TI-Capital Humano

Desarrollador .Net

Manual del Curso

Contenido

1 Introducción a ASP .NET

ASP.NET es un framework para aplicaciones web desarrollado y comercializado por Microsoft. Es usado por programadores para construir sitios web dinámicos, aplicaciones web y servicios web XML. El código de las aplicaciones puede escribirse en cualquier lenguaje compatible con el Common Language Runtime (CLR), entre ellos Microsoft Visual Basic, C#, JScript .NET y J#. Estos lenguajes permiten desarrollar aplicaciones ASP.NET que se benefician del Common Language Runtime, seguridad de tipos, herencia, etc.

2 FrameWork de páginas y controles

El FrameWork de páginas y controles ASP.NET es un marco de trabajo de programación que se ejecuta en un servidor Web para generar y representar de forma dinámica páginas Web ASP.NET. Las páginas Web ASP.NET se pueden solicitar a cualquier explorador o dispositivo del cliente y ASP.NET representa el marcado (como HTML) al explorador que realizó la solicitud. Como norma, puede utilizar la misma página para varios exploradores, porque ASP.NET representa el marcado adecuado para el explorador que realiza la solicitud. Sin embargo, puede diseñar una página Web ASP.NET para ejecutarse en un explorador determinado, es compatible con los controles móviles de los dispositivos preparados para trabajar en Web como teléfonos celulares, PC portátiles y asistentes digitales personales (PDA).

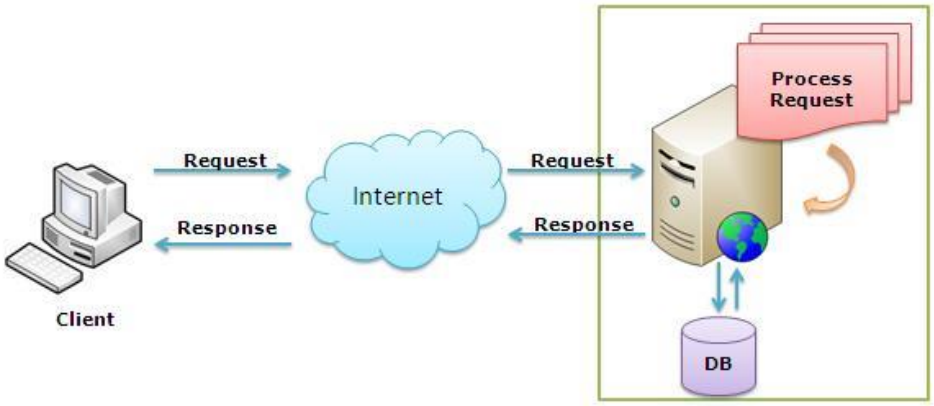
Las páginas Web ASP.NET están completamente orientadas a objetos. En las páginas Web ASP.NET se puede trabajar con elementos HTML que usen propiedades, métodos y eventos. El marco de trabajo de páginas ASP.NET elimina los detalles de implementación relacionados con la separación de cliente y servidor inherente a las aplicaciones Web presentando un modelo unificado que responde a los eventos de los clientes en el código que se ejecuta en el servidor. El marco de trabajo también mantiene automáticamente el estado de la página y de los controles que contenga durante el ciclo vital de procesamiento de la página.

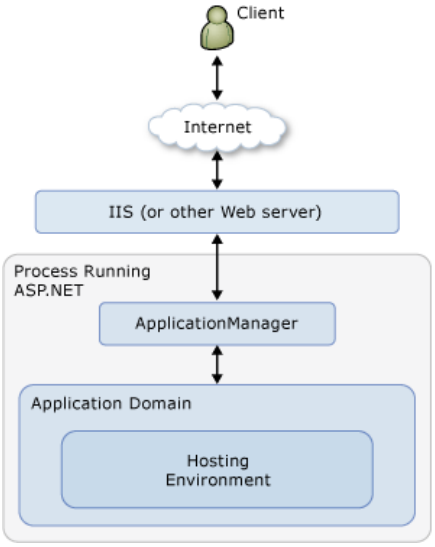
El marco de trabajo de páginas y controles ASP.NET también permite encapsular la funcionalidad común de la interfaz de usuario en controles fáciles de usar y reutilizables. Los controles se escriben una vez, se pueden utilizar en varias páginas y se integran en la página Web ASP.NET en la que se colocan durante la representación.

