INGENIERIA INFORMATICA

RESUMEN HADA

HADA RESUMEN

INDICE:

Tema 0 : Plataforma .NET y C#.....2

Tema 1 : Control de Versiones 3-8

Tema 2 : Programación Dirigida por Eventos 9-14

Tema 3: Introducción a las aplicaciones web 15-18

Tema 4: Desarrollo de aplicaciones web con ASP.NET 20-27

Tema 5: Modelo de capas 28

Tema 6: ADO.NET (1/2) 29-30

Tema 7: ADO.NET (2/2)=>Desconectado 31-35

Tema 8: ASPNET 36-39

Tema 9: Bibliotecas 40

Tema 10: Ajax en .net 41-43

Resumen Tema 0: (Plataforma .NET y C#)

¿Qué es .Net?

- es un **conjunto de bibliotecas de desarrollo.**

CLR: (Common Language Runtime)-> como una máquina virtual.

Es la parte de .NET encargada de **ejecutar las aplicaciones** desarrolladas para la plataforma, debido a esto, a las aplicaciones de **.NET** se las conoce como **aplicaciones "manejadas" o aplicaciones de código gestionado**.

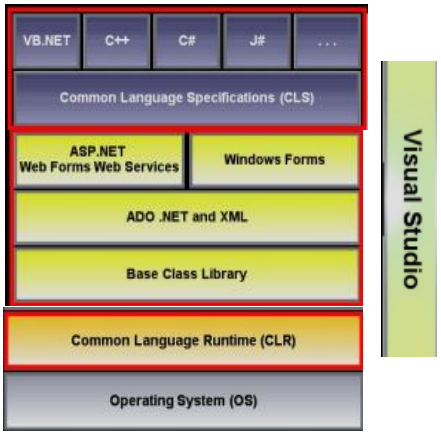
- **Trabaja encima del sistema operativo** para aislar a la plataforma de éste, esto le permite ejecutar aplicaciones **.NET multiplataforma.**
- Garantiza también la **seguridad de los tipos de datos**, no errores por conversión de los tipos de datos en la ejecución de una aplicación .NET**, regulado por Common Type System (CTS), que define los tipos de datos de .NET, y las construcciones de programación de los lenguajes que el CLR puede utilizar de forma adecuada y correcta,** el CTS asegura que los objetos escritos en diferentes lenguajes .NET se pueden interactuar entre sí.
- Posibilidad de **reutilizar** porciones de **código** escritos **en diferentes lenguajes.**
- **Gestiona la vida de los objetos,** se lleva a cabo por el recolector de basura.

CLS – (Common Language Specification): la plataforma .NET no está atada a un determinado lenguaje de programación.

el CLS se asegura de que los lenguajes de .NET puedan interactuar entre sí de manera efectiva al seguir un conjunto común de reglas.

BCL – (Base Class Library):

ofrece bibliotecas/APIs especializadas que pueden ser utilizadas por todos los lenguajes de programación de .NET.



C#

C# es un lenguaje de programación de la familia de C, muy parecido a C++ y Java.

Es un lenguaje orientado a objetos y seguro por tipo. C# soporta los conceptos de encapsulación, herencia y polimorfismo y permite el uso de propiedades y métodos genéricos.

Existen varios tipos de colecciones, las más versátiles emplean genericidad (List, Stack, Queue…). Las más usadas son: **ArrayList, Hastable, Queue (fifo), Stack (lifo).**

Resumen Tema 1: (Control de Versiones)

git: Un sistema de control de versiones **distribuido**, cada desarrollador tiene una copia completa del historial del proyecto en su ordenador local.

Dropbox, Google Drive (no son sistemas de control de versiones), Historial de versiones ⇒ temporal

¿qué nos proporcionan los sistemas de control de versiones (SCV)?

● **Gestionar automáticamente los cambios** que se realizan sobre uno o varios ficheros de un proyecto.

● **Restaurar** cada uno de los ficheros de un proyecto **a un estado anterior** (no solo al inmediatamente anterior).

● **Permitir la colaboración** de diversos programadores en el desarrollo de un proyecto.

SCV Centralizado: **1 único repositorio donde tenemos las versiones** (solo hay una copia maestra del proyecto)

1. **Único punto de fallo:** Si el servidor central falla, el acceso a las versiones de los archivos se pierde temporalmente.
2. **No permite trabajar sin conexión:** Dado que todos los cambios se registran en el servidor central, necesitas una conexión a este para poder enviar (commit) tus cambios.

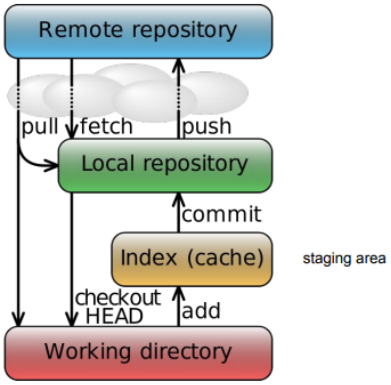
SCV Distribuido: Cada desarrollador obtiene su propio repositorio local con la historia completa del proyecto. Los cambios se guardan localmente y luego se pueden sincronizar con otros repositorios.

1. Puedes trabajar sin conexión (siempre tienes tu repositorio contigo)

Repositorio: donde se guardan todas las versiones de los archivos de un proyecto. En el caso de git se trata de un directorio. Cada desarrollador tiene su propia copia local de este directorio.

Check Out / Clone: para obtener una copia de trabajo desde el repositorio remoto, proporcionando una copia local del repositorio en SCV distribuidos.

Check In / Commit: los cambios se guardan en el repositorio local del usuario, no directamente en el repositorio central.



●**Push**: **traslada los contenidos de la copia local** del repositorio **a la copia maestra del mismo. (repositorio remoto)**

● **Update/Pull/Fetch+Merge/Rebase**: **actualiza nuestra copia local del repositorio a partir de la copia maestra** del mismo, además de actualizar la copia de trabajo con el contenido actual del repositorio local.

● **Conflicto**: Situación que **surge cuando dos desarrolladores hacen un commit con cambios en la misma región del mismo fichero**. El scv lo detecta, pero es el programador el que debe corregirlo.

Git: Historia:

- **Fue creado por Linus Torvalds en abril de 2005 para los desarrolladores de Linux, reemplazando a BitKeeper.**
- Los elementos de Git se dividen en **'plumbing'** (bajo nivel) que funciona como un **sistema de archivos direccionable por contenido**, y **'porcelain'** (alto nivel) que **proporciona herramientas amigables para el usuario**.
- Cuenta con aplicaciones escritas en C y en shell-script. Con el paso del tiempo algunas de estas últimas se han reescrito en C.
- **Git identifica los objetos almacenados usando valores SHA-1**.

INFORMACIÓN:

git status: muestra el estado de los archivos en el directorio de trabajo y del sitio intermedio entre el directorio y el repositorio local **(index/staging area/caché)**

- **untracked:** no incluidos en el repositorio bajo control de versiones
- **committed:** confirmados y almacenados en tu copia local del repositorio
- **modified:** modificados respecto a la copia local del repositorio
- **(un)staged**: (no) preparados para enviar a tu copia local del repositorio

git log: muestra el historial de (commits) en orden cronológico inverso.

-p: **diferencias entre confirmaciones (commits)**

- **Podemos llamarlo con una etiqueta, con el SHA-1, con HEAD~1**

git show: muestra **detalles del último commit** con diferencias detalladas**.**

git diff: muestra las **diferencias entre el directorio de trabajo y el commit más reciente.**(línea por línea)

Etiquetas: Podemos etiquetar confirmaciones (commits) **para marcar puntos importantes**

```
> git tag tagname commit                  //crear etiqueta
> git tag -a -m "mensaje" tagname commit  //etiqueta anotada
> git tag -l                              //lista todas las etiquetas
> git tag -d tagname                      //elimina una etiqueta
> git show tagname
//muestra la información de la etiqueta y el objeto de referencia
```

Descartar cambios:

1. **De mi directorio de trabajo:** cojo la última versión de mi repositorio local
   git checkout archivo:

   cambia el HEAD al último commit de la rama especificada. Esto no afectará a los cambios en tu directorio de trabajo.
2. **Deshace todos los cambios en tu copia local del repositorio y la restaura al estado del último commit.**
   git reset –hard HEAD~N … o SHA-1
3. **El archivo o directorio ya no será considerado para futuros commits.** Este comando es útil cuando has agregado accidentalmente un archivo y quieres dejar de rastrearlo sin eliminarlo de tu sistema de archivos local. **(Para quitarlo cuando hemos hecho un add).git rm --cached nombre-del-archivo-o-directorio**

Cómo organizar un repositorio - ¿Qué incluir?

Incluye en tu repositorio archivos de código, aquellos necesarios para construir y ejecutar la solución, archivos de configuración y scripts de creación de bases de datos. **(sln y .csproj)**

No incluyas archivos generables, binarios, resultados de compilación, la BBDD ni archivos binarios grandes. Evita crear repositorios de gran tamaño para no ralentizar las operaciones de Git. **(.exe, Debug, release…)**

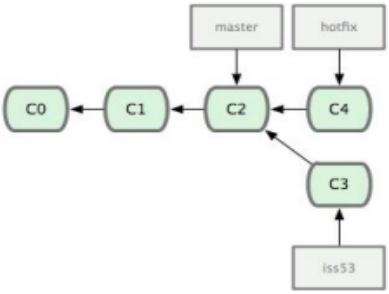
.gitignore: definimos los ficheros y carpetas que queremos NO incluir (y por tanto no rastrear) en el repositorio

Ramas:

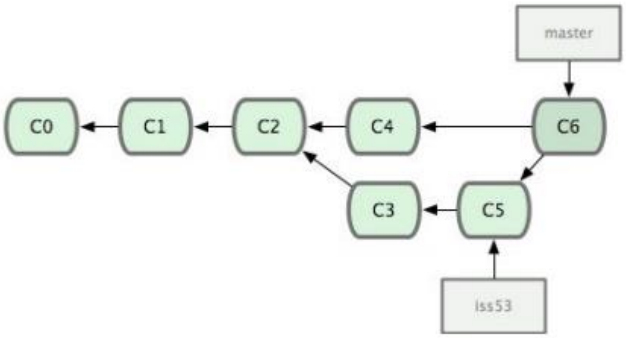
La rama por defecto se llama master. Por cada commit que hagamos, la rama avanzará automáticamente. La rama master **siempre apuntará la última confirmación** (commit) realizado.
Git usa ramas y un puntero especial llamado HEAD para gestionar el desarrollo paralelo.
Con **git branch** creas una nueva rama, que apunta al mismo commit al que apunta la rama actual.
Con **git checkout** puedes saltar entre ellas, cambiando el puntero HEAD a la rama que cambias.

Las modificaciones se registran solo en la rama actual, permitiendo la diferencia de ramas.

G**it merge** fusiona ramas, **avanzando el puntero si es posible o creando un nuevo commit si hay conflictos.** Las ramas no necesarias pueden ser eliminadas. Git Identifica el último commit común entre las dos ramas. Este commit es la base desde la cual Git comenzará a aplicar los cambios.



En este caso, Git puede hacer lo que se conoce como un "avance rápido" (fast-forward), lo que significa que simplemente **mueve el puntero de la rama master hacia adelante, ya que no hay cambios en la master que no estén ya en hotfix**.



