Introducció a ASP.NET

Francisco Ortega Belmonte

PID_00194031

Universitat Oberta de Catalunya

Índex

Objectius

Els objectius d'aquest mòdul didàctic són els següents:

1. Mostrar l'ús de la tecnologia Ajax en un dels entorns de desenvolupament més utilitzats avui dia: .NET.

2. Conèixer què és ASP.NET i la seva implementació Ajax.

3. Familiaritzar-se amb l'entorn de desenvolupament Visual Studio 2010.

4. Crear llocs web i dissenyar pàgines ASP.NET interactives que responguin a les diferents accions dels usuaris.

5. Comprendre el cicle de vida d'una pàgina ASP.NET entre diferents tipus de peticions.

1. L'entorn de desenvolupament ".NET Framework"

.NET és un terme utilitzat per Microsoft per a donar nom a un conjunt de tecnologies, –algunes de revolucionàries, altres no tant–, que estan dissenyades per a ajudar els desenvolupadors a crear diferents tipus d'aplicacions, des d'aplicacions d'escriptori o serveis de Windows fins a aplicacions web.

El conjunt de tecnologies que formen l'entorn de desenvolupament[1] ".NET Framework" són les següents:

(1)En anglès, *framework*.

- Els **llenguatges de programació .NET**: entre els quals hi ha Visual Basic, C#, F# i C++.
- El **Common Language Runtime (CLR)**: és el motor que executa tots els programes de .NET i que proveeix serveis automàtics per a aquestes aplicacions, com la comprovació de seguretat o la gestió de memòria.
- La **biblioteca de classes de .NET**: conté milers de classes, interfícies, estructures... amb una funcionalitat predefinida que es poden utilitzar en les aplicacions .NET.
- **ASP.NET**: és el motor que permet executar les aplicacions web creades en .NET.
- **Visual Studio**: és l'eina de desenvolupament per excel·lència de Microsoft. Permet desenvolupar qualsevol tipus d'aplicació .NET i depurar-la.

1.1. C# i Visual Basic

Durant aquest apartat, com es podrà veure, utilitzarem el llenguatge C# per a fer els exemples. Aquest llenguatge s'assembla bastant a Java i a C++. Al seu torn, C# i Visual Basic són bastant similars; de fet, gairebé qualsevol bloc de codi pot ser traduït d'un llenguatge a un altre, però la seva sintaxi és diferent.

Tant C# com a Visual Basic utilitzen la biblioteca de classes de .NAT i són compatibles amb el CLR.

1.2. El Common Language Runtime (CLR)

El Common Language Runtime (CLR[2]) és el motor que suporta tots els llenguatges .NET. En realitat, suporta el llenguatge intermedi entre el llenguatge natiu i el llenguatge de programació Microsoft Intermediate Language (MSIL[3]). El CLR interpreta aquest codi intermedi i el transforma en codi màquina.

(2)*CLR* és la sigla de *Common Language Runtime*.

(3)*MSIL* és la sigla de *Microsoft Intermediate Language*.

Per entendre-ho millor, el CLR seria el símil en Microsoft de la màquina virtual en Java.

1.3. La biblioteca de classes de .NET

La biblioteca de classes de .NET és un repositori gegant de classes que proporcionen una funcionalitat predefinida, des de la lectura d'un arxiu XML fins a l'enviament d'un correu electrònic. Qualsevol dels llenguatges .NET pot utilitzar aquesta biblioteca de classes.

Certes parts de la biblioteca mai no s'utilitzen en programació web i solament s'utilitzen per a fer aplicacions d'escriptori, mentre que altres estan concebudes exclusivament per a aplicacions web.

Com hem pogut veure, la filosofia de Microsoft és proporcionar la infraestructura de les funcionalitats més comunes perquè els desenvolupadors només hagin de programar codi específic de l'aplicació.

2. El llenguatge de programació C#

Abans de començar a crear una aplicació ASP.NET el programador ha de triar el llenguatge de programació, C# o Visual Basic. L'opció normal si no s'ha programat mai és Visual Basic, perquè s'assembla més al llenguatge natural; en canvi, si ja s'ha programat abans en C, C++ o Java l'elecció sol ser C#.

En aquest mòdul mostrarem les característiques principals de C#, els tipus de dades, les operacions, els condicionals o els bucles existents.

2.1. Característiques i peculiaritats

Les principals característiques que defineixen C# com un dels llenguatges més potents i moderns d'avui dia són:

a) És orientat a objectes. A excepció d'algunes dades simples com els enters, flotants o *strings*, la resta de les entitats són classes predefinides en les biblioteques de classes de .NET.

b) És un llenguatge compilat. Es necessita un compilador específic que tradueixi el llenguatge C# a codi MSIL que, com sabem, és el que interpreta el CLR.

c) Disposa de la biblioteca de classes .NET. Des de C# tenim accés a tota la biblioteca de classes de .NET, que implementa multitud de funcionalitats.

d) Disposa de *garbage collector*: La destrucció i alliberament de memòria es gestiona de manera automàtica i totalment transparent al desenvolupador.