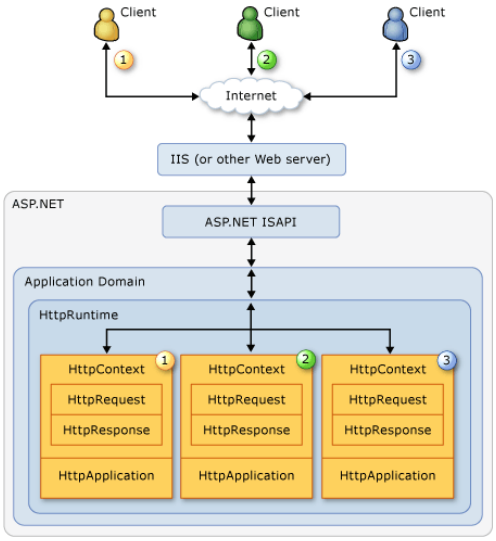
El marco de trabajo de páginas y controles ASP.NET también proporciona funciones para controlar la apariencia y el funcionamiento general de los sitios Web a través de temas y máscaras. Se pueden definir temas y máscaras y, a continuación, aplicarlos en las páginas o controles.

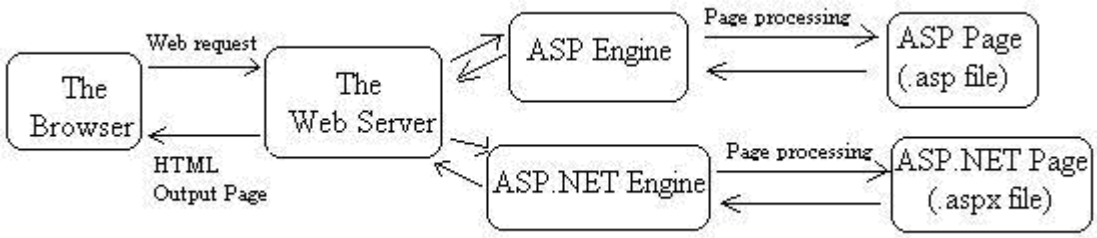
Además de los temas, es posible definir páginas principales que se crean para conseguir un diseño coherente en las páginas de la aplicación. Una página principal única define el diseño y el comportamiento estándar deseados para todas las páginas (o un grupo de páginas) de la aplicación. A continuación, se pueden crear páginas de contenido individuales con el contenido específico de la página que se desee mostrar. Cuando los usuarios solicitan las páginas de contenido, las combinan con la página principal con el fin de generar un resultado que combine el diseño de la página principal con el de la página de contenido.

3 Flujo aplicación Web con ASP .NET y IIS




Client
Internet
IIS (or other Web server)
Process Running ASP.NET
ApplicationManager
Application Domain
Hosting Environment


Client
Client
Client
1
2
3
Internet
IIS (or other Web server)
ASP.NET
ASP.NET ISAPI
Application Domain
HttpRuntime
HttpContext
HttpRequest
HttpResponse
HttpApplication
HttpContext
HttpRequest
HttpResponse
HttpApplication
HttpContext
HttpRequest
HttpResponse
HttpApplication



Dentro de ASP.NET, deben ocurrir varios pasos de procesamiento para que una aplicación ASP.NET se inicialice y procese las solicitudes. Además, ASP.NET es solo una pieza de la arquitectura del servidor web que atiende las solicitudes realizadas por los navegadores. Es importante que comprenda el ciclo de vida de la aplicación para que pueda escribir código en la etapa de ciclo de vida adecuada para el efecto que desea.

Etapas

1. El usuario solicita un recurso de aplicación del servidor web.

El ciclo de vida de una aplicación ASP.NET comienza con una solicitud enviada por un navegador al servidor web (para aplicaciones ASP.NET, normalmente IIS). ASP.NET es una extensión ISAPI del servidor web. Cuando un servidor web recibe una solicitud, examina la extensión del nombre de archivo del archivo solicitado, determina qué extensión ISAPI debe manejar la solicitud y luego pasa la solicitud a la extensión ISAPI adecuada. ASP.NET maneja las extensiones de nombre de archivo que se le han asignado, como .aspx, .ascx, .ashx y .asmx.