Cuando hay cambios en ambas ramas después del último commit común, Git tiene que combinar esos cambios. En estos casos, **Git crea una nueva instantánea que es el resultado de la fusión de las dos ramas, y crea un nuevo commit que apunta a esta instantánea**. Esto se conoce como una "fusión confirmada" y este commit tiene la particularidad de tener más de un commit padre.

Después de que todos tus cambios se han fusionado con la rama principal (en este caso, master**), ya no necesitas la rama en la que estabas trabajando** (en este caso, iss53), por lo que puedes borrarla.

Con rebase esta todo junto (perdemos el historial de como y cuando se hicieron los commits)

Comandos:

git branch [-a] [-r]: para ver las ramas existentes. La opción **-a muestra todas las ramas (locales y remotas)**, mientras que **-r muestra solo las ramas remotas.**

git show-branch: proporciona información sobre las ramas y los commits en tu repositorio.

git checkout [-b] [rama-de-partida]: Este comando se utiliza para cambiar a una rama específica. **La opción -b se utiliza para crear una nueva rama y cambiar a ella en un solo paso.**

gitk --all: gitk es una interfaz gráfica para Git. La **opción --all muestra todas las ramas y commits.**

Repositorios remotos y operaciones con ellos:

git remote add: Agrega un nuevo repositorio remoto a tu repositorio local. Se requiere un nombre y una URL para el repositorio remoto.

git clone: Crea una copia local de un repositorio remoto.

git push: Envia tus cambios a una rama específica de un repositorio remoto.

git checkout -b: Crea una nueva rama local, cambia a ella, y obtiene los cambios de una rama remota específica.

git fetch: Obtiene los últimos cambios del repositorio remoto sin fusionarlos en tu rama actual.

git merge: Fusiona los cambios de una rama a otra.

git pull: Obtiene y fusiona los últimos cambios de una rama específica de un repositorio remoto a tu rama actual. **git pull es una combinación de git fetch y git merge.**

git rebase: Aplica tus cambios sobre los cambios de otra rama.

Stash, Bisect, Herramientas gráficas, Interacción con otros SCVs:

git stash: se usa para guardar temporalmente cambios que no quieres commitear inmediatamente. Si me bajo lo que tengo en el repositorio lo puedo perder.
git bisect: se usa para encontrar el commit que introdujo un bug utilizando búsqueda binaria. Cuando acabo git bistec reset

Las herramientas gráficas como gitk, git gui, git view, gitg, giggle, gource, y plugins para IDEs o editores (Atom, Sublime, VisualStudio, VisualStudioCode, etc.) te proporcionan una interfaz de usuario para interactuar con Git.

Git puede interactuar con otros sistemas de control de versiones (SCVs), como CVS y Subversion.

Ejemplos de uso:

Para crear una rama local que siga los cambios en una remota al hacer pull,
git branch --track ramalocal origin/ramaremota

Para crear una rama que no parta del último commit de otra,
git branch --no-track feature3 HEAD~4

Para ver quién hizo qué commit en un fichero del proyecto,
git blame fichero

Para crear una rama para resolver un bug y luego integrarlo de nuevo en la rama principal, puedes hacer checkout a una nueva rama con git checkout -b fixes, hacer tus cambios, realizar un commit con git commit -a -m "Crashing bug solved.", cambiar de nuevo a la rama master con git checkout master, y finalmente fusionar los cambios con git merge fixes.

Para revertir los cambios en un archivo específico a la última versión bajo control de versiones,
git checkout -- src/main.cs.

Para revertir los cambios en un directorio completo a la penúltima versión de la rama "test",
git checkout test~1 -- src/.

Si has modificado varios archivos y quieres revertir todos los cambios, (hay un conflicto y lo quiero sobrescribir los cambios del repositorio remoto al directorio de trabajo).
git checkout -f o git reset --HARD

Para deshacer un commit que es una mezcla (merge) de varios commits, deshace el efecto de los commits. **git revert mantendrá un registro de los commits que estás revirtiendo, mientras que git reset --hard los eliminará permanentemente.**
git revert HEAD~1 -m 1.

Para obtener un archivo tal y como estaba en una versión determinada del proyecto,
git show HEAD~4:index.html > oldIndex.html
obtiene el contenido del archivo index.html tal como estaba en el cuarto commit antes del commit actual y lo guarda en oldIndex.html en tu directorio de trabajo.
o git checkout HEAD~4 -- index.html. (aquí se guarda en index.html)

Para ver los cambios que ha habido en el repositorio:
git diff, git log --stat, o git whatchanged.

Para saber cuántos commits ha hecho cada miembro del proyecto en la rama actual,
git shortlog -s -n. **En todas las ramas**, git shortlog -s -n --all.
Para corregir el mensaje de explicación del último commit que has hecho,
git commit --amend. Esto abrirá el editor por defecto y te permitirá modificarlo.

Para preparar y confirmar todos tus cambios rastreados git commit -a -m ""

•**Repositorios remotos para Git:** **permiten sincronizar repositorios locales con remotos** y brindan una interfaz web para trabajar con Git.

•**GitHub:** **es una plataforma de desarrollo colaborativo que permite almacenar proyectos utilizando Git**. Los proyectos pueden ser:
públicos (visibles para todos, todo el mundo puede clonarlos)
privados (visibles solo para ti y tus colaboradores seleccionados).

•**Conflictos:** cuando varios desarrolladores **trabajan en el mismo contenido**. Para minimizarlos, cada desarrollador **suele trabajar en una rama separada**. Sin embargo, cuando se producen conflictos, **es responsabilidad del desarrollador resolverlos.**
Los conflictos se pueden producir al inicio de o durante el proceso de combinar ramas (git merge).

•**Resolución de conflictos:** si Git falla al combinar ramas, marcará el archivo como conflictivo e interrumpirá el proceso.

Conflicto I: Falla de Git al inicio del proceso de combinación de ramas (merge).
Este conflicto ocurre cuando Git encuentra **diferencias entre el directorio de trabajo y el área de preparación** (staging area) del proyecto. **(conflictos con cambios locales pendientes**)

```
error: Entry '<fileName>' not uptodate. Cannot merge.
(Changes in working directory)
```

No se pueden combinar las ramas porque esos cambios pendientes podrían ser sobrescritos por los commits que se van a combinar.
Resolver con: **git stash, git checkout, git commit o git reset.**

Conflicto II: Falla de Git durante el proceso de combinación de ramas (merge).

Este conflicto sucede entre la rama local y la rama remota que se está combinando.
Resolver manualmente los archivos conflictivos. Usar:
git merge –abort: saldrá del proceso de combinación y volverá a la rama al estado anterior, como si nada hubiera pasado
git reset.

```
error: Entry '<fileName>' would be overwritten by merge.
```

•**Flujos de trabajo en Git**: **es un protocolo que se debe seguir para trabajar de manera consistente y productiva**. Existen diferentes flujos de trabajo públicos disponibles como:
"Centralized Workflow" (más simple): **todos los cambios en la rama master** similar SCV centralizado, no se necesitan otras ramas.
"Gitflow Workflow" (más usado): este flujo de trabajo implica crear **una rama para cada nueva idea**, añadir commits a medida que se hacen cambios, abrir una "Pull Request" para iniciar una discusión sobre los commits, y finalmente combinar las ramas.

Resumen Tema 2: (Programación Dirigida por Eventos)

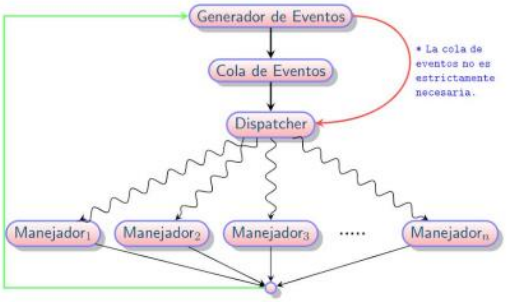
Programación secuencial vs. dirigida por eventos:

En la programación secuencial, el flujo de control es lineal y predecible. El programa se ejecuta de principio a fin, tomando decisiones basadas en la lógica incorporada en el código.

Por otro lado, **la programación dirigida por eventos está basada en la reacción a ciertos eventos o condiciones que pueden ocurrir en cualquier momento**, lo que da al usuario un mayor control sobre el flujo del programa. el programa define una serie de eventos y las acciones que deben realizarse cuando estos eventos ocurren. El programa espera continuamente a que ocurran estos eventos y, una vez detectados, ejecuta el código correspondiente.

Este modelo de programación es común en aplicaciones con una interfaz gráfica de usuario, donde el flujo de control se basa en la interacción del usuario con la interfaz. En este caso, el usuario (generador de eventos) toma el control sobre la aplicación, lo que puede llevar a una secuencia de ejecución de código distinta en cada ejecución del programa.

En la programación dirigida por eventos**, la aplicación inicializa un sistema de eventos, define los eventos que pueden ocurrir, prepara los generadores de eventos y establece el código a ejecutar en respuesta a cada evento**. Esta secuencia de ejecución ininterrumpida se conoce como el **bucle de espera de eventos.**



Una vez producidos son detectados por el **"dispatcher"** o planificador de eventos, el cual se encarga de invocar el código que previamente hemos dicho que debía ejecutarse.

El principio de HollyWood:

El principio de Hollywood es una **estrategia de diseño** comúnmente utilizada en la **programación dirigida por eventos y en la construcción de frameworks**. Este principio puede resumirse como "No nos llame...ya le llamaremos". En otras palabras, **en lugar de que el código del cliente llame a un servicio o a un framework, el framework es quien llama al código del cliente cuando es necesario.**

Callback:

qsort() toma cuatro argumentos:

-base: Un puntero al primer elemento del array a ordenar.

-nmemb: El número de elementos en el array.

-size: El tamaño en bytes de cada elemento en el array.

-compar: Un "callback" que compara dos elementos.

Callback: Es un fragmento de código ejecutable que se pasa como argumento a otro código. Este otro código es quien tiene la responsabilidad de "llamar de vuelta" (execute back) a esta función en el momento adecuado. Es un puntero a una función

Ejemplo de uso de qsort(): En el código de ejemplo, qsort() se utiliza para ordenar un array de enteros. La función int_cmp() se utiliza como "callback" para comparar dos enteros.

```
int int_cmp (const void* pe1, const void* pe2) {
      int e1 = *((const int*) pe1);
      int e2 = *((const int*) pe2);
      return e1-e2;
}
 qsort (v, 4, sizeof(int), int_cmp);
```

Ejecución diferida: Esto es una ventaja significativa de los "callbacks". Se pueden escribir funciones con la misma firma (o signatura) que el "callback" requerido, y luego se pueden pasar estas funciones a qsort(). Esto proporciona una gran flexibilidad, ya que se puede cambiar el comportamiento de qsort() simplemente proporcionando diferentes funciones de comparación.

Delegados:

Los delegados en C# son una característica del lenguaje que permite tratar a los métodos como objetos.