D'altra banda, algunes de les peculiaritats que diferencien C# d'altres llenguatges de programació són:

a) Diferenciació entre majúscules i minúscules. En C# no és el mateix definir una variable anomenada *prova* que una altra variable *Prova*. Alguns llenguatges de programació, com per exemple Visual Basic, no fan aquesta distinció.

b) Comentaris. Els comentaris són textos descriptius que són ignorats pel compilador. C# disposa de dos tipus de comentaris bàsics:

- El comentari d'una línia:

- El comentari de múltiples línies:

c) Terminació de sentència. Tota sentència en C# ha d'acabar amb el caràcter "`;`". És important recordar que una sentència es pot compondre de més d'una línia, com podem veure a continuació:

```
   // Sentència en una línia
valor = valor1 + valor2 + valor3;
   // Sentència en diverses línies
valor = valor1 + valor2 +
valor3;
```

d) Blocs. Tot conjunt de sentències ha de començar amb el caràcter "`{`" i acabar amb el caràcter "`}`". Aquests conjunts poden ser funcions, classes, bucles...

```
{
   //Conjunt de sentències
}
```

2.2. Variables i tipus de dades

Igual que en tots els llenguatges de programació, en C# es poden mantenir dades en memòria utilitzant variables. Les variables poden emmagatzemar nombres, text, dates, hores, i fins i tot poden desar adreces en altres objectes. Quan es declara una variable se li assigna un nom i s'especifica el tipus de dades al qual pertany.

```
//Declaració d'un enter anomenat codi
int codi;
//Declaració d'un string anomenat nom
string nom;
```

En la taula següent podem veure els tipus de dades disponibles en C#:

Figura 1. Tipus de dades en C#

Nom del tipus	Descripció
`byte`	Enter de 0 a 255
`short`	Enter de –32.768 a 32.767
`int`	Enter de –2.147.483.648 a 2.147.483.647
`long`	Enter de –9,2e18 a 9,2e18
`float`	Nombre de coma flotant de –3,4e38 a 3,4e38
`double`	Nombre de coma flotant de –1,8e308 a 1,8e308
`decimal`	Nombre fraccionari de 128 bits
`char`	Caràcter Unicode
`string`	Conjunt de caràcters Unicode

Nom del tipus	Descripció
`bool`	Valor de *true* o *false*
`DateTime`	Representació de dates

2.2.1. Assignacions i inicialitzacions

Quan es declara una variable, s'hi poden assignar valors sempre que aquests valors siguin del tipus correcte.

```
//Declaració de variables
int codi;
string nom;
//Assignació de valors
codi = 10;
nom = "UOC"
```

També es pot assignar un valor en el mateix moment en què es declara la variable.

```
int codi = 10;
string nom = "UOC"
```

2.2.2. *Arrays*

Els *arrays* permeten al desenvolupador emmagatzemar una sèrie de valors del mateix tipus. Es pot accedir a cada valor mitjançant la posició que ocupa en l'*array*, tenint en compte que el primer valor d'un *array* té índex 0.

```
//Creació d'un array amb quatre strings
string[] stringArray = new string[4];
//Creació i inicialització d'un array amb quatre strings
string[]
stringArray = {"1","2","3","4"};
```

Per a accedir a un element, se n'ha d'especificar l'índex entre claudàtors `[]`.

```
int[] intArray = {1,2,3,4};
int element = intArray[2]; //element s'inicialitza amb un 3
```

2.2.3. `ArrayList`

Els *arrays* en C# no poden ser redimensionats, és a dir, una vegada creats no es poden canviar de mida. Si es necessita un *array* dinàmic, es pot utilitzar una classe de la biblioteca de classes de .NET anomenada *ArrayList*, que a més accepta que els elements dels *arrays* siguin de qualsevol tipus.

Utilització de la classe `ArrayList`

A continuació veiem un exemple d'utilització de la classe ArrayList:

```
//Creació d'un objecte ArrayList
ArrayList llistaDinamica = new ArrayList();
//Addició de diferents strings a la llista
llistaDinamica.Add("primer");
llistaDinamica.Add("segon");
llistaDinamica.Add("tercer");
```

2.2.4. Enumeracions

Una enumeració és un grup de constants relacionades, a cadascuna de les quals es dóna un nom descriptiu. Cada valor d'una enumeració correspon a un nombre enter prefixat.

```
//Definició d'una enumeració anomenada TipusUsuari
enum TipusUsuari
{
   Admin,
   Convidat,
   Erroni
}
```

Ara ja es pot utilitzar `TipusUsuari` com un tipus de dades especial amb només tres valors.

```
//Creació d'un nou valor de tipus Admin
TipusUsuari usuari = TipusUsuari.Admin;
```

2.2.5. Operacions amb variables

En C# el desenvolupador pot utilitzar tots els tipus estàndard d'operacions. Quan es treballa amb tipus numèrics es poden utilitzar els símbols matemàtics que podem veure en la taula següent:

Figura 2. Operacions en C#

Operador	Descripció	Exemple
+	Suma	`1 + 1 = 2`
-	Resta	`5 - 2 = 3`
*	Multiplicació	`2 * 5 = 10`

Operador	Descripció	Exemple
/	Divisió	`5.0 / 2 = 2.5`
%	Residu de la divisió sencera	`7 % 3 = 1`

Per a controlar l'ordre en què s'executen, les expressions poden agrupar-se entre parèntesis.

```
int nombre;
nombre = 4 + 2 * 3;
//nombre serà 10
nombre = (4 + 2) * 3
//nombre serà 18
```

La divisió en C# pot ser una mica confusa. Si es divideix un nombre enter per un altre enter, C# fa una divisió entera, és a dir, es descarta automàticament la part fraccionària del resultat. La solució a aquest problema és definir un dels nombres, o tots dos, com a fraccionaris; per exemple, `5` passaria a ser `5.0`. D'aquesta manera obtindrem un resultat correcte.

