta ahora (la única cabecera que
      hemos añadido) a la salida hacia el usuario.
      5.- Mandamos el fichero en sí desde
      Response.WriteFile(filepath), donde, obviamente, filepath es el
      path interno del fichero en nuestro servidor.
      6.- Enviamos todo y terminamos la ejecución de la página.*/


    public void Button4_Click(object sender, EventArgs e)
    {
        string cadena1 = "SELECT visitas FROM video where
      idVideo='1'";
        OdbcCommand comando1 = new OdbcCommand(cadena1, conexion);
            comando1.Connection.Open();
        double visitasNuevas =
      Convert.ToInt32(comando1.ExecuteScalar())+ 1;
        string cadena2 = "UPDATE video SET visitas='"+visitasNuevas+"'
            where idVideo='1'";
        OdbcCommand comando2 = new OdbcCommand(cadena2, conexion);
        conexion.Close();
        comando2.Connection.Open();
        comando2.ExecuteNonQuery();
        video = @"Videos\MOV00041.MPG";
        Response.Redirect("vervideo.aspx?video="+video);

    }

    public void Button5_Click(object sender, EventArgs e)
    {
        string cadena1 = "SELECT visitas FROM video where
      idVideo='3'";
        OdbcCommand comando1 = new OdbcCommand(cadena1, conexion);
        comando1.Connection.Open();
        double visitasNuevas =
      Convert.ToInt32(comando1.ExecuteScalar()) + 1;
        string cadena2 = "UPDATE video SET visitas='" + visitasNuevas
      + "' where idVideo='3'";
        OdbcCommand comando2 = new OdbcCommand(cadena2, conexion);
        conexion.Close();
        comando2.Connection.Open();
        comando2.ExecuteNonQuery();
        video = @"Videos\MOV00094.MPG";
        Response.Redirect("vervideo.aspx?video="+video);
    }
```

```
    public void Button6_Click(object sender, EventArgs e)
    {
        string cadena1 = "SELECT visitas FROM video where
      idVideo='2'";
        OdbcCommand comando1 = new OdbcCommand(cadena1, conexion);
        comando1.Connection.Open();
        double visitasNuevas =
      Convert.ToInt32(comando1.ExecuteScalar()) + 1;
        string cadena2 = "UPDATE video SET visitas='" + visitasNuevas
      + "' where idVideo='2'";
        OdbcCommand comando2 = new OdbcCommand(cadena2, conexion);
        conexion.Close();
        comando2.Connection.Open();
        comando2.ExecuteNonQuery();
        video = @"Videos\ramona.mp4";
        Response.Redirect("vervideo.aspx?video="+video);

    }


    protected void Button7_Click(object sender, EventArgs e)
    {
        Response.Redirect("Default.aspx");
    }
    protected void Button8_Click(object sender, EventArgs e)
    {
        Response.Redirect("EstadisticasUsuario.aspx");
    }
}
```

10.3.7. Página Web.Config

```
<?xml version="1.0"?>
<!--
    Note: As an alternative to hand editing this file you can use the
    web admin tool to configure settings for your application. Use
    the Website->Asp.Net Configuration option in Visual Studio.
    A full list of settings and comments can be found in
    machine.config.comments usually located in
    \Windows\Microsoft.Net\Framework\v2.x\Config
-->
<configuration>
      <connectionStrings>
            <!--Preparo el texto para insertar la cadena de conexión--
                  >
            <add name="BaseDatos" connectionString="DRIVER={MySQL ODBC
                  3.51 Driver};SERVER=localhost;
            DATABASE=clientes;USER=root;PASSWORD=1234;"/>
      </connectionStrings>
      <location path="Formulario.aspx">
            <system.web>
                  <authorization>
                  <allow users="?"/>
            </authorization>
            </system.web>
      </location>
      <system.web>
            <compilation debug="true">
                  <assemblies>
```

```
                        <add assembly="System.Windows.Forms,
                        Version=2.0.0.0, Culture=neutral,
                  PublicKeyToken=B77A5C561934E089"/>
                        <add assembly="System.Configuration.Install,
                        Version=2.0.0.0, Culture=neutral,
                  PublicKeyToken=B03F5F7F11D50A3A"/>
                  </assemblies>
            </compilation>
            <authentication mode="Forms">
                  <forms loginUrl="Default.aspx"
                  defaultUrl="Catalogo.aspx">
                  </forms>
            </authentication>
      </system.web>
</configuration>
```

11. Bibliografía

[1] Microsof Visual C# . NET Aprenda Ya
John Sharp
Jon Jagger
Ed. McGrawHill Profesional ISBN: 84-481-3338-2

[2] ASP.NET Programación Web con Visual Studio y Web Matriz
Oliver Dewit
Ed. eni ediciones ISBN: 2-7460-1978-7

[3] Aprendiendo ASP.NET
Chris Payne
Ed. Prentice Hall ISBN: 970-26-0340-4

[4] ASP.NET
Mridula Parilar
Ed. ANAYA ISBN: 84-415-1385-6

[5] Bases de Datos Relacionales
Matilde Celma Giménez
Juan Carlos Casamayor Ródenas
Laura Mota Herranz
Ed. Prentice Hall ISBN: 84-205-3850-7

[6] Revista dotNetMania, disponible en Internet en: www.dotnetmania.com

[7] MSDN Online en Español, disponible en: http://www.microsoft.com/spanish/msdn/spain/default.aspx

[8] Portal El Guille: http://www.elguille.info