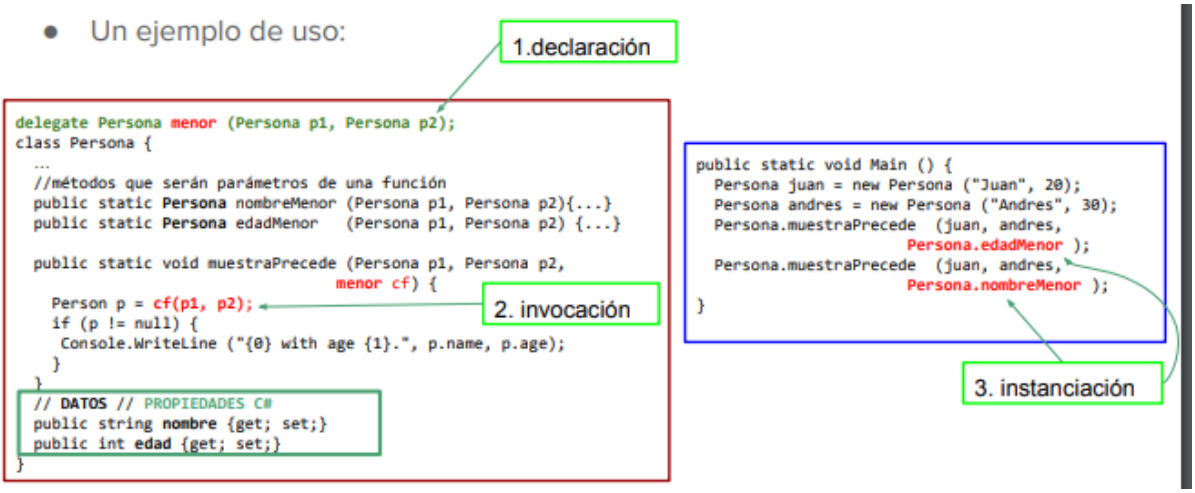
En lugar de simplemente pasar el método como parámetro, primero declaramos una variable del tipo delegado. Un delegado es un tipo de dato que representa una referencia (puntero) a un método con una serie de parámetros y un tipo de resultado.

Sintaxis: public delegate int PerformCalculation(int x, int y); //Necesitamos System.Delegate

Primero se declara un delegado llamado "menor" que representa una función que toma dos "Persona" como argumentos y devuelve una "Persona".

Luego se declara una clase "Persona" con dos métodos estáticos, "nombreMenor" y "edadMenor", que pueden ser utilizados con el delegado "menor". Estos métodos son diseñados para comparar dos personas en diferentes formas, respectivamente, por nombre y por edad.

- Un ejemplo de uso:



El método "muestraPrecede" utiliza el delegado para determinar cuál de las dos personas es "menor" de acuerdo a la función pasada como delegado.

En el método "Main", se crean do s objetos "Persona" llamados "juan" y "andres", y se usan con el método "muestraPrecede" junto con los métodos "edadMenor" y "nombreMenor" como delegados.

Esto muestra la utilidad de los delegados en C#: podemos escribir código que toma funciones como argumentos y las usa en diferentes formas, permitiéndonos hacer que nuestro código sea más flexible y reutilizable.

Propiedades: (para leer, escribir o calcular el valor de un campo privado)

● Las propiedades se pueden usar como si fueran miembros de datos públicos. **Una propiedad es, esencialmente, un campo de una clase que tiene un método "get" y/o un método "set" asociado. Estos métodos son llamados "accessors" (descriptores de acceso).**

El accessor "get" es utilizado para retornar el valor del campo, y el accessor "set" es utilizado para establecer el valor del campo (La palabra clave value se usa para definir el valor que va a asignar el descriptor de acceso set)

● Esto permite acceder fácilmente a los datos a la vez que proporciona la seguridad y la flexibilidad de los métodos.

Un "campo de respaldo" es un campo privado que es utilizado por una propiedad para almacenar su valor. _seconds es un campo de respaldo para la propiedad Hours. Esto permite que la propiedad Hours tenga una lógica adicional en sus accessors, como la conversión entre segundos y horas y la validación del valor de la propiedad.

```
private double _seconds;
public double Hours
{
    get { return _seconds / 3600; }
    set {
       if (value < 0 || value > 24)
```

Las "propiedades implementadas automáticamente": cuando los accessors de una propiedad simplemente obtienen o establecen el valor de un campo sin ninguna lógica adicional. En estos casos, C# proporciona automáticamente el campo de respaldo, por lo que no es necesario declararlo explícitamente.

```
public string Name
{ get; set; }
```

Funciones Lambda: Se definen en el momento que se van a usar.

Las funciones lambda en C# son funciones anónimas (sin nombre) que puedes definir en el lugar donde las necesitas. A veces, es difícil o imposible para el compilador deducir los tipos de datos. Cuando esto pasa, se pueden especificar explícitamente.

Aquí hay algunas formas de definir funciones lambda en C#: // Num. of param. ≠ 1 → use of parenthesis

1. **x => x * x:** devuelve el cuadrado de x.
2. **(x, y) => x == y:** devuelve true si x es igual a y, y false en caso contrario.
3. **(int x, string s) => s.Length > x:** True si la longitud de s es mayor que x, y false en caso contrario.
4. **() => SomeMethod():** Esta es una función lambda sin parámetros que llama a SomeMethod cuando se invoca.

Marco de Prog. dirigida por eventos en C# y .Net:

Un evento representa un cambio de estado en un objeto y puedes asociar código a un evento para que se ejecute cuando este ocurra. **Cada vez que se produzca un evento para un objeto concreto de una clase, se ejecutarán todos los handlers que le hayamos asociado, secuencialmente uno tras otro y no tiene por qué ser en el orden en que se conectaron.**

Un emisor es una clase que genera un evento. En el ejemplo, VideoEncoder es el emisor que genera un even to cuando un video se codifica.

```
public void Encode(Video video)
{
    OnVideoEncoded(video);
}
```

Un receptor es una clase que responde a un evento. En el ejemplo, MailService y MessageService son receptores que responden al evento generado por VideoEncoder.

Para que los emisores y receptores puedan comunicarse a través de eventos, necesitan un event handler. Un event handler es un método con una firma específica que se invoca cuando ocurre un evento. Por ejemplo:

```
public void OnVideoEncoded(object sender, EventArgs a)
```

Cuando VideoEncoder genera un evento, invoca a OnVideoEncoded en cada uno de los receptores asociados a este evento. Cada receptor implementa este método de acuerdo a su lógica interna.

Clase emisora

1. **Definición del delegado: El delegado actúa como un contrato entre el emisor y el receptor, especificando la firma del método que el emisor puede llamar cuando ocurra el evento. No hace falta ya existe**

   ```
   public delegate void VideoEncodedEventHandler(object sender, EventArgs args)
   ```

2. **Definición del evento: El evento videoEncoded se define basándose en el delegado VideoEncodedEventHandler.**

   ```
   public event VideoEncodedEventHandler videoEncoded;
   ```

3. **Lanzamiento del evento: Cuando ocurre el evento, el método OnVideoEncoded invoca al delegado videoEncoded, pasándole la información necesaria.**

```
videoEncoded(this, EventArgs.Empty);
```

4. **También tenemos que definir los argumentos a pasar en el evento, esto se hace en una nueva clase que hereda de EventArgs y que se llama como el argumento:**

```
// Aquí está la magia, esto define un delegado y un evento en una línea.
public event EventHandler<VideoEventArgs> videoEncoded;
```

Event Handler representa **la signatura de métodos/funciones que pueden conectarse con ese evento + parámetros del evento**

```
videoEncoded(this, new VideoEventArgs(){Video=video;});
```

```
public class VideoEventArgs: EventArgs
{
    public Video v { get; set; }
}
```

Clase Receptora

1. **Definir método/s manejador del evento**

```
public class MailService
{
    public void OnVideoEncoded(object source, VideoEventArgs e)
    {
        Console.WriteLine("MailService: enviando email");
    }
}

public class MessageService
{
    public void OnVideoEncoded(object source, VideoEventArgs e)
    {
        Console.WriteLine("MessageService: enviando sms");
    }
}
```

2. **Conectarlos (Asociar el evento con el manejador/handler)**
   -En el constructor de la clase manejadora (receptora)
   - En el main

```
public class MailService
{
    public MailService(VideoEncoder v)
    {
        v.videoEncoded += OnVideoEncoded;
    }

    public void OnVideoEncoded(object source, VideoEventArgs e)
    {
        Console.WriteLine("MailService: enviando email");
    }
}

public class MessageService
{
    public MessageService(VideoEncoder v)
    {
        v.videoEncoded += OnVideoEncoded;
    }

    public void OnVideoEncoded(object source, VideoEventArgs e)
    {
        Console.WriteLine("MessageService: enviando sms");
    }
}
```

```
static void Main(string[] args)
{
    Video video = new Video();
    VideoEncoder v = new VideoEncoder(); // publisher

    MailService ms = new MailService(); // subscriber
    MessageService mg = new MessageService(); // subscriber

    v.videoEncoded += ms.OnVideoEncoded; // Se subscriben al evento
    v.videoEncoded += mg.OnVideoEncoded; // Se subscriben al evento

    v.Encode(video); // Este método invocará el evento
}
```

//LANZA EL EVENTO

Resumen Tema 3: (Introducción a las aplicaciones web)

Arquitectura cliente/servidor: típica de desarrollo de aplicaciones

Es un modelo de diseño software en que los servidores comparten recursos o servicios, y los clientes solicitan dichos recursos.

• El cliente solicita y el servidor responde.

• Es más ventajoso para aplicaciones distribuidas a través de una red de ordenadores.

- **Cliente:**
  - El software de la parte cliente se conoce como **front-end**
  - Inicia la petición
  - Espera y recibe la respuesta del servidor
  - **Se puede conectar a un grupo de servidores de manera simultánea**
  - Suele ofrecer un interfaz gráfico al usuario final
- **Servidor:**
  - El software de la parte del servidor se conoce como **back-end • Pasivo**
  - Espera las peticiones de los clientes
  - Cuando recibe una petición, la procesa y devuelve la respuesta
  - Normalmente **acepta un gran número de conexiones simultáneas de los clientes**

Ventajas

+ **Distribución de aplicaciones:** concurrencia de procesos
+ **Estandarización**. Los recursos están publicados y otras aplicaciones pueden acceder a ellos
+ **Portabilidad**
+ **Escalabilidad**

Desventajas

- Aumenta la comunicación. Congestión tráfico de red
- **Falta de robustez.** Caídas del servidor

Arquitectura de n-capas:

• No implica separación física en distintos ordenadores de la red (una aplicación de 3 capas puede existir en un único ordenador).

- Ventajas
  - Permitir modificar una capa sin tener que modificar toda la aplicación.
  - Simplificación de la administración de los sistemas
  - Disponibilidad inmediata de cambios
  - Posibilidad de balanceo de carga de trabajo entre ordenadores

Una arquitectura de 3 capas está divida en:

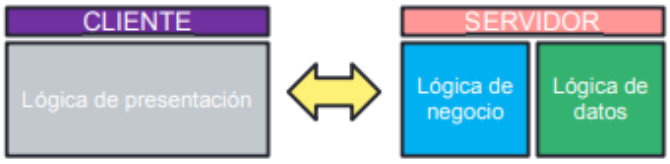
• **Interfaz de usuario (o lógica de presentación), componentes que interactúan con el usuario final. Interfaz gráfico (GUI) o basado en texto**.