En C# l'operació addició + es pot utilitzar també per a concatenar *strings*.

```
//Concatenació de tres strings
nomComplet = nom + " " + cognoms;
```

En C# quan l'operació i l'assignació es fan sobre la mateixa variable es poden utilitzar els operadors de la manera següent:

```
//Afegir 10 a valor. És el mateix que valor = valor + 10
valor += 10
//Multiplicar valor per 3. És el mateix que valor = valor * 3
valor *= 3
```

2.2.6. Conversions de tipus

Convertir informació d'un tipus de dada a un altre és una cosa bastant comuna en el món de la programació. Les conversions de dades poden ser de dos tipus: implícites o explícites; per exemple, la conversió d'un tipus `int` a un de `long` és implícita, perquè no implica pèrdua d'informació; no obstant això, la conversió d'un tipus `long` a un d'`int` és explícita, a causa que hi pot haver pèrdua d'informació.

Per a les conversions implícites no es necessita cap codi especial:

```
int valorPetit;
```

```
long valorGran;
//La conversió sempre es pot fer perquè un long pot contenir un int
valorGran = valorPetit;
```

En canvi, en les conversions explícites, és necessari indicar el tipus entre parèntesis a l'esquerra de la variable.

```
int valorGran = 1000;
short valorPetit;
//Si el valorGran supera la capacitat d'un short hi haurà pèrdua d'informació.
valorPetit = (short)valorGran;
```

2.2.7. El tipus `String`

La classe `String` de la biblioteca de classes de .NET ens permet manipular de múltiples maneres una cadena de text.

Exemple

```
string mistring = "Això és un string d'exemple ";
mistring = mistring.Trim();          //"Això és un string d'exemple";
mistring = mistring.Substring(0,4);   //"Això";
mistring = mistring.ToUpper();        //"AIXÒ";
mistring = mistring.Replace("O","A"); //"AQUESTA";

int longitude = mistring.Length;      // = 4;
```

En els exemples anteriors cada mètode de la classe `String` genera un nou *string* que és assignat a la variable `mistring`. Totes aquestes sentències es poden fer en una sola línia:

```
mistring = mistring.Trim().Substring(0,4).Replace("O","A");
```

En la taula següent veiem algun dels mètodes i propietats més importants de la classe `String`:

Figura 3. Funcions i propietats de la classe `String`

Membre	Descripció
`Lenght`	Retorna el nombre de caràcters de l'*string*.
`ToUpper()` i `ToLower()`	Retorna un *string* amb tots els caràcters de l'*string* original en majúscula o minúscula.
`Trim()`	Retorna un *string* sense els espais de l'*string* original.
`Insert()`	Insereix un altre *string* en l'*string* original en la posició indicada.
`Remove()`	Elimina un nombre determinat de caràcters des d'una posició determinada.
`Replace()`	Reemplaça un *substring* de l'*string* original amb un altre *string*.
`Substring()`	Retorna una porció de l'*string* d'una longitud determinada.
`StartsWith()` i `EndsWith()`	Determina si un *string* comença o acaba en un *substring* determinat.
`Split()`	Retorna un *array* amb tots els *substrings* resultants de seccionar l'*string* per a un caràcter determinat.

Membre	Descripció
`Join()`	Fusiona un *array* de *strings* en un *string* únic.

2.2.8. Els tipus `DateTime` i `TimeSpan`

Els tipus de dades `DateTime` i `TimeSpan` disposen de mètodes i propietats per a manipular dates i temps, respectivament.

Exemples

Podem veure a continuació un exemple de manipulació de dates amb `DateTime`:

```
DateTime lamevaData = DateTime.Now;
miFecha = lamevaData.AddDays(100);
string dateString = lamevaData.Year.ToString();
```

En l'exemple següent veiem com podem buscar la diferència en minuts entre dues dates:

```
DateTime lamevaData1 = DateTime.Now;
DateTime lamevaData2 = DateTime.Now.AddHours(3000);
TimeSpan diferencia;
diferencia = lamevaData2.Substract(lamevaData1);
double nombreMinuts;
nombreMinuts = diferencia.TotalMinutes;
```

Les classes `DateTime` i `TimeSpan` també suporten els operadors aritmètics + i −, fins i tot entre un objecte `DateTime` i un altre de `TimeSpan`.

Exemple

```
DateTime lamevaData1 = DateTime.N
TimeSpan interval = TimeSpan.From
DateTime lamevaData2 = lamevaData
```

A continuació podem veure dues taules amb alguns mètodes i propietats de les classes `DateTime` i `TimeSpan`:

Figura 4. Funcions i propietats de la classe `DateTime`

Membre	Descripció
`Now`	Retorna la data i hora actuals.
`Today`	Retorna la data actual.
`Year`, `Date`, `Month`, `Hour`, `Minute`, `Second` i `Millisecond`	Retorna una part d'un `DateTime`.
`DayOfWeek`	Retorna el dia de la setmana d'un `DateTime`.
`Add()` i `Subtract()`	Afegeix o resta un `TimeSpan` d'un `DateTime`.
`AddYears()`, `AddMonths()`, `AddDays()`, `AddHours()`, `AddMinutes()`, `AddSeconds()`, `AddMilliseconds()`	Afegeix una porció de temps a un `DateTime`.
`DaysInMonth()`	Retorna el número de dia d'un mes determinat.

Figura 5. Funcions i propietats de la classe `TimeSpan`

Membre	Descripció
`Days`, `Hours`, `Minutes`, `Seconds`, `Milliseconds`	Retorna una part d'un `TimeSpan`.

Membre	Descripció
`TotalDays`, `TotalHours`, `TotalMinutes`, `TotalSeconds`, `TotalMilliseconds`	Retorna el total de dies, hores... que té un `TimeSpan`.
`Add()` i `Subtract()`	Afegeix o resta un `TimeSpan` a un altre `TimeSpan`.
`FromDays()`, `FromHours()`, `FromMinutes()`, `FromSeconds()`, `FromMilliseconds()`	Construeix un `TimeSpan` a partir de dies, hores...