Si no se ha asignado una extensión de nombre de archivo a ASP.NET, ASP.NET no recibirá la solicitud. Es importante comprender esto para las aplicaciones que utilizan la autenticación ASP.NET. Por ejemplo, debido a que los archivos .htm normalmente no se asignan a ASP.NET, ASP.NET no realizará comprobaciones de autenticación o autorización en las solicitudes de archivos .htm. Por lo tanto, incluso si un archivo contiene solo contenido estático, si desea que ASP.NET verifique la autenticación, cree el archivo con una extensión de nombre de archivo asignada a ASP.NET, como .aspx.

Si crea un controlador personalizado para dar servicio a una extensión de nombre de archivo en particular, debe asignar la extensión a ASP.NET en IIS y también registrar el controlador en el archivo Web.config de su aplicación.

2. ASP.NET recibe la primera solicitud de la aplicación.

Cuando ASP.NET recibe la primera solicitud de cualquier recurso en una aplicación, una clase llamada ApplicationManager crea un dominio de aplicación.

Los dominios de aplicación proporcionan aislamiento entre aplicaciones para variables globales y permiten que cada aplicación se descargue por separado. Dentro de un dominio de aplicación, se crea una instancia de la clase denominada HostingEnvironment, que proporciona acceso a información sobre la aplicación, como el nombre de la carpeta donde se almacena la aplicación.

ASP.NET también compila los elementos de nivel superior en la aplicación si es necesario, incluido el código de la aplicación en la carpeta App_Code.

3. Los objetos principales de ASP.NET se crean para cada solicitud.

Una vez que se ha creado el dominio de la aplicación y se ha creado una instancia del objeto HostingEnvironment, ASP.NET crea e inicializa objetos principales como HttpContext, HttpRequest y HttpResponse. La clase HttpContext contiene objetos que son específicos de la solicitud de aplicación actual, como los objetos HttpRequest y HttpResponse. El objeto HttpRequest contiene información sobre la solicitud actual, incluidas las cookies y la información del navegador. El objeto HttpResponse contiene la respuesta que se envía al cliente, incluidos todos los resultados y cookies representados.

Una vez que se ha creado el dominio de la aplicación y se ha creado una instancia del objeto HostingEnvironment, ASP.NET crea e inicializa objetos centrales como HttpContext, HttpRequest y HttpResponse. La clase HttpContext contiene objetos que son específicos de la solicitud de aplicación actual, como los objetos HttpRequest y HttpResponse. El objeto HttpRequest contiene información sobre la solicitud actual, incluidas las cookies y la información del navegador. El objeto HttpResponse contiene la respuesta que se envía al cliente, incluidos todos los resultados y cookies representados.

4. Se asigna un objeto HttpApplication a la solicitud

Una vez que se han inicializado todos los objetos de la aplicación principal, la aplicación se inicia creando una instancia de la clase HttpApplication. Si la aplicación tiene un archivo Global.asax, ASP.NET crea en su lugar una instancia de la clase Global.asax que se deriva de la clase HttpApplication y utiliza la clase derivada para representar la aplicación.

La primera vez que se solicita una página o proceso ASP.NET en una aplicación, se crea una nueva instancia de HttpApplication. Sin embargo, para maximizar el rendimiento, las instancias de HttpApplication pueden reutilizarse para varias solicitudes.

Cuando se crea una instancia de HttpApplication, también se crean los módulos configurados. Por ejemplo, si la aplicación está configurada para hacerlo, ASP.NET crea un

Módulo SessionStateModule. Una vez creados todos los módulos configurados, se llama al método Init de la clase HttpApplication.