• **Lógica de negocio**, contienen las reglas de negocio **(funcionalidades)** de nuestra aplicación. Corresponde con el núcleo de la aplicación.

• **Persistencia (o lógica) de datos**, permite el **acceso y almacenamiento de los datos**

Modelos de distribución :

Presentación distribuida: la interfaz de usuario y las funciones de validación de entrada se procesan en el dispositivo del usuario, mientras que el procesamiento de datos y la lógica empresarial se llevan a cabo en el servidor.



Aplicación distribuida: presenta una flexibilidad máxima, con la lógica de presentación, la lógica de negocio entre el cliente y el servidor y la lógica de datos en el servidor. Esto puede ayudar a equilibrar la carga y mejorar el rendimiento de la aplicación.

Datos distribuidos: implica una carga máxima en el cliente y requiere un mayor ancho de banda. La lógica de presentación y de negocio se maneja en el servidor, mientras que la lógica de datos se procesa en el servidor.

Aplicaciones web:

se ejecuta en el entorno web basada en la arquitectura cliente/servidor. La **comunicación** entre el cliente (navegador web) y el servidor (servidor web) se realiza **mediante el protocolo HTTP.**

Cliente:

Gestiona las peticiones del usuario y la recepción del contenido de las páginas que provienen del servidor web. Puede **interpretar los documentos HTML y otros recursos.** Las tecnologías utilizadas en el cliente pueden incluir **HTML, CSS, JavaScript, DHTML, VBScript, ActiveX, Applets Java, plug-ins, XML.**

- **HTML: Lenguaje de programación para definir el contenido de una página web.**
  • W3C validator(Revisa el código fuente, busca errores de sintaxis, devuelve el resultado de la prueba de validación).

- **CSS: Lenguaje de diseño gráfico que define la presentación de un documento estructurado en HTML.**

- **JavaScript: Lenguaje de programación embebido dentro de una página web para crear contenido e interactividad dinámica.**(Incorpora el Document Object model (DOM), que se utilizada para interactuar con la página web)

Servidor:

Espera las peticiones de los clientes y proporciona recursos como páginas estáticas HTML, recursos multimedia, y programas que generan páginas web dinámicas. Las tecnologías utilizadas en el servidor pueden incluir **ASP/ASP.NET, JSP, PHP, CGI, ColdFusion, Servlets.**

enviar el resultado HTML al cliente, se puede mezclar con el código HTML (PHP, ASP)

- **PHP:** (Invisible al cliente) **Lenguaje de Script en el servidor para desarrollo web de contenido dinámico.** Se suele combinar con el sistema gestor de base de datos MySQL.

- **ASP: Lenguaje de Script, con un intérprete (no compilado).** Tecnología de Microsoft en el lado del servidor para crear páginas web dinámicas. Utiliza el servidor IIS. (no limitado al servicio web)

- **ASP.NET:** Plataforma para construir aplicaciones Web, forma parte del .NET Framework y admite múltiples lenguajes.
  Para desarrollar aplicaciones ASP.NET necesitamos: • Editor o entorno de desarrollo (Microsoft Visual Studio) • .NET Framework instalado • Un servidor web con IIS

Servidores web:

Internet Information Server (IIS):

Es un servidor web/ conjunto de servicios de Microsoft. Es una plataforma web unificada que integra:

• el servidor web (HTTP/HTTPS), • ASP.NET, y • otros servicios como FTP, SMTP, NNTP

• También puede incluir PHP o Perl

Ventajas
+ Ejecución de aplicaciones web en ASP, ASP.NET, y PHP en un mismo servidor
+ Confiable, seguro y fácil de utilizar
+ Posibilidad de agregar o eliminar componentes IIS integrados e incluso reemplazarlos por módulos personalizados
+ Aumenta la velocidad del sitio web mediante el almacenamiento en caché dinámico integrado y mejora de la compresión
+ Soporte técnico por parte de Microsoft

Desventajas
- No se recomienda su uso para desplegar aplicaciones en PHP, Python, Perl o Ruby (que se ejecutan de forma óptima en Linux y UNIX)
- Su licencia no es gratuita
- No es multiplataforma (sólo Windows)
- Código fuente propietario

Apache:

Es un servidor web de código abierto multiplataforma, el más utilizado en el mercado del hosting.

Ventajas

+ Software de código abierto (instalación y configuración adaptable mediante módulos)
+ Sin coste de licencia
+ Buen soporte debido a que es el más utilizado por lo que muchos programadores contribuyen con mejoras
+ Multiplataforma (Windows, Linux y MacOS)
+ Buena interacción con PHP y MYSQL

Desventajas

- No posee interfaz gráfica
- No contiene una configuración estándar. Puede ser adaptada a cada aplicación
- No hay soporte técnico
- Difícil de administrar y configurar
- Pocas actualizaciones

Metodología de diseño: (aplicación web)

1. **Análisis de requisitos: Analizar todas las solicitudes y objetivos del proyecto.** Es un punto crítico para entender lo que se necesita lograr.
2. **Elección de las tecnologías web a usar: HTML, CSS, ASP, ASP.NET, PHP, etc.**
3. **Elección del gestor de base de datos:** como SQL Server, MySQL, MariaDB…
4. **Arquitectura de la aplicación**: **Identificar la estructura del sistema**, incluyendo la relación cliente-servidor y cómo se manejarán las peticiones y respuestas.
5. **Diseño de estructura y mapa de navegación web:** Definir las páginas web que se van a utilizar y su navegación en el sitio web.
6. **Creación del contenido de las páginas:** Implementar las páginas estáticas de la web.
7. **Diseño gráfico:** mediante CSS para definir la apariencia y estilo del sitio web.
8. **Generación de scripts para controles en el cliente:** Implementar el código Javascript para agregar interactividad y controlar la interfaz de usuario.
9. **Diseño y desarrollo de páginas dinámicas**: Crear páginas que cambien o se actualicen en función de las interacciones del usuario o de los datos de la base de datos.
10. **Validación y pruebas:** cumplir con los requisitos definidos en el paso 1.
11. **Despliegue de la solución:** Lanzar el sitio web en un entorno de producción, lo que puede incluir la configuración del servidor, la carga de los archivos del sitio web y la configuración de la base de datos.

1. **¿Qué son las páginas con extensión ?htm? ¿Es lo mismo ?html que .htm?**
   La única diferencia entre las extensiones .htm y .html es el número de caracteres

2. **¿Cuál es el lenguaje estándar para aplicar estilos de presentación a nuestras páginas web?** (CSS)

3. **¿En qué lugar se ejecuta el código JavaScript?** se ejecuta en el cliente

Resumen Tema 4: (Programación Dirigida por Eventos)

HTML (Hypertext Markup Language)

Es el lenguaje estándar para crear páginas web. Es un lenguaje de marcado, lo que significa que utiliza etiquetas para definir elementos y estructurar el contenido en la página web.

Los elementos HTML son los componentes fundamentales de una página HTML, tienen dos propiedades básicas: **atributos (entre comillas) y contenido**. Un elemento HTML típicamente **consiste en una etiqueta de apertura y una de cierre**, con atributos definidos dentro de la etiqueta de apertura y contenido situado entre las dos etiquetas.

Una página HTML típica tiene una estructura específica que incluye etiquetas **<HTML>, <HEAD>, y <BODY>.**

- **<HTML>** delimitan el documento HTML.
- **<HEAD>** delimitan el encabezado del documento HTML y suelen incluir metadatos, scripts y estilos CSS. (Meta: información de interés como autor, descripción, palabras clave).=>**SEO**
- **<BODY>** delimitan el cuerpo del documento HTML, que es donde se incluyen los contenidos visibles del documento.

HTML ofrece desde etiquetas para definir encabezados, párrafos y bloques de texto, hasta etiquetas para crear listas, imágenes, tablas y formularios.

Las listas en HTML pueden ser:

- no ordenadas (con la etiqueta <ul>)
- ordenadas (con la etiqueta <ol>).
- Los elementos de la lista se definen con la etiqueta <li>. **Las listas ordenadas pueden modificarse para usar diferentes tipos de numeración, como letras o números romanos, mediante el atributo type** en la etiqueta <ol>. También es **posible comenzar la numeración de la lista en un número específico usando el atributo start.**

Enlace: El destino del enlace puede ser a :

• Un lugar de la página en curso • Otra página del sitio web • Algún lugar de otra página del sitio web • Una página de otro sitio existente en la web • Una dirección de correo electrónico • Un archivo para descargas. En otra página poner https://

Etiquetas de organización (permiten formar bloques, que describen mejor las partes de un documento)

Todas las webs están formadas por:

• Encabezado de página

• Herramientas de navegación

• Contenido

• Zona anexa de elementos asociados al contenido (publicidad)

• Pie de página

HTML proporciona varias etiquetas para organizar y estructurar el contenido de una página web. Entre ellas se encuentran las siguientes:

<header>: Agrupa los elementos del encabezado de la página.

<nav>: Indica los elementos del menú de navegación.

<aside>: Indica elementos anexos al contenido principal.

<figure>: Permite incorporar una sección de imagen.

<figcaption>: Contiene el texto explicativo de una imagen asociada.

Formularios:

Los formularios HTML **permiten recabar información del usuario.** Se definen con la etiqueta **<form>** y **la información introducida por el usuario se envía a una página (normalmente al servidor) después de pulsar un botón.**

Los formularios pueden incluir varios tipos de campos**, como:**

- **campos de texto** <input type= "text>
- **listas desplegables<select>/<option> => formar elementos de la lista,**
- **botones de selección única <input type="radio" y múltiple, botones<input type="checkbox">**

de envío y cancelación, áreas de texto, campos ocultos, campos de contraseña, y campos para la transferencia de archivos.

Los atributos de la etiqueta form incluyen 'name' para asignar un nombre al formulario, 'action' para indicar la acción que debe realizar el formulario, 'enctype' para especificar el formato de los datos del formulario, y 'method' para indicar cómo se deben enviar los datos.

Dos opciones para enviar la información (method):

• **GET. Los datos se envían (caracteres ASCII) dentro de la propia URL, unidos por &**

(Para enviar poco texto, datos que no requieran de seguridad, sencillez)

• **POST. Los datos no se envían en la URL y son invisibles para el usuario**

(Para enviar grandes campos de texto, imágenes, URL más legibles, seguridad de información)

Aplicaciones Web ASP.NET:

Son una combinación de archivos, páginas, manejadores, módulos y código ejecutable que pueden ser invocados desde un directorio virtual. Estas aplicaciones se dividen en varias páginas web y comparten un conjunto común de recursos y opciones de configuración. **Cada aplicación tiene su propio conjunto de caché y datos de estado de sesión.**

Las aplicaciones web ASP.NET **se almacenan en un directorio virtual.** Un directorio virtual **es un recurso compartido identificado por un alias que representa la ubicación física en el servidor.** Este directorio puede ser creado y administrado a través de Internet Information Server (IIS).

Un Sitio Web es un conjunto de páginas web independientes y es adecuado para páginas web sencillas, mientras que un **Proyecto Web** es un conjunto de páginas web vinculadas con un archivo de proyecto y es adecuado para aplicaciones más avanzadas.