2.3. Condicionals lògics

Moltes vegades es necessita una condició lògica per a decidir quina acció cal prendre sobre la base d'una informació determinada. Una condició lògica pot ser avaluada com a veritable o falsa i en funció d'això executar un bloc de codi o un altre. Per a crear una condició podem utilitzar qualsevol combinació de variables o literals amb els operadors lògics que veiem en la taula següent:

Figura 6. Condicionals en C#

Operador	Descripció
==	Igual que
!=	Diferent que
<	Més petit que
>	Més gran que
<=	Més petit o igual que
>=	Més gran o igual que
&&	AND lògic
||	OR lògic

La sentència `if` és la peça clau de la lògica condicional en la programació. A continuació veiem un exemple en què s'avalua una condició i s'executa un bloc de codi o un altre:

```
if (nombre > 10)
{
   //Bloc A
}
else
{
   //Bloc B
}
```

C# també posa a la nostra disposició la sentència `switch`, que ens serveix per a avaluar una variable per a diversos valors possibles. L'única limitació de la sentència `switch` és que la variable solament pot ser dels tipus `int`, `bool`, `char`, `string` o el valor d'una enumeració.

```
switch (nombre)
{
   case 1:
      //Bloc A
      break;
   case 1:
      //Bloc B
      break;
   default:
      //Bloc C
      break;
}
```

Veiem que en cada bloc de codi hem d'indicar la paraula `break`; si no ho féssim, es continuaria executant el codi de manera seqüencial.

2.4. Bucles

En C#, igual que en la majoria de llenguatges, hi ha tres tipus de bucles per a executar blocs de codi de manera iterativa: `for`, `foreach` i `while`;

El bucle `for` permet al programador poder repetir un bloc de codi un nombre predeterminat de vegades. Per a crear un `for` és necessari especificar un valor inicial, un valor final i l'increment de cada iteració.

```
for (int i = 0; i < 10; i++)
{
   //Bloc de codi
}
```

El bucle `foreach` ens permet recórrer els elements d'un conjunt de dades. Amb un `foreach` no és necessari crear una variable per al comptador, i a més és especialment útil per a recórrer llistes, *arrays* o col·leccions de dades en general.

```
string[] stringArray = {"un","dos","tres"};
foreach (string element in stringArray)
{
   System.Diagnostigs.Debug.Write(element + " ");
}
```

Finalment, C# disposa del bucle `while`, que serveix per a comprovar una condició especifica abans o després de cada iteració d'un bloc de codi. Quan aquesta condició és falsa, se surt del bucle; per contra, mentre sigui veritable es van fent iteracions en el bloc de codi.

```
int i = 0;
while (i < 10)
{
   i += 1;
}
```

Una altra manera d'implementar el bucle `while` és posant la condició al final del bloc de codi, i d'aquesta manera ens assegurem que el codi s'executa almenys una vegada.

```
int i = 0;
do
{
   i += 1;
} while (i < 10);
```

2.5. Mètodes

Els mètodes en C# són el bloc de codi més bàsic que podem utilitzar per a organitzar el nostre codi. Quan es declara un mètode la primera part de la declaració indica el tipus de dades del valor de tornada i la segona el nom del mètode.

```
//Mètode que no retorna cap valor
void MetodeQueNoTornaValor()
{
   //Codi
}

//Mètode que retorna un enter
int MetodeQueRetornaUnInt()
{
   return 10;
}
```

Els mètodes poden acceptar paràmetres. Els paràmetres es declaren de manera molt similar a la declaració de variables.

```
int sumaNombres (int nombre1,int nombre2)
{
   return nombre1 + nombre2;
}
```

```
//Crida al mètode
int resultat = sumaNombres(10,10);
```

Cal destacar que en la declaració dels paràmetres els podem donar un valor per defecte.

```
private string NomUsuari (int ID, bool esAdmin = false)
{
   //Codi
}
```

Una característica pròpia de C# relacionada amb els mètodes és la sobrecàrrega, és a dir, podem crear més d'un mètode amb el mateix nom però amb diferents paràmetres.

```
public decimal GetPreuProducte(int ID)
{
   //Codi
}
public decimal GetPreuProducte(string nom)
{
   //Codi
}
```

2.6. Classes i objectes

C#, com hem dit, és un llenguatge de programació orientat a objectes. Una classe és la definició del codi d'un objecte i la creació d'una classe en C# es fa de la manera següent:

```
public class LaMevaClasse
{
   //Codi de la classe
}
```

En una classe podem definir variables que siguin variables membre o propietats de la classe. Aquestes variables poden tenir diferents nivells d'accés, que s'especifiquen en la declaració de la variable.

```
public class Producte
{
   private string nom ;
   private decimal preu;
}
```

En la taula següent podem veure els diferents nivells d'accés disponibles en C#:

Figura 7. Nivells d'accés en C#

Nivell d'accés	Accessibilitat
public	Accessible des de qualsevol classe
private	Només accesible des de membres de la mateixa classe
internal	Accesible des de qualsevol classe de l'assemblat actual
protected	Accessible des de membres de la mateixa classe o des de classes que hereten d'aquesta

Per a una classe també podem definir una sèrie de mètodes, que anomenarem *mètodes membre*, i igual que una variable membre, ha de tenir el nivell d'accés definit a la declaració.

```
public class Producte
{
   private string nom ;
   private decimal preu;

   public decimal GetPreu()
   {
      return this.preu;
   }
}
```

Un mètode especial que totes les classes han de tenir és el constructor de classe, que és l'encarregat d'inicialitzar els paràmetres de la classe. Hi pot haver un o més constructors, el nom dels quals serà el mateix de la classe a la qual pertanyen, però que es diferenciïn en el conjunt dels paràmetres que es passa a cadascun.

```
public class Producte
{
   private string nom ;
   private decimal preu;

   public void Producte(int preuInici)
   {
      this.preu = preuInici;
   }
      public decimal GetPreu()
   {
      return this.preu;
   }
}
```

Una vegada hem construït una classe com l'anterior en podem crear tantes instàncies com vulguem; cadascuna serà el que es denomina *objecte*.

```
Product producteVenda = new Product(50);
int preu = producteVenda.GetPreu();
```

3. Introducció a les eines de desenvolupament

Les aplicacions ASP.NET es poden crear amb un simple editor de textos, introduint tot el codi –tant el disseny de la interfície (XHTML/CSS/JavaScript) com la lògica (codi C#)– en mòduls . *aspx*, que serien compilats dinàmicament pel motor d'ASP.NET en el moment de rebre les sol·licituds.