5. La solicitud es procesada por la canalización HttpApplication.

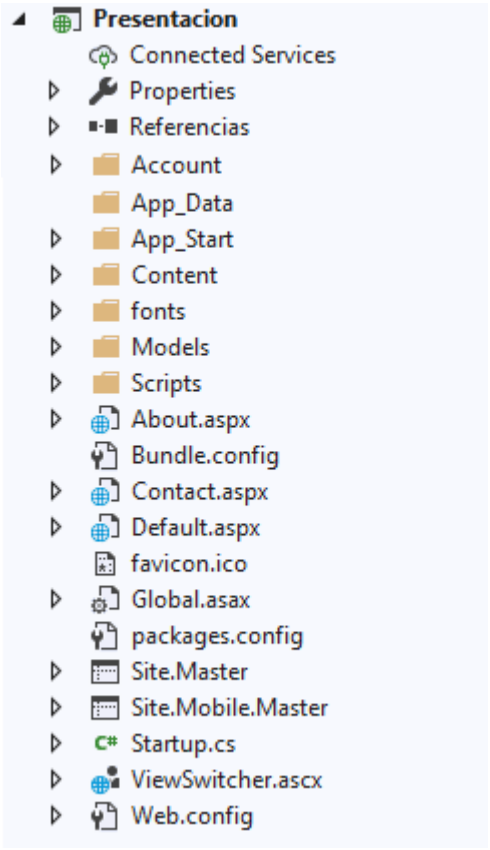
Los siguientes eventos son ejecutados por la clase HttpApplication mientras se procesa la solicitud. Los eventos son de particular interés para los desarrolladores que desean ampliar la clase HttpApplication.

1. **Valide la solicitud**, que examina la información enviada por el navegador y determina si contiene marcas potencialmente maliciosas. La validación de la solicitud se realiza comparando todos los datos de entrada con una lista de valores potencialmente peligrosos. Si se produce una coincidencia, ASP.NET genera una HttpRequestValidationException.
2. **Realice la asignación de URL**, si se ha configurado alguna URL en la sección UrlMappingsSection del archivo Web.config.
3. **Genere el evento BeginRequest**. El evento BeginRequest señala la creación de cualquier nueva solicitud dada. Este evento siempre se genera y es siempre el primer evento que ocurre durante el procesamiento de una solicitud.
4. **Genera el evento AuthenticateRequest.** Ocurre cuando un módulo de seguridad ha establecido la identidad del usuario. El evento AuthenticateRequest indica que el mecanismo de autenticación configurado ha autenticado la solicitud actual. La suscripción al evento AuthenticateRequest garantiza que la solicitud se autenticará antes de procesar el módulo adjunto o el controlador de eventos.
5. **Genere el evento PostAuthenticateRequest**.
6. **Genere el evento AuthorizeRequest**. Ocurre cuando un módulo de seguridad ha verificado la autorización del usuario. El evento AuthorizeRequest indica que ASP.NET ha autorizado la solicitud actual. La suscripción al evento AuthorizeRequest garantiza que la solicitud se autenticará y autorizará antes de procesar el módulo adjunto o el controlador de eventos.
7. **Genere el evento PostAuthorizeRequest.**
8. **Genere el evento ResolveRequestCache**. Ocurre cuando ASP.NET completa un evento de autorización para permitir que los módulos de almacenamiento en caché atiendan solicitudes del caché, sin pasar por la ejecución del controlador de eventos (por ejemplo, una página o un servicio web XML).
9. **Genere el evento PostResolveRequestCache**.
10. Según la extensión del nombre de archivo del recurso solicitado (asignado en el archivo de configuración de la aplicación), seleccione una clase que implemente IHttpHandler para procesar la solicitud. Si la solicitud es para

un objeto (página) derivado de la clase Page y la página debe compilarse, ASP.NET compila la página antes de crear una instancia de la misma.
11. Genere el evento PostMapRequestHandler. Se produce cuando ASP.NET ha asignado la solicitud actual al controlador de eventos adecuado.
12. Genere el evento AcquireRequestState. Ocurre cuando ASP.NET adquiere el estado actual (por ejemplo, el estado de la sesión) que está asociado con la solicitud actual. El evento AcquireRequestState se genera después de que se haya creado el controlador de eventos.
13. Genere el evento PostAcquireRequestState.
14. Genere el evento PreRequestHandlerExecute.
15. Llame al método ProcessRequest (o la versión asincrónica BeginProcessRequest) de la clase IHttpHandler adecuada para la solicitud. Por ejemplo, si la solicitud es para una página, la instancia de la página actual maneja la solicitud. Habilita el procesamiento de solicitudes web HTTP mediante un HttpHandler personalizado que implementa la interfaz IHttpHandler. Coloque su código HttpHandler personalizado en el método virtual ProcessRequest.
16. Genere el evento PostRequestHandlerExecute.
17. Genere el evento ReleaseRequestState.
18. Genere el evento PostReleaseRequestState.
19. Realice un filtrado de respuestas si la propiedad Filtro está definida.
20. Genere el evento UpdateRequestCache.
21. Genere el evento PostUpdateRequestCache.
22. Genere el evento EndRequest.