ASP.NET permite el uso de dos enfoques para el desarrollo de código:

- **Code-inline** implica la colocación de etiquetas declarativas y **código en el mismo archivo.**
- **Code-behind** implica la colocación de código para **manejar eventos en un archivo físico separado de la página que contiene los controles del servidor y las etiquetas**. El último enfoque es beneficioso en entornos de equipo donde los diseñadores trabajan en la interfaz de usuario y los desarrolladores trabajan en el código. •Extensión .aspx • Extensión .cs, .vb … (code-behind)

Finalmente, Visual Studio proporciona su propio servidor de desarrollo, lo que significa que no necesitamos tener instalado IIS para realizar pruebas en nuestro ordenador. Sin embargo, IIS sería necesario para desplegar la aplicación en un servidor.

Formulario ASP.NET: capacidad de mantener el estado de los controles entre solicitudes (gracias al estado de vista), la validación del lado del servidor, y la capacidad de trabajar con eventos de control en el código del lado del servidor (gracias al modelo de eventos de postback).

Formularios Web (Web Forms) en ASP.NET son una técnica para el Desarrollo Rápido de Aplicaciones (RAD). Se desarrollan en un servidor web y los usuarios interactúan con ellos a través de un navegador. Son orientados a objetos, eventos y gestión de estado.

Todos los controles de servidor en Web Forms deben estar dentro de la etiqueta <form> con el atributo runat="server", que indica que el formulario se procesa en el servidor, y pueden ser accedidos por scripts en el servidor •Una página .aspx debe contener un único control.

Controles de servidor

Tienen propiedades que pueden ser establecidas:

- **declarativamente** (en la etiqueta)
- **mediante programación** (en el código).

Junto con la página, tienen eventos que los desarrolladores pueden manejar

• para ejecutar acciones específicas durante la ejecución de la página

• o **en respuesta a una acción del lado del cliente que envía la página al servidor**.

Los controles se indican con el prefijo asp: y el atributo runat="server".

Hay varios tipos de controles ASP.NET:

2. **Controles HTML (no ASP.NET)**
3. **Controles estándar**
4. **Controles de validación**
5. **Controles de login**
6. **Controles de navegación**
7. **Controles Webparts**
8. **Controles de datos**
9. **Controles de usuario**

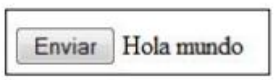
Los eventos ya están definidos, yo tengo que definir los manejadores, la página ASPX, cómo se configura el control en la página, la página ASPX.CS, cómo se manejan los eventos del control en el código del servidor.

Página aspx

```
<form id="form1" runat="server">
<asp:Button id="BotonEnviar" Text="Enviar" runat="server" OnClick="WriteText" />
<asp:Label id="Label1" runat="server" />
</form>
```

Página aspx.cs

```
protected void WriteText(object sender, EventArgs e)
{
    Label1.Text = "Hola mundo";
}
```



Páginas maestras:

Son una excelente forma de garantizar la coherencia y la reutilización de código en una aplicación web.

Una página maestra **es una plantilla base para otras páginas de la aplicación. Permite definir un diseño y comportamiento comunes para todas las páginas que la utilizan.**

Las páginas de contenido son las que muestran el contenido específico de cada página, y se combinan con la página maestra para mostrar una salida que combine ambas.

Algunas ventajas de usar páginas maestras incluyen la **facilidad para hacer cambios de diseño en todas las páginas a la vez y la reutilización de la interfaz de usuario.**

Una página maestra es un archivo **ASP.NET con extensión .master** y se identifica con la directiva **@ Master reemplaza a @Page que se usa en las páginas ordinarias**. Incluye controles **ContentPlaceHolder que definen áreas reemplazables por contenido específico en las páginas de contenido.**

El **<@ Page %> especifica los atributos específicos de página utilizados por el motor ASP.NET al analizar y compilar la página.** Esto **incluye su archivo de página maestra, la ubicación de su archivo de código y su título, entre otros datos.**

En tiempo de ejecución, IIS controla las páginas maestras**: cuando un usuario solicita una página de contenido, se lee la directiva @ Page. Si la directiva hace referencia a una página maestra, también se lee la página maestra, esta se combina con la página maestra, llenando los controles ContentPlaceHolder con su contenido específico, y se muestra la página combinada al usuario.**

Desde la perspectiva del usuario, la combinación de la página maestra con la página de contenido se ve como una única página.

Eventos de los controles del servidor:

Modelo de eventos en ASP.NET:

En páginas web ASP.NET, **los eventos asociados a controles de servidor son generados en el cliente (navegador), pero son manejados por la página ASP.NET en el servidor web. La información del evento se captura en el cliente y se envía un mensaje de evento al servidor mediante una petición HTTP.**

Enlace de eventos a métodos:

Un **evento** es un mensaje que un objeto envía cuando ocurre una **acción**, como un clic en un botón. **Este mensaje debe traducirse en una llamada a un método del código**. El enlace entre el mensaje del evento y un método específico (es decir, un controlador de eventos) se realiza utilizando un **delegado de eventos.**

Eventos y delegados:

El objeto que dispara el evento se conoce como **el remitente del evento**, y el objeto que captura y responde al evento se denomina **receptor del evento.**

.NET Framework define un **tipo especial (Delegate)** que **proporciona la funcionalidad de un puntero a una función, actuando como un intermediario entre el remitente y el receptor del evento.**

Manejadores de eventos:

Los manejadores de **eventos deben coincidir con el prototipo del delegado EventHandler**. **Tienen dos parámetros: el objeto que dispara el evento** (Object sender) **y los argumentos del evento que contienen información específica sobre el evento** (EventArgs o tipo derivado)**.**

Asociación del manejador al control:

El método del controlador del evento puede asociarse al control de varias maneras: escribirlo manualmente en el código, hacer doble clic en un control en la vista de diseño (evento por defecto), o seleccionar el evento de la ventana de propiedades del control.

Eventos Postback vs No-Postback: •Una página se carga después de cada petición (postback)

Los eventos Postback hacen que **la información del formulario se envíe al servidor inmediatamente,** mientras que los eventos **No-Postback** o "cached" **guardan la información hasta el próximo evento Postback.**

- Button, Link Button y Image Button causan eventos **postback.**

- TextBox, DropDownList, ListBox, RadioButton y CheckBox, proveen eventos **cached.**
  - Sin embargo podemos sobrecargar este comportamiento en los controles para poder realizar eventos postback, cambiando la propiedad *AutoPostback* a true**.**

Eventos de página:

Las páginas ASP.NET **disparan eventos del ciclo de vida como Init**, **Load**, **PreRender**... Estos eventos se pueden enlazar a métodos utilizando la convención de **nomenclatura Page_nombreDelEvento.**

Propiedad IsPostBack:

La propiedad Page.IsPostBack **permite ejecutar código** de manera condicional **dependiendo de si la página ha sido enviada previamente o no. Se utiliza en el método Page_Load para diferenciar la carga inicial de la página de las cargas subsecuentes debido a eventos Postback.**

```
protected void Page_Load(object sender, EventArgs e)
{
   if (IsPostBack) {
      Response.Write("<br>Page has been posted back.");
   }
}
```

Importante: Los métodos de control de eventos de página no requieren ningún argumento. Estos eventos (como Load) pueden aceptar los dos argumentos estándar, pero en estos argumentos no se pasa ningún valor.

Navegación entre formularios:

Navegación tradicional:

En HTML se utiliza el elemento <a> con el atributo href para la **navegación**. En ASP.NET, se puede utilizar un control **HyperLink** con la propiedad **NavigateUrl** para el mismo propósito.

Diferentes formas de navegar:

En ASP.NET, se pueden usar los controles **HyperLink** y **LinkButton** para navegar a otras páginas. También se puede usar el método **Response.Redirect** para navegar a otra página mediante código.

Paso de parámetros a otra página:

Se pueden pasar parámetros a otra página **a través de la URL** utilizando la colección **QueryString**. Esta colección **permite incluir pares de nombre-valor** en la URL. Sin embargo, hay algunas **limitaciones** con QueryString**: los caracteres utilizados deben ser válidos en una URL, la información es visible para los usuarios en la barra del navegador, los usuarios pueden modificar la información provocando errores inesperados, y muchos navegadores imponen un límite en la longitud de la URL.**

Se pueden **codificar los datos que se pasan** en la URL para solucionar el problema de los caracteres especiales utilizando los métodos de la clase **HttpServerUtility**.(reemplaza los caracteres especiales por secuencias de escape)

Para **recuperar los datos** pasados a una página, se utiliza el objeto **Request**, que proporciona información sobre la petición HTTP del cliente. La información pasada a través de **QueryString** se puede recuperar mediante **Request.QueryString.**

Aplicación web - Navegación:

Por ejemplo, en el evento BtnEnviar_Click asociado al botón "Enviar datos", se puede usar Response.Redirect para navegar a la nueva página pasando los parámetros en la URL.

En la nueva página, se puede utilizar el evento Page_Load para cargar los datos recibidos y mostrarlos.

CSS

CSS permite definir la presentación de un documento web y proporciona una **estética coherente a todas las páginas de un sitio web**.

Las hojas de estilo pueden ser de tres tipos: **externas(**INTRODUCIR**:** <link rel="Stylesheet" type="text/css" href="~/hoja.css" />)**, internas**(dentro del head) **o en línea.**

Con CSS, puedes **utilizar selectores para aplicar estilos a elementos específicos de tu documento. Los selectores pueden ser de:**

- **tipo** (El estilo se aplica a todos los elementos del mismo tipo)**,** `p{ font-size: 150%;}`
- **clase (**Se utiliza cuando asignamos a cada elemento una clase determinada) `.pageinfo`
- **id.**(permite aplicar una hoja de estilo, aunque **sólo se podrá invocar una vez en el documento**) `<p id="textoazul"> Texto azul </p>` `#textoazul {color: blue;`

El uso de CSS para la maquetación web ha reemplazado el uso de tablas HTML, proporcionando una mayor flexibilidad y mejor rendimiento. Los elementos div/capas/layouts en CSS permiten la creación de "contenedores" o bloques en los que se pueden insertar diferentes elementos.

Resumen Tema 5: (Modelo de capas)

Una arquitectura de capas divide una aplicación en divisiones lógicas desarrolladas y mantenidas como módulos independientes, permitiendo una **mayor flexibilidad e independencia**.(cada capa puede ser sustituida en cualquier momento sin afectar a las otras).

Una arquitectura de 3 capas está dividida en:

- **interfaz de usuario** (interactúa con el usuario final),
- **lógica de negocio** (contiene las reglas de negocio de la aplicación)
- **persistencia** (contiene el acceso y almacenamiento de los datos).

La **capa de presentación maneja la entrada y salida básica del usuario**, proporcionando la interfaz gráfica de usuario, (interacción usuario).

Las **entidades de negocio (EN)** son componentes que **representan entidades de negocio del mundo** real, como un producto o un pedido. Estos componentes **contienen la información de una clase de dominio con sus atributos, operaciones y restricciones**. (Tienen asociado un CAD)

Los **componentes de acceso a datos (CAD) encapsulan la tecnología de acceso a datos y la base de datos (BD) al resto de la aplicación**. Proveen métodos para realizar tareas sobre la BD, como crear, leer, actualizar y borrar registros (CRUD).