Aquesta, no obstant això, és una solució viable únicament per a projectes molt petits, ja que és difícil mantenir un disseny d'interfícies a partir de l'escriptura manual dels atributs de tots els elements i d'una barreja de codis en diferents llenguatges en un mateix arxiu.

Depenent de quins siguin els nostres objectius i necessitats, tenim a la nostra disposició diferents eines útils per al desenvolupament d'aplicacions basades en ASP.NET. El punt de partida seria Visual Web Developer Express. Es tracta d'una eina gratuïta, descarregable directament des del lloc web de Microsoft i amb els elements necessaris per a construir aplicacions bàsiques i de nivell mitjà, amb limitacions sobretot a l'hora de treballar amb servidors de bases de dades.

Més sofisticats són els entorns de treball que ofereixen les diferents versions del Visual Studio 2010, que, a més d'aplicacions ASP.NET, tenen capacitat per a desenvolupar molts tipus de projectes: aplicacions per a Windows, per a dispositius mòbils, per a l'Office, etc.

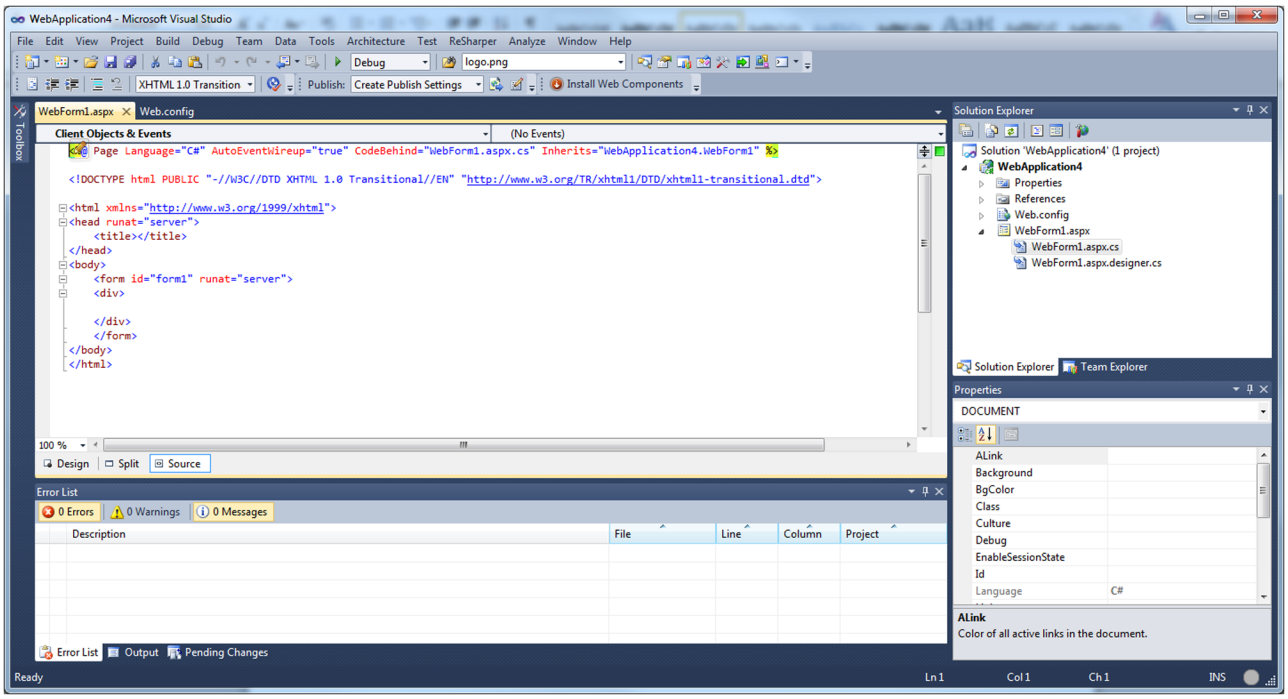
3.1. Visual Studio

Des de fa anys, l'eina de desenvolupament per excel·lència per a plataformes Windows es denomina Microsoft Visual Studio, un producte que reuneix compiladors per a diferents llenguatges, editors de codi amb característiques avançades d'ajuda a la codificació, dissenyadors d'interfícies d'usuari, depuració integrada, gestió de projectes complexos, edició de bases de dades tant locals com remotes, etc.

L'última versió del Visual Studio és la 2010 (figura 8), i va unida a la versió 4.0 de la plataforma .NET. A l'hora d'iniciar un nou projecte, no obstant això, és possible triar la versió de la plataforma .NET sobre la qual s'executarà: 2.0, 3.0, 3.5 o 4.0. Això permet usar una única eina per al desenvolupament d'aplicacions sobre diferents versions de la plataforma.

Hi ha disponibles múltiples edicions del Visual Studio dirigides tant a usuaris individuals (Standard i Professional) com a grups de treballs (Team System). Quant al desenvolupament d'aplicacions web basades en ASP.NET, les seves capacitats són bàsicament les mateixes.