3.1Crear aplicación Web con ASP .NET

3.1.1 Estructura de una Aplicación Web Forms

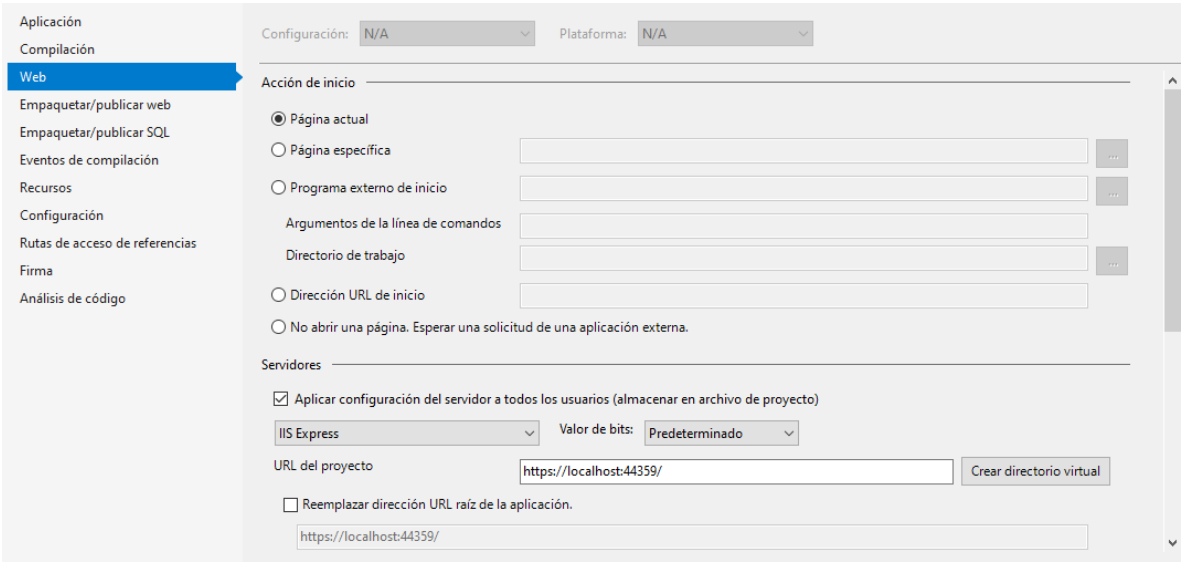


1. Properties

En cuanto a las propiedad del Proyecto, una diferencia importante con respecto a los Proyectos anteriormente vistos, es la pestaña Web.

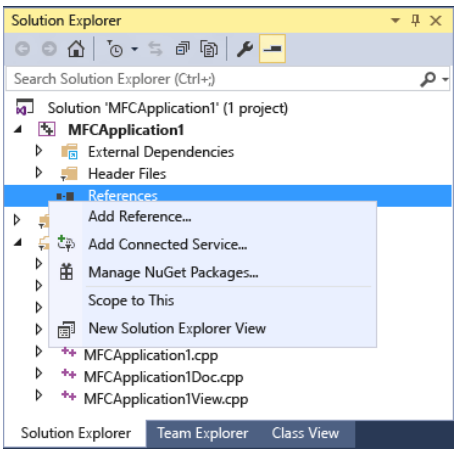
En esta pestaña podemos establecer con cuál página se iniciará nuestra aplicación al momento de ejecutarla:

- Pagina actual .- Indicará que la página que tenemos en la ventana principal de nuestro IDE será la que se muestre en primera instancia.
- Página específica.- Indicará que la página que será siempre la que se muestre en primera instancia.
- En momento de desarrollo se recomienda tenerla en Página Actual, para tener flexibilidad de ir construyendo y probando cada una de las páginas que compondrán nuestra aplicación; pero cuando se pase a producción deberá establecerse la página inicial de la aplicación.
-



2- References.