Los CAD también pueden contener métodos que realizan algún filtro o búsqueda específica.

Intenta definir todos los métodos que devuelven un tipo concreto de Entidad de Negocio en un solo CAD.(Si estás recuperando todos los clientes que han pedido un específico producto, implementa la función en ClienteCAD ObtenerClientesPorProducto).

Los CADs también pueden realizar otras tareas en su implementación, como **controlar la seguridad y la autorización, realizar la paginación de datos, gestionar transacciones de entidades complejas y llamar a procedimientos almacenados.**

Dividiremos el código en tres capas o componentes:

a. Capa de interfaz de usuario.
b. Capa de lógica de negocio o *Entidad de Negocio* (**EN**).
   - Se le asocia un **CAD** mediante el cual esta **EN** puede almacenarse/modificarse/recuperarse... en la bbdd con la que trabajemos.
c. Capa de persistencia o *Componente de Acceso a Datos* (**CAD**).
   - Los *CAD* implementan la lógica de comunicación con la bbdd, la cual es bidireccional entre las **EN** y la bbdd.
   - Las operaciones habituales que proporciona un **CAD** son las de *creación*, *lectura*, *actualización* y *borrado* de registros de la bbdd.

Resumen Tema 6: (ADO.NET)

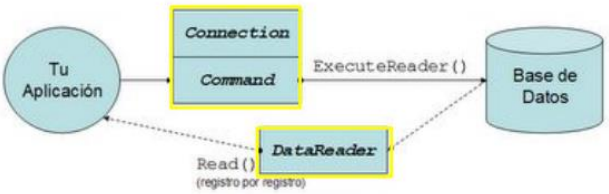
ADO.NET es la tecnología que se utiliza en las aplicaciones asp.net para la comunicación con bases de datos. Está optimizada para aplicaciones distribuidas, como las aplicaciones web, y se basa en XML.

Existen dos enfoques principales :

- **Entorno conectado, los usuarios están constantemente conectados a una fuente de datos (BD).** Este enfoque es más fácil de mantener y controlar la concurrencia, y los datos suelen estar más actualizados. Sin embargo, requiere una **conexión de red constante** y puede tener **limitaciones de escalabilidad**. **El proceso de trabajo incluye conectar a una base de datos, solicitar datos, devolver y transmitir actualizaciones, y finalmente cerrar la conexión.**
- **Entorno desconectado, los datos se pueden modificar de forma independiente y los cambios se escriben posteriormente en la base de datos**. Este enfoque mejora la escalabilidad y el rendimiento de las aplicaciones al minimizar el tiempo de utilización de las conexiones. Sin embargo**, los datos pueden no estar siempre actualizados y pueden surgir conflictos de cambios que necesitan resolución**. **El proceso de trabajo es similar al del entorno conectado, pero la conexión a la base de datos solo se necesita durante la solicitud de datos y la transmisión de actualizaciones.**

Modo conectado:

En este enfoque, se utilizan principalmente tres tipos de objetos:



1. **Connection:** Se usan para **establecer las conexiones** con el proveedor de datos adecuado **usando el método Open.**
2. **Command**: **Sirven para ejecutar sentencias SQL** y procedimientos almacenados.
3. **DataReader: Proporciona un flujo constante de datos.** Ofrece un cursor de solo lectura que avanza por los registros solo hacia adelante. **Mantienen una conexión activa con el origen de datos, pero no permiten realizar cambios.**

Para **recuperar los datos**, necesitas una sentencia SQL, un objeto Command para ejecutar la sentencia SQL y un objeto DataReader para capturar los registros recuperados(cada fila devuelta por la consulta se puede acceder utilizando el objeto DataReader.)**. Los objetos Command representan las sentencias SQL**.

```
SqlCommand com=
new SqlCommand("Select * from clientes",c);
```

Una vez definida la sentencia SQL y la conexión disponible, se ejecuta el comando, **(se utiliza el método ExecuteReader del objeto Command).**

- `SqlDataReader dr= com.ExecuteReader();`

ExecuteReader: Ejecuta un comando y devuelve un comando que implementa DataReader (Permite iterar a partir de los registros recibidos)

Los objetos DataReader se utilizan para acceder a los valores de la fila recuperada. **Para leer la siguiente fila**, se usa nuevamente el método **Read**. **Si devuelve false, significa que hemos intentado leer después del final del conjunto de resultados.**

- `dr.Read()`

```
while (dr.Read())
{
    lista.Add(dr["usuario"].ToString());
}
```

MyDataReader["Email"].

Este código devuelve el valor de la columna Email para la fila actual que está siendo leída por el DataReader. (sensibles a mayúsculas y minúsculas)

IMPORTANTE: cuando terminemos de dr.Read cerrar el DataReader: • `dr.Close();`

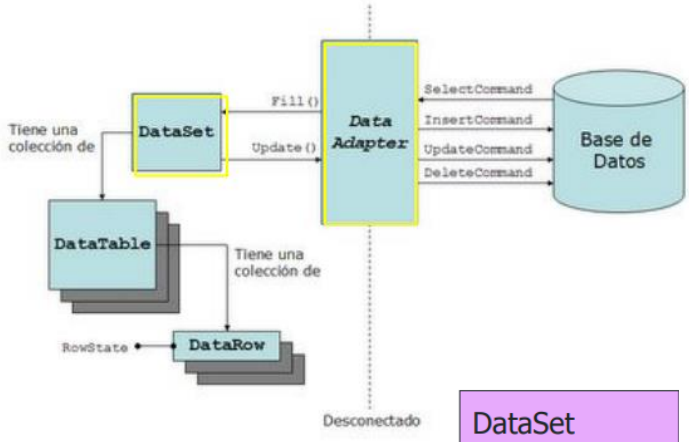
Y connection: • `c.Close();` (Connection en el finally)

Modificación de datos: Insert, Update, Delete

• **Método ExecuteNonQuery**: obtiene el número de registros afectados.

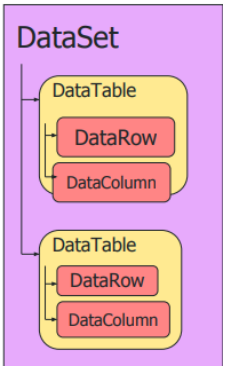
Resumen Tema 7: (ADO.NET)

Modo desconectado:



se utilizan principalmente cuatro tipos de objetos:

1. **Objeto Connection:** Este objeto establece la conexión con la base de datos. En el caso de SQL Server, usarías un objeto SqlConnection. (método Open).



2. **Objeto DataSet: Este objeto representa mi base de datos en memoria que no requiere una conexión continua**. Un DataSet **puede contener varias DataTables,** cada una de las cuales puede representar una tabla de la base de datos. Las DataTables, a su vez, contienen objetos **DataRow**, que representan las filas individuales de datos (accedemos usando el nombre de campo o un índice),
   y objetos **DataColumn**, que representan información sobre la columna: tipos de datos, valor predeterminado…)
   Podemos trabajar con una BD que es copia de las partes con las que queremos trabajar de la BD real, liberando la conexión.
   - **Si queremos reflejar los cambios en la BD real, debemos confirmar nuestro objeto DataSet**

3. **Objeto DataAdapter**: Este objeto maneja la conexión a la base de datos por nosotros, y **se utiliza para insertar datos en un objeto DataSet y actualizar la base de datos con los cambios realizados en el DataSet**. Si estás utilizando SQL Server, utilizarías un SqlDataAdapter. **no tengo que preocuparme por abrir y cerrar la conexión**

4. **Objeto CommandBuilder: Este objeto es opcional y se utiliza para crear automáticamente los comandos SQL necesarios para actualizar la base de datos con los cambios realizados en el DataSet**. Si estás utilizando SQL Server, utilizarías un SqlCommandBuilder. (También se pueden proporcionar las sentencias SQL explícitamente o por medio de procedimientos almacenados.)

DataSet, DataRow y DataColumn son agnósticos respecto a la base de datos y no tienen un prefijo específico para SQL Server.

ATENCIÓN: En modo desconectado siempre necesitamos recuperar los datos (SELECT) para poder trabajar con ellos (INSERT, UPDATE, DELETE)

1. **Definición de un comando Select: Primero, creamos una base de datos en memoria utilizando un objeto DataSet. Después, llenamos este objeto con las tablas en las que estamos interesados trabajando con el objeto DataAdapter y su método Fill() (para recuperar los registros de la base de datos y colocarlos en un DataSet). Ejemplo:**

```
DataSet bdvirtual = new DataSet();
SqlDataAdapter da = new SqlDataAdapter("select * from Cliente", c);
da.Fill(bdvirtual,"cliente");
```

2. **Trabajar con la base de datos en local: Una vez que tenemos nuestros datos en un objeto DataSet, podemos manipularlos como si estuviéramos trabajando con la base de datos directamente**, pero sin la necesidad de una conexión constante. Podemos acceder y modificar estos datos utilizando objetos **DataRow** y **DataColumn**.

```
// Crear una nueva instancia de DataTable y asignar la tabla 'Cliente' del DataSet a ella.
DataTable t = bdvirtual.Tables["cliente"];
```

3. **Insertar un nuevo registro: Para agregar un nuevo registro a la base de datos local, creamos una nueva fila (DataRow), llenamos la fila con los datos del nuevo registro, y luego añadimos la fila a la tabla correspondiente del DataSet**. Ejemplo:

```
DataRow nuevafila = t.NewRow();
nuevafila[0] = textBox1.Text;
nuevafila[1] = textBox2.Text;
nuevafila[2] = textBox3.Text;
t.Rows.Add(nuevafila);
```

```
t.Rows[i]["usuario"]=cli.Usuario;
t.Rows[i]["contraseña"] = cli.Contraseña;
```

·(A través de un índice)

4. **Actualizar la base de datos real: Después de manipular los datos en el DataSet, podemos actualizar la base de datos real con los cambios realizados. Utilizamos un CommandBuilder para construir automáticamente las sentencias SQL necesarias para realizar las actualizaciones en la base de datos y**, utilizamos el objeto DataAdapter y su método Update() para actualizar la BD.Ejemplo:

```
SqlCommandBuilder cbuilder = new SqlCommandBuilder(da);
da.Update(bdvirtual, "cliente");
```

También tenemos que poner try, catch, finally

Control GridView:

El control GridView es un poderoso control en ASP.NET que permite presentar datos en forma de tabla (filas y columnas). Con el asistente no mantiene las 3 capas

Primero, necesitas **obtener los datos que deseas mostrar**. Este proceso normalmente involucra la ejecución de una **consulta SQL** y la **colocación de los resultados en un DataSet. Luego, establece los datos que se van a mostrar en el GridView y llamas a DataBind() para enlazar los datos.**

```
ENCliente enl = new ENCliente();
DataSet d = new DataSet();

protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        d = enl.listarClientesD();
        GridView1.DataSource = d;
        GridView1.DataBind();
    }
}
```

También puedes **configurar el GridView para permitir la edición de los datos**. Para hacer esto, puedes configurar **AutoGenerateSelectButton=true**, que agrega un botón de selección a cada fila, y luego puedes **manejar el evento SelectedIndexChanged para** mostrar los datos de la fila seleccionada en los controles de entrada para la edición. **(copiar las celdas al textbox)**

```
protected void GridView2_SelectedIndexChanged(object sender, EventArgs e)
{
    TextBox1.Text = GridView2.SelectedRow.Cells[1].Text;
    TextBox2.Text = GridView2.SelectedRow.Cells[2].Text;
    TextBox3.Text = GridView2.SelectedRow.Cells[3].Text;
    TextBox3.Enabled = false;
}

public void EditButton_Click(object sender, EventArgs e)
{
    ENCliente en = new ENCliente();
    en.Usuario = TextBox1.Text;
    en.Contraseña = TextBox2.Text;
    d = en.ModificarCliente(GridView2.SelectedIndex);
    GridView2.DataSource = d;
    GridView2.DataBind();
}
```

Capa EN:

```
public DataSet ModificarCliente(int i)
{
    CADcliente c = new CADcliente();
    DataSet a = c.ModificarCliente(this,i);
    return a;
}
```

cuando el usuario hace clic en el botón de edición, se llama a un método que actualiza los datos en la base de datos y luego vuelve a enlazar el GridView para mostrar los datos actualizados.