Figura 8. Visual Studio 2010



A diferència del Visual Studio, el Visual Web Developer és una eina específica per al desenvolupament d'aplicacions web amb ASP.NET que no té opcions per a la construcció d'altres tipus de projectes. Per tant, si necessitem dissenyar solucions complexes en què l'aplicació web sigui solament una de les parts, el Visual Web Developer ens resultarà insuficient i haurem de recórrer a alguna de les edicions del Visual Studio. Pel que fa al nostre curs, tindrem més que suficient amb aquesta eina.

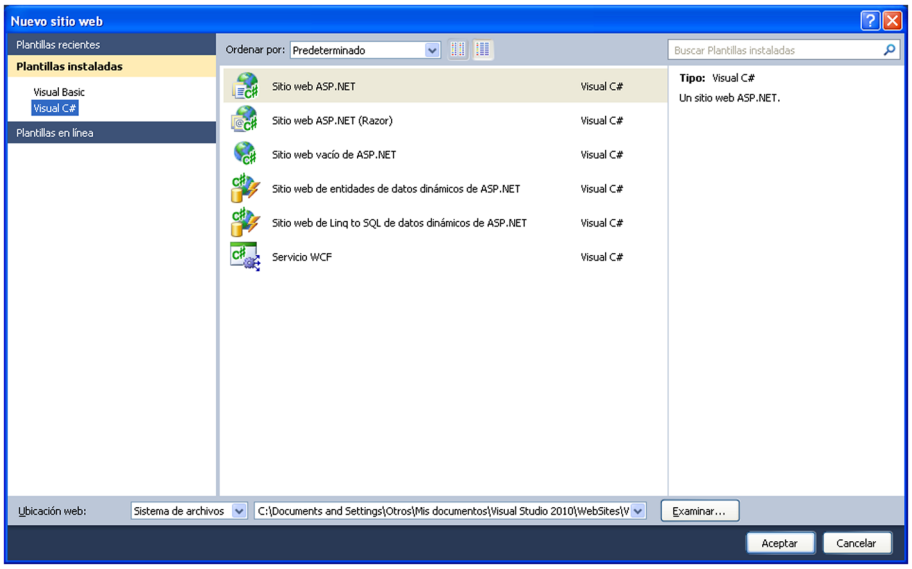
El Visual Web Developer permet usar com a llenguatge d'implementació tant el Visual Basic com el C#, però no permet seleccionar la versió de la plataforma .NET sobre la qual s'executarà el projecte, i s'assumeix sempre que serà la 4.0.

Una altra diferència respecte al Visual Studio és que el Visual Web Developer solament permet treballar amb bases de dades locals, no remotes. Tampoc no hi ha la possibilitat d'efectuar depuració remota, amb l'aplicació executant-se en un equip que no sigui el de desenvolupament.

Pel que fa a la resta, el Visual Web Developer incorpora el mateix dissenyador de pàgines ASP.NET que el Visual Studio, el mateix editor de codi, els mateixos components i també els mateixos serveis fonamentals: els de la plataforma .NET 4.0.

El gestor de projectes és pràcticament idèntic al del Visual Studio, igual que l'Explorador de bases de dades, l'Explorador d'objectes i moltes altres utilitats incloses en l'entorn.

Figura 9. Formulari de creació amb les diferents plantilles disponibles

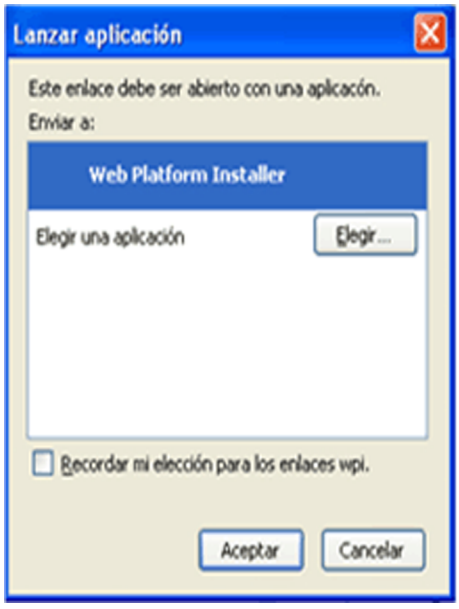


3.1.1. Instal·lació del Visual Web Developer

Perquè puguem descobrir les capacitats del Visual Web Developer primer és necessari instal·lar l'eina. Per a això accedirem a la pàgina de Microsoft Visual Web Developer 2010 Express.

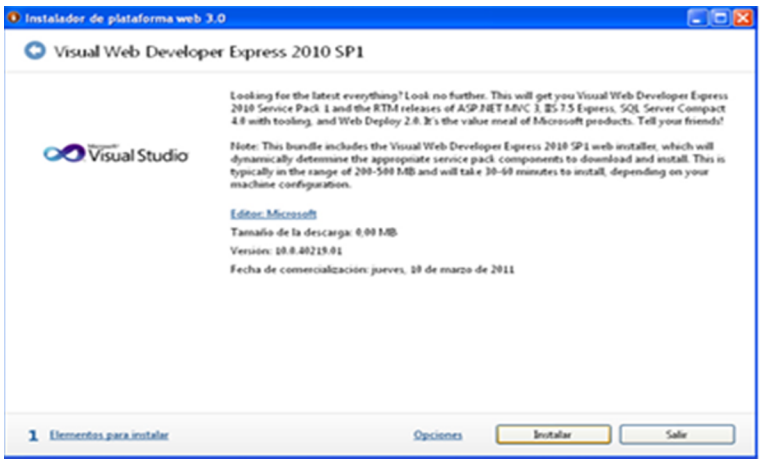
Una vegada a la pàgina, farem clic sobre el botó *Install now*. Això llançarà el Web Platform Installer, que ens baixarà i instal·larà el producte automàticament.