Al igual que eAntes de escribir código en un componente externo o en un servicio conectado, el proyecto debe contener primero una referencia a él. Una referencia es básicamente una entrada de un archivo de proyecto que contiene la información que Visual Studio necesita para localizar el componente o el servicio.Para agregar una referencia, haga clic con el botón derecho en el nodo **Referencias** o **Dependencias**del **Explorador de soluciones** y elija **Agregar referencia**. También puede hacer clic con el botón derecho en el nodo del proyecto y seleccionar **Agregar** > **Referencia**. Para obtener más información, vea Adición o eliminación de referencias.



Puede agregar una referencia a los siguientes tipos de componentes y servicios:

- Bibliotecas de clases o ensamblados de .NET Framework
- Aplicaciones para UWP

- componentes COM
- Otros ensamblados o bibliotecas de clases de proyectos de la misma solución
- servicios Web XML
-

3- BundleConfig.cs

Un Bundle es una agrupación de enlaces a archivos externos desde el HTML. Es algo habitual. Para evitar una tarea tan repetitiva se han creado estos Bundles, que son accesos directos a agrupaciones de este tipo de archivos. Por ejemplo, este conjunto de llamadas a scripts de javascript:

```
<scriptsrc="~/Scripts/jquery-2.1.1.min.js"></script>

<scriptsrc="~/Scripts/jquery-ui-1.11.1.js"></script>

<scriptsrc="~/Scripts/jquery.validate.min.js"></script>

<scriptsrc="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
```

Puede simplificarse a un Bundle como este:

<%: Scripts.Render("~/bundles/modernizr") %>

La ventaja simplificando el código es clara. En este caso solo hay cuatro scripts, pero podría haber docenas. Además, no solo se pueden agrupar scripts, sino también Hojas de estilo.

¿Dónde se configuran?

Puedes crear muchos Bundles, conteniendo numerosos scripts u hojas de estilo. Todos ellos se construyen dentro del fichero **BundleConfig.cs** en el **App_Start**.

4- RouteConfig.cs

Primeramente es necesario explicar lo que es el enrutamiento.

El enrutamiento permite a la aplicación web usar direcciones URL que son fáciles de recordar y que son más compatibles con los motores de búsqueda.

El enrutamiento de direcciones URL permite configurar una aplicación para que acepte direcciones URL de solicitud que no se asignan a archivos físicos. Una dirección URL de solicitud es simplemente la dirección URL que un usuario escribe en el explorador para buscar una página en el sitio Web. El enrutamiento se usa para definir direcciones URL semánticamente significativas para los usuarios y que pueden ayudar con la optimización del motor de búsqueda (SEO).

De forma predeterminada, la plantilla de formularios Web Forms incluye direcciones URL descriptivas de ASP.net. Gran parte del trabajo de enrutamiento básico se implementará mediante el uso de direcciones URL descriptivas, donde prácticamente la ruta de la página es la dirección física de la página o recurso, como sigue:

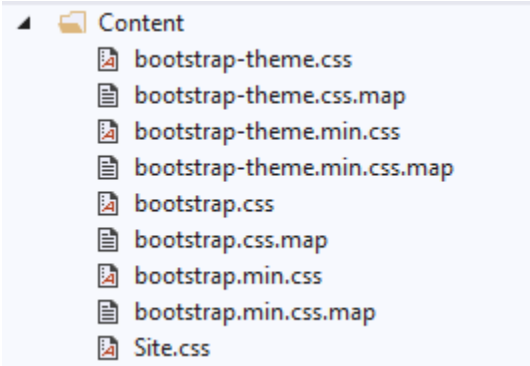
https://localhost:44300/Alumnos.aspx?IdAlumno=2

Rutas

Una ruta es un patrón de dirección URL que se asigna a un controlador. El controlador puede ser un archivo físico, como un archivo. aspx, en una aplicación de formularios Web Forms. Un controlador también puede ser una clase que procesa la solicitud. Para definir una ruta, cree una instancia de la clase Route especificando el patrón de la dirección URL, el controlador y, opcionalmente, un nombre para la ruta.

5- Content.

El directorio Content está pensado para el contenido estático de la aplicación, especialmente útil para archivos css e imágenes asociadas.