Ejemplo Borrar: (le podemos pasar la fila o el Id //DNI)

1. habilitar la opción **AutoGenerateDeleteButton** en el GridView. Esto hará que ASP.NET genere automáticamente un botón "Eliminar" en cada fila.

2. manejar **el evento RowDeleting para eliminar la fila correspondiente** de la base de datos. **la propiedad RowIndex de GridViewDeleteEventArgs para determinar qué fila está siendo eliminada.**

```
protected void GridView2_RowDeleting(object sender, GridViewDeleteEventArgs
{
    ENCliente en = new ENCliente();
    DataSet d = en.BorrarCliente(e.RowIndex);
    GridView2.DataSource = d;
    GridView2.DataBind();
}
```

Capa EN y capa CAD

```
public DataSet BorrarCliente(int i)
{
    CADcliente c = new CADcliente();
    DataSet a = c.BorrarCliente(this, i);
    return a;
}
```

```
public DataSet BorrarCliente(ENCliente cli, int i)
{
    DataSet bdvirtual = new DataSet();
    SqlConnection c = new SqlConnection(s);
    SqlDataAdapter da = new SqlDataAdapter("select * from Cliente", c);
    da.Fill(bdvirtual, "cliente");
    DataTable t = new DataTable();
    t = bdvirtual.Tables["cliente"];
    t.Rows[i].Delete();
    SqlCommandBuilder cbuilder = new SqlCommandBuilder(da);
    da.Update(bdvirtual, "cliente");
    return bdvirtual;
}
```

usa el índice de la fila para encontrar la fila correspondiente en el DataSet y la elimina

Paginación GridView:

Se habilita estableciendo la propiedad AllowPaging en true. También se puede establecer el número de elementos por página con la propiedad **PageSize**, se necesita escribir código para el evento **PageIndexChanging**.

```
protected void GridView2_PageIndexChanging(object sender,
 GridViewPageEventArgs e)
        {
            d = enl.listarClientesD();
           GridView2.PageIndex = e.NewPageIndex;
            GridView2.DataSource = d;
            GridView2.DataBind();
        }
```

Concurrencia: (Evitar conflictos)

Se refiere a la situación en la que varios usuarios pueden modificar los datos de los mismos registros al mismo tiempo. Existen diferentes formas de manejar la concurrencia:

- **concurrencia pesimista: cuando una fila es leída por un usuario, se bloquea para cualquier otro usuario hasta que el usuario original la libera.**
- **concurrencia positiva : las filas están disponibles para su lectura en todo momento y pueden ser leídas por varios usuarios al mismo tiempo**. Sin

embargo, **si un usuario intenta modificar una fila que ya ha sido modificada por otro, se produce un error y no se realiza la modificación.**

- **"Last Win": no proporciona control sobre la concurrencia. El último cambio que se escribe en los datos es el que permanece, independientemente de cualquier conflicto potencial.**
- **Escribir código para gestionar el conflicto**

En ADO.NET, se utiliza una forma de concurrencia positiva: el objeto DataSet mantiene dos versiones de cada fila: la versión original **(como se leyó de la base de datos)** y la versión actualizada **(representando los cambios del usuario). Cuando se actualiza una fila, se comparan los valores originales con la fila actual en la base de datos para ver si ha habido modificaciones. Si la fila ha sido modificada, es necesario capturar una excepción; de lo contrario, se realiza la actualización.**

El evento RowUpdated se utiliza para permitir a los usuarios determinar qué cambios deben conservarse después de cada operación de actualización.

Conectado vs Desconectado: El acceso a datos conectado (como con DataReader) puede ser más rápido y ligero que el acceso a datos desconectado (como con DataSet), pero es de solo lectura y no puede trabajar con más de una tabla o base de datos. El acceso a datos desconectado puede ser más lento y pesado, pero permite operaciones de lectura y escritura y puede trabajar con más de una tabla o base de datos.

- Para realizar consultas de sólo lectura, que únicamente serán necesarias realizarlas una vez (no tendremos que volver a acceder a filas anteriores) el objeto recomendado es **DataReader**.
- *Por ejemplo, para comprobar si un articulo se encuentra entre una tabla que guarda la lista de artículos del inventario de un almacén, basta con realizar una única consulta de sólo lectura.*

- Sin embargo, si vamos a realizar un acceso a datos más complicado, como puede ser la consulta de todos los artículos de diferentes tipos que pertenecen a un proveedor, la elección correcta sería utilizar **DataSet.**

Resumen Tema 8: (ASP.NET)

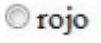
Controles de lista:

Son controles de interfaz de usuario que permiten a los usuarios seleccionar elementos de una lista. Estos controles incluyen:

```
                                                .Value
Label1.Text = DropDownList1.SelectedItem.Text.ToString();
```

- **Lista desplegable (DropDownList): seleccionar un solo elemento de una lista. Se puede utilizar la propiedad SelectedIndex, SelectedItem(Obtiene el elemento) o SelectedValue para obtener el elemento seleccionado.**

- **Lista de botones de radio (RadioButtonList): permite a los usuarios seleccionar un solo elemento de una lista.**



- **Lista de casillas de verificación (CheckBoxList): permite a los usuarios seleccionar varios elementos de una lista.(**

- **ListBox: Una lista de elementos que permite a los usuarios seleccionar uno o varios elementos. Necesito la propiedad:** SelectionMode=Multiple

(Si estoy rellenando elementos en una lista y el AutoPostBack está a true, cuando los inserto en el page_Load tengo que comprobar que solo se hace la 1ª vez: if (!Page.IsPostBack)

Cada uno de estos controles (es un Objeto de tipo ListItem). Cada objeto ListItem tiene propiedades de **Text** (el contenido que se muestra), **Value** (un valor oculto en el código HTML), y **Selected** (indicando si el elemento está seleccionado o no).

Para encontrar todos los elementos seleccionados en un control de lista con selección múltiple, es necesario recorrer la colección Items y comprobar la propiedad ListItem.Selected de cada elemento. Foreach ...

Los elementos de la lista pueden ser añadidos de forma declarativa en el código HTML o dinámicamente en el código utilizando el método Add del objeto Items del control.

Por ejemplo:

```
DropDownList1.Items.Add("rojo");
DropDownList1.Items.Add(new ListItem("rojo","Red"));
```

Controles de navegación: Menú:

Se utiliza para crear un menú que puede colocarse en una página maestra. (

El control de menú tiene dos modos de visualización:

- **Modo estático: el menú está completamente expandido todo el tiempo**.
- **Modo dinámico,** solo se muestran los elementos hijos cuando el usuario mantiene el puntero del ratón sobre el nodo padre.

El control de menú es una colección de objetos MenuItem,

En el código C#, se puede utilizar el evento **MenuItemClick** para manejar las acciones que ocurren cuando el usuario hace clic en un elemento del menú. La acción específica puede depender del valor del elemento del menú que se haya seleccionado.

Controles de validación:

Son herramientas esenciales para asegurar que los usuarios introduzcan datos correctamente. Estos controles permiten validar datos **en el lado del cliente (usando JavaScript) y en el lado del servidor (usando C#).** ASP.Net detecta si el navegador soporta validación del lado del cliente, **si no se puede validar en el cliente sólo se hace en el servidor**

ASP.NET proporciona varios controles de validación predefinidos:

1. **RequiredFieldValidator: Asegura que el usuario ha proporcionado una entrada.**
2. **CompareValidator: Compara la entrada del usuario con un valor constante o el valor de otro control.**
   Tiene un •ControlToCompare/ValueToCompare • ControlToValidate • Type • Operator

```
ControlToValidate="txtBox1"  Type="Integer"
ControlToCompare="txtBox2"   Operator="LessThan"
```

3. **RangeValidator: Verifica que la entrada del usuario está entre los límites superior e inferior especificados.**
   Tiene un •MaximumValue • MinimumValue • Type
4. **RegularExpressionValidator: Comprueba que la entrada del usuario coincide con un patrón definido por una expresión regular (• ValidationExpression)**
5. **CustomValidator: Permite realizar validaciones personalizadas utilizando la lógica definida por el programador.**(Asociar con un evento propio)
   • ClientValidationFunction • OnServerValidate
6. **ValidationSummary: Muestra un resumen de todos los mensajes de error de los controles de validación.• ShowSummary**

Estos controles tienen propiedades comunes:

- **ErrorMessage (para mostrar el mensaje de error)**
- **ControlToValidate (para indicar el control a validar)** =>TextBox1
- **Text (para mostrar texto inicial).**
- **IsValid que indica si la validación fue exitosa o no. La página también tiene una propiedad IsValid que resume el estado de todos los controles de validación en la página.**

El proceso de validación (servidor)

Cada control Button tiene una propiedad Causes Validation, false ignora los controles de validación. True se realiza la validación de cada control. Para usar estos controles, se deben configurar sus propiedades y, a continuación, comprobar la propiedad IsValid en el evento del botón de envío para determinar si todos los controles de validación han pasado.

Objetos Session y Application:

- **Los objetos Session están asociados a un usuario particular y sirven como una manera de transportar y mantener los datos del usuario en páginas web.**
- **Los objetos Application son compartidos por todos los usuarios y permiten almacenar información compartida por toda la aplicación web**.

En ASP.NET,están implementados como colecciones o conjuntos de pares nombre-valor.

Sesión

Es el período de tiempo en el que un usuario particular interactúa con una aplicación web. Durante una sesión, **la identidad única de un usuario se mantiene internamente**. **Los datos se almacenan temporalmente en el servidor.** Finaliza si hay un timeout o si tú finalizas la sesión del visitante en el código.

Los objetos de sesión nos permiten preservar las preferencias del usuario y otra información del usuario al navegar por la aplicación web. En ASP.NET son tablas Hash en memoria con un timeout. Son únicos y están identificadas usando enteros 32-bit long conocidos como Session IDs

Application

Proporciona una manera sencilla de almacenar en el servidor datos comunes a todos los visitantes de nuestro sitio web.

```
Application["SiteName"] = "Mi aplicación";
```

- Cualquier página de la aplicación puede leer esa cadena:

```
string appName = (string)Application["SiteName"];
```

Para evitar problemas con múltiples sesiones intentando cambiar el valor de una variable de aplicación al mismo tiempo, **ASP.NET proporciona los métodos Application.Lock() y Application.Unlock() para manejar la exclusión mutua.**

Una vez que las variables están bloqueadas, las sesiones que intentan cambiarlas tienen que esperar hasta que las variables sean desbloqueadas.