Figura 10. Web Platform Installer



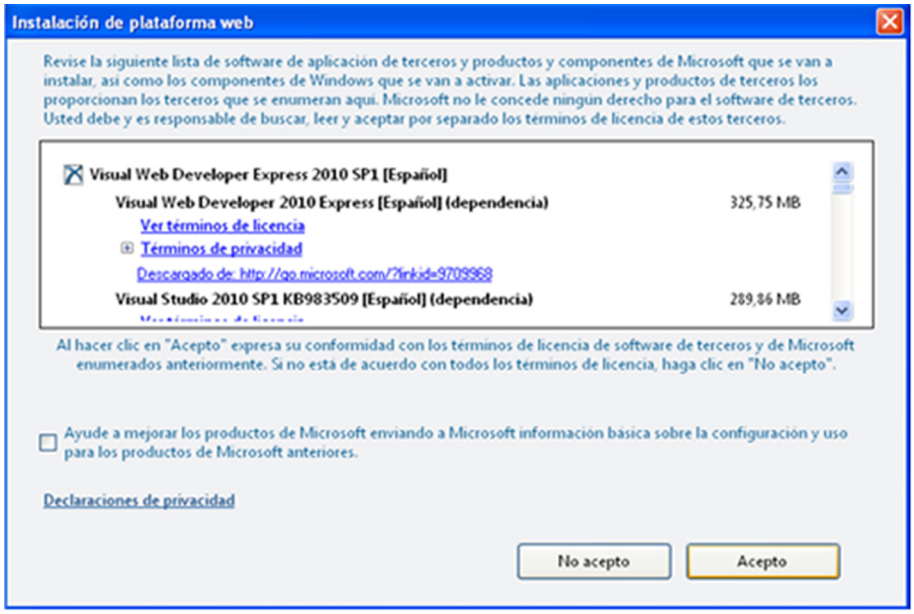
Una vegada el Web Platform Installer inicií l'instal·lador del Visual Web Developer, premem en el botó *Instalar*.

Figura 11. Instal·lació del Visual Web Developer Express



Acceptem les condicions d'ús i el programa d'instal·lació baixarà el Visual Web Developer i l'instal·larà de manera autònoma.

Figura 12. Condicions d'ús



3.1.2. Galeria de plantilles

En triar l'opció *File* > *New* > *Web Site*, o fer clic en el botó equivalent en la barra d'eines, veiem la galeria de plantilles instal·lades. Cada plantilla permet crear un tipus de projecte diferent o que contingui una sèrie d'elements diferents de partida.

En la galeria de plantilles (figura 9) sempre triarem l'opció *Sitio web ASP.NET* i, abans de fer clic sobre el botó *Aceptar*, establirem els paràmetres següents:

- La ubicació on s'allotjarà el projecte, que es pot triar entre aquestes tres possibilitats: *Sistema de archivos*, *HTTP* i *FTP*. En el nostre cas sempre triarem *Sistema de archivos* i indicarem la ruta on s'allotjarà el projecte.

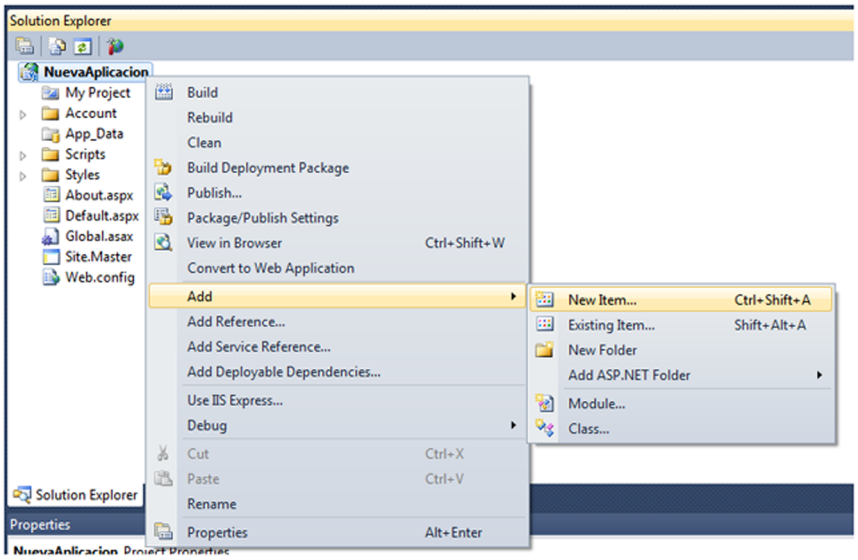
- El llenguatge de programació que s'utilitzarà per a codificar la lògica: *Visual Basic* o *C#*. En el nostre cas, *C#*.

Una vegada establerta la configuració de l'aplicació, un clic en el botó *Aceptar* generarà el projecte amb els mòduls inicials.

3.1.3. Explorador de solucions

Una vegada generat el projecte, els elements que el componen apareixeran en el Solution Explorer (figura 13), una eina fonamental per a l'administració dels projectes.