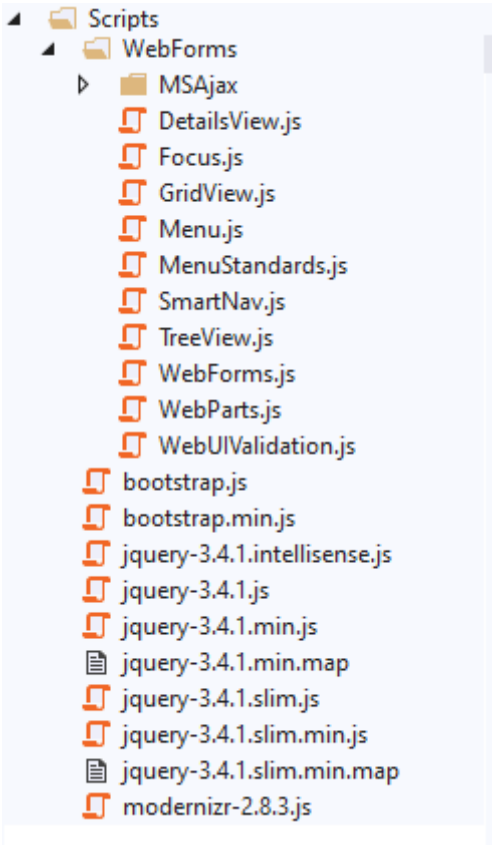
ASP.NET Web Form nos ofrece por defecto las clases definidas en bootstrap

6- Fonts

El directorio fonts está pensado para ubicar los archivos utilizados para las fuentes e iconos que utilizarías en una aplicación, por default la plantilla viene cargada con los fonts de bootstrap.

7- Scripts

El directorio scripts está pensado para ubicar los archivos de javascript (*.js). El código javascript es ejecutado en el contexto del navegador, es decir, en la parte cliente, y nos permite ejecutar acciones sin necesidad de enviar los datos al servidor.



ASP.NET Web Forms incluye varias librerias de javascript por defecto:

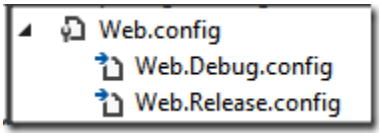
- **jquery.js**. Esta popular y súper útil librería nos va a permitir gestionar fácilmente peticiones **AJAX**, manipular el DOM en cliente, etc … Esta librería se ha convertido en un framework muy popular, y es la base de para la creación de **pluggins** que nos van a permitir dotar a nuestro sitio web de efecto sorprendentes sin apenas esfuerzo. **http://jquery.com/**
- **modernizr.js**. Esta librería nos permite validar fácilmente si el navegador que esta ejecutando la página web es compatible con HTML5, en en caso de que no sea así proporcional un mecanismo alternativo (*polyfill*). **http://modernizr.com/**. Por ejemplo, si nuestro navegador es compatible con HTML5 interpretará sin problema la etiqueta *VIDEO*, pero en el caso de que estemos navegando con un navegador antiguo deberemos ofrecer al usuario un mecanismo alternativo para el video (un flash por ejemplo).

Nota: El nombre de los archivos puede varias dependiendo de la versión instalada, por ejemplo el archivo jQuery.js se llama jquery-1.7.1.js, ya que corresponde con la versión 1.7.1 de la librería.

8- Web.Config

El archivo web.config es el archivo principal de configuración de ASP.NET. Se trata de un archivo XML donde se define la configuración de la aplicación. Veremos poco a poco el

contenido de este fichero, aunque vamos a ver aquí algunas características generales que es necesario conocer.



Para simplificar el despliegue de las aplicaciones, el archivo de configuración se presenta en diferentes versiones o “sabores”, de forma que podemos modificar el archivo de configuración dependiendo de nuestra configuración de despliegue. Si observamos estos archivos observaremos que contienen transformaciones XML que se aplican sobre el archivo web.config al desplegar la aplicación.

De este modo cuando desplegamos la aplicación en modo *Debug* se aplicarán las transformaciones XML definidas en *Web.Debug.config*.

El archivo de configuración aplica un mecanismo de jerarquía a nivel de los archivos de configuración, en la que un archivo de mayor profundidad dentro de la jerarquía sobrescribe al de menor, en el contexto del recurso solicitado.

9- Favicon.ico

Archivo que representa el icono de la aplicacion web, es agregado automaticamnete al crear el Proyecto plantilla de Visual Studio.