El archivo global.asax: (No Contiene etiquetas HTMO ni ASP.NET)

• Se utiliza para definir variables globales y reaccionar a eventos globales (de nivel de aplicación y sesiones).

Al crear una variable de aplicación, es importante tener en cuenta que no se debe reiniciar cada vez. Para ello, se puede usar el **evento Application_Start** en el archivo Global.asax. Este evento se dispara una sola vez cuando la aplicación se inicia o la sesión.

• Importante: Cualquier cambio en el archivo global.asax reiniciará la aplicación

• Sólo puede haber un archivo global.asax . Debe residir en el directorio raíz de la aplicación

```
// Global.asax
void Application_Start(object sender, EventArgs e)
{
    // Código que se ejecuta al iniciarse la aplicación
    Application.Add("contador", 0);
}

void Session_Start(object sender, EventArgs e)
{
    // Código que se ejecuta cuando se inicia una nueva sesión
    Application["contador"] = (int)Application["contador"] + 1;
}
```

Cookies:

Las cookies son pequeños fragmentos de datos que una aplicación web puede guardar en el navegador del cliente para su uso posterior. **A diferencia de las sesiones, las cookies no se borran automáticamente cuando se cierra el navegador, a menos que el usuario las borre.**

Uso de Cookies en ASP.NET

En ASP.NET, las cookies se representan mediante la clase HttpCookie. Puedes leer las cookies del usuario a través de la propiedad Cookies del objeto Request y modificar las cookies del usuario a través de la propiedad Cookies del objeto Response.

```
// default.aspx.cs
protected void Page_Load(object sender, EventArgs e)
{
    HttpCookie userCookie;
    userCookie = Request.Cookies["UserID"];
    if (userCookie == null)
    {
        Label1.Text ="No existe la cookie, creando cookie ahora";
        userCookie = new HttpCookie("UserID", "Ana López");
        userCookie.Expires = DateTime.Now.AddMonths(1);
        Response.Cookies.Add(userCookie);
    }
    else
    {
        Label1.Text = "Bienvenida otra vez, " + userCookie.Value;
    }
}
```

Por defecto, las cookies expiran cuando se cierra el navegador, pero puedes cambiar este comportamiento estableciendo un tiempo de expiración específico. Ejemplo: se le asigna el valor de la cookie llamada UserID

Para borrar una cookie en ASP.NET, debes reemplazarla por una cookie con la **misma clave y una fecha de expiración en el pasado.**

```
HttpCookie userCookie= new HttpCookie("UserID");
userCookie.Expires=DateTime.Now.AddDays(-1);
Response.Cookies.Add(userCookie);
```

Email:

En ASP.NET, puedes usar la clase **SmtpClient** y la clase **MailMessage (configurar los detalles para el correo electrónico: destinatario, el asunto, el cuerpo del correo electrónico) para enviar correos electrónicos**.

```
SmtpClient smtpClient = new SmtpClient("smtp.gmail.com", 587);
MailMessage message = new MailMessage();
```

Resumen Tema 9: (Bibliotecas)

Una biblioteca es un conjunto de recursos, usualmente binarios, que incluyen subprogramas, clases, datos, iconos, entre otros. **Distribuir estos recursos en una biblioteca facilita su uso y reutilización (no es necesario recompilar). Se puede usar una biblioteca enlazando nuestro código con ella, lo que nos da acceso a su contenido.**

Hay dos tipos de bibliotecas:

- **Estáticas: el código de la librería se adjunta al ejecutable, aumentando su tamaño, pero no presentan problemas de dependencia y su distribución es simple.**
- **Dinámicas: se enlazan en tiempo de ejecución.**

Las bibliotecas están diseñadas para distribuirse en formato binario por varias razones. Evita que el usuario tenga que generar este formato a partir de los archivos fuente, a menudo un proceso complicado y costoso. Además, las bibliotecas dinámicas pueden actualizarse para resolver problemas sin necesidad de recompilación, aunque pueden ser fuente de problemas si no se manejan correctamente.

● Su finalidad es producir una DLL y no un ejecutable.

Visual Studio y proyectos de tipo “Librería”.

- **Los *proyectos* de *tipo librería* se pueden crear independientemente** de que tengan un proyecto de tipo “aplicación gráfica o de consola” asociados en la misma solución. De hecho, suele ser lo habitual.
- **Podemos crear nuestros proyectos de tipo *librería* que posteriormente podemos reutilizar en aplicaciones concretas.**
- Es una buena manera de dividir el trabajo dentro de un grupo de programadores. Cada *subgrupo* se dedica a crear una librería.

Añadrir referencias en el proyecto Web

Resumen Tema 10: (AJAX en .net)

Introducción a AJAX

Las aplicaciones basadas en HTTP tradicionales pueden resultar lentas e ineficientes, ya que requieren refrescar la página entera para cada cambio, haciendo que el usuario deba esperar hasta que la petición (request) finalice para seguir interactuando con la aplicación.

SOLUCIÓN: MANEJO DE LA PETICIÓN DE UN MODO ASÍNCRONO

AJAX (Asynchronous JavaScript and XML) surge como una solución a estos problemas, permitiendo realizar llamadas asíncronas que no bloquean la interfaz de usuario mientras se producen. Esta tecnología utiliza el objeto XmlHttpRequest para recuperar datos del servidor de forma asíncrona y JavaScript para actualizar el contenido de la página, lo que elimina el parpadeo y permite un uso más eficiente del ancho de banda.

• Provee librerías script en el cliente que incorporan tecnologías JavaScript y HTML dinámico (DHTML), y las integra con la plataforma de desarrollo basada en ASP.NET (FACILITA LA LLAMADA ASINCRONA AL SERVIDOR), y en el servidor componentes para que soporte las llamadas asíncronas del cliente

AJAX ofrece elementos de IU familiares, permite actualizaciones parciales de la página que refrescan sólo las partes de la página web que han cambiado y ofrece soporte para los navegadores más populares. (más eficiencia en una página web)

El funcionamiento de AJAX se basa en enviar una petición asíncrona XML-HTTP al servidor sin interrumpir la interactividad de la interfaz de usuario. Una vez se completa la petición XML-HTTP, JavaScript refresca la porción de la página afectada por la respuesta.

Se elimina start-stop entre el usuario y servidor. Una aplicación AJAX introduce un motor AJAX como intermediario entre el usuario y el servidor, es responsable de presentar la interfaz al usuario y de comunicarse con el servidor de parte del usuario. **Este motor puede manejar cualquier respuesta a una acción del usuario que no requiera volver al servidor, como la validación de datos simple, la edición de datos en memoria y la navegación. Si el motor necesita algo del servidor para elaborar la respuesta, hace estas peticiones de manera asíncrona, sin paralizar la interacción del usuario con la aplicación.**

Creación de una aplicación Web AJAX

Los controles de servidor ASP.net AJAX son componentes que integran código del cliente y del servidor para producir comportamientos AJAX. Algunos de los principales controles son:

1. **ScriptManager: registra el script de la librería Microsoft AJAX en la página, permitiendo características como la representación parcial de páginas y las llamadas a servicios web. Este control es necesario para utilizar los demás controles.**
2. **UpdatePanel: Este control permite refrescar partes seleccionadas de la página, en lugar de refrescar la página completa. Esto se hace a través de un postback síncrono.**
3. **UpdateProgress: Este control provee información de estado sobre las actualizaciones parciales de la página en los controles UpdatePanel.**
4. **Timer: Este control ejecuta postbacks en intervalos de tiempo definidos. Puede utilizarse para enviar la página completa, o usarse junto al control UpdatePanel para realizar actualizaciones parciales de la página en un intervalo definido.**

si tienes un botón fuera de un UpdatePanel y quieres que al hacer clic en este botón se actualice el contenido del UpdatePanel, puedes agregar un AsyncPostBackTrigger a la propiedad Triggers del UpdatePanel y establecer el ID del botón como el control que desencadena la actualización.

Control ASP.NET AJAX ToolKit

La librería ASP.NET AJAX Control Toolkit es una extensión de ASP.NET AJAX que contiene controles Web y extendedores para utilizar las características avanzadas de ASP.NET AJAX. Los controles pueden ser entidades por sí mismos, mientras que los extendedores añaden comportamientos a un control Web existente. Estos controles y extendedores mejoran la funcionalidad y la experiencia del usuario en la página web.

• Estos controles van desde un simple botón con una alerta asociada, hasta un complejo panel que podemos arrastrar por la pantalla; • en ambos casos, mandando y recogiendo información entre el cliente y el servidor sin ningún tipo de recarga de página.

• Instalarlo (es un exe): **lo que hace es añadir una librería DLL al visual studio con los controles nuevos**

- **El extendedor AlwaysVisibleControl permite mantener un control en la página web visible y flotante sobre el contenido mientras se realiza un scroll o se redimensiona la ventana del navegador.**
- **El extendedor CollapsiblePanel permite que cualquier control ASP.NET pueda ser maximizado o minimizado a voluntad. El estado del contenido (abierto o cerrado) se guarda durante los postbacks, por lo que permanecerá igual cuando se recargue la página. IMPORTANTE, este panel tendrá height=0**

- **El extendedor ModalPopup permite mostrar contenido deshabilitando la interacción con el resto de la página, emulando el efecto de la función**

"window.open(...)" de JavaScript sin necesidad de salir de la página en que estamos ni de abrir una nueva ventana del navegador. Style="display:none"

- **SlideShow extender: permite la creación de una presentación de diapositivas (slideshow) de imágenes. Necesita una lista de imágenes para mostrar que puede ser proporcionada por un método de un servicio web**(Webservice)**.**
- **PasswordStrength: Este es un extensor de control que puede ser adjunto a un TextBox.** Puede mostrar la fortaleza de la contraseña en forma de texto o como una barra de indicador.

```
<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
<asp:PasswordStrength ID="TextBox2_PasswordStrength" runat="server"
  Enabled="True" TargetControlID="TextBox2"
  DisplayPosition="RightSide"
  StrengthIndicatorType="BarIndicator"
  PreferredPasswordLength="10"
  PrefixText="Seguridad:"
  TextStrengthDescriptions="Muy baja;Debil;Media;Fuerte;Excelente"
  MinimumNumericCharacters="1"
  MinimumSymbolCharacters="1"
  HelpStatusLabelID="Label2"
  RequiresUpperAndLowerCaseCharacters="true"
  BarBorderCssClass="barBorder"
  BarIndicatorCssClass="barInternal">
</asp:PasswordStrength>
```

- **ConfirmButton: Este es un extensor que se puede asociar a los botones de una página.** Cuando el usuario hace clic en el botón, aparece un cuadro de diálogo de confirmación. Si el usuario acepta, el botón funciona normalmente; **si el usuario cancela, se puede definir un comportamiento específico mediante la propiedad OnClientCancel. Es útil para solicitar confirmación al usuario antes de realizar acciones que podrían tener un impacto significativo, como la eliminación de datos.**

```
ConfirmText="Seguro que deseas aceptar?"
```