10- Global.asax

Toda aplicación ASP.NET Web Form es una instancia de una clase derivada de System.Web.HttpApplication. Esta clase es el punto de entrada de nuestra aplicación – el *Main* de la aplicación web por decirlo de alguna manera.

11-packages.config

El packages.configarchivo se usa en algunos tipos de proyectos para mantener la lista de paquetes referenciados por el proyecto. Esto permite que NuGet restaure fácilmente las dependencias del proyecto cuando el proyecto se transporte a una máquina diferente, como un servidor de compilación, sin todos esos paquetes.

Si se usa, packages.configgnormalmente se encuentra en una raíz de proyecto. Se crea automáticamente cuando se ejecuta la primera operación NuGet, pero también puede crearse manualmente antes de ejecutar cualquier comando como nuget restore.

Los proyectos que usan PackageReference no usan packages.config.

El esquema es simple: seguir el encabezado XML estándar es un <packages>nodo único que contiene uno o más <package>elementos, uno para cada referencia.

4 Ciclo de vida de los eventos en ASP.NET

4.1PreInit:

Se provoca después que la fase inicial se ha completado y antes de que comience la fase de inicialización.
Utilice este evento para lo siguiente:

- Ver la propiedad IsPostBack para determinar
- si esta es la primera vez en que se está procesando. Las propiedades IsCallback y IsCrossPagePostBack también se han creado en este momento.
- Crear o volver a crear controles dinámicos.
- Establecer una página principal dinámicamente.
- Ajustar la propiedad Theme dinámicamente.
- Leer o establecer valores de propiedad de perfil.

4.2Init:

Se provoca después de que todos los controles se han inicializado y la configuración de los Skin han sido aplicadas. El evento Init de los controles individuales se produce antes del evento Init de la página.
Utilice este evento para leer o inicializar las propiedades del control.

4.3InitComplete:

Lanzado al final de la fase de inicialización de la página. Sólo una operación se lleva a cabo entre los eventos Init e InitComplete.
Utilice este evento para realizar cambios en el view state que usted quiera que se conserven después del postback.

4.4PreLoad:

Se provoca una vez cargado el view state y todos los controles, y después de procesar los datos del postback que esta incluido con la instancia de Request.

4.5Load:

El objeto Page llama al método OnLoad, y luego hace lo mismo de forma recursiva para cada control secundario hasta que la página y todos los controles estén cargados. La carga de los controles individuales se produce después de la carga de los eventos de la página.
Utilice el evento OnLoad para establecer las propiedades de los controles y establecer conexiones de base de datos.

Eventos de los controles:
Aquí se controlan los eventos de los controles, como el onClick de un botón o el evento TextChanged de un TextBox.

4.6LoadComplete:

Lanzado en el final de los eventos de los controles.
Utilice este evento para realizar tareas que requieren que todos los demás controles de la página esten cargados. Jump

4.7PreRender:

Se provoca después que el objeto Page ha creado
todos los controles que se requieren en la página, incluyendo controles secundarios de controles compuestos. (Para ello, el objeto Page llama EnsureChildControls de los controles y de la página.)
Utilice el evento para hacer cambios finales
a los contenidos de la página o en sus controles antes de que comience la fase de Rendering.

4.8PreRenderComplete:

Se provoca después que se establece la propiedad DataSourceID y se llama a su método DataBind

4.9SaveStateComplete:

Se provoca una vez el View State y el control state de la página y todos los controles se han guardado. Cualquier cambio en la página o control en este punto afecta al renderizado, pero los cambios no se recuperarán en el siguiente postback.

4.10 Render:

Este no es un evento, sino que, en esta etapa de procesamiento, el objeto Page llama a este método en cada control. Todos los controles de servidor Web ASP.NET tienen un método Render que escribe el codigo en el navegador.
Si crea un control personalizado, normalmente se invalida este método para generar el marcado del control. Sin embargo, si el control personalizado incorpora sólo estándar ASP.NET de controles de servidor Web y no marcado personalizado, no es necesario reemplazar el metodo Render.

4.11 Unload:

Se lanza para cada control primero, y a continuación para la página.
En los controles, utilizar este evento para hacer la limpieza final para los controles específicos, tales como el cierre de las conexiones de base de datos.
Para la propia página, utilice este evento para realizar el trabajo de limpieza final, como cerrar los archivos abiertos y las conexiones de base de datos.