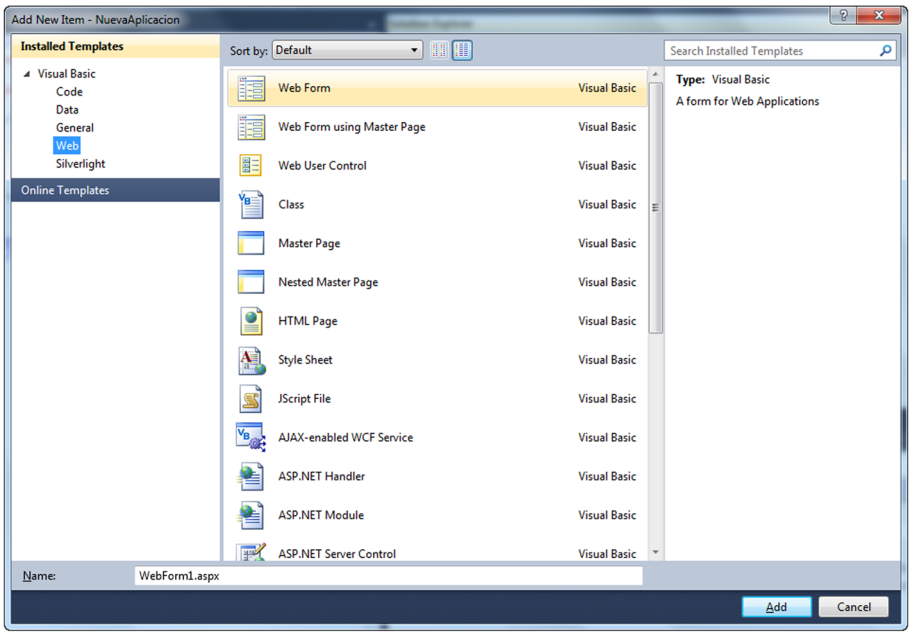
Figura 13. L'Explorador de solucions i el menú contextual associat al projecte



En principi, en aquesta finestra trobarem tres elements: la carpeta `App_Data`, la pàgina `Default.aspx` i el mòdul de configuració `web.config`. La pàgina té associat un arxiu de codi anomenat `Default.aspx.cs`. En seleccionar qualsevol dels elements en la finestra *Properties*, apareixeran totes les seves propietats. Amb el botó secundari del ratolí podrem accedir al menú contextual, específic segons el tipus d'element que hi hagi sota el punter del ratolí. Simplement amb un doble clic en un element dels mostrats en l'Explorador de solucions l'obrirem en el seu editor/dissenyador predeterminat. Recorrent al menú contextual es pot optar per obrir el mòdul en un altre programa, eliminar-lo del projecte, copiar-lo o dur a terme tasques específiques.

El menú contextual associat al projecte permet generar-lo i executar-lo, i també agregar referències i altres elements. L'opció *Add* > *New element* dóna pas a un quadre de diàleg similar al de la figura 14, en el qual es troben tots els tipus d'elements que és possible utilitzar en una aplicació ASP.NET.

Figura 14. Quadre de diàleg per a agregar nous elements a un projecte

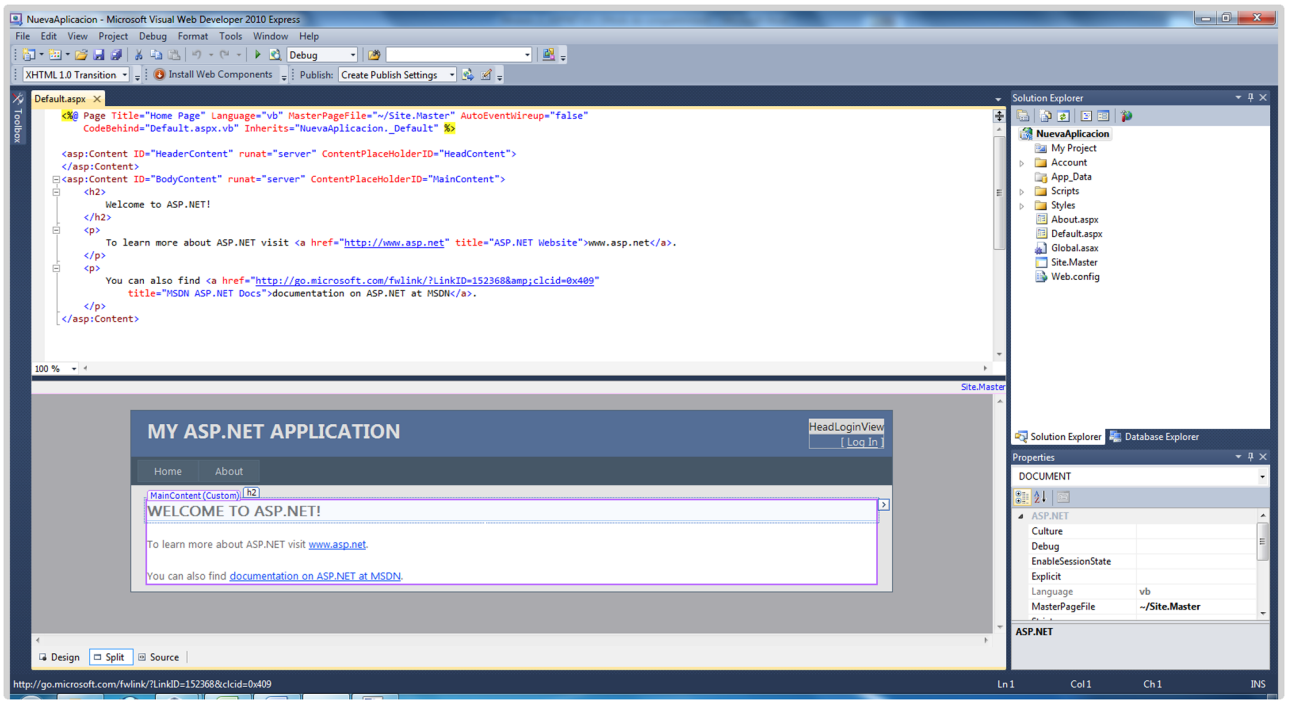


3.1.4. El dissenyador de pàgines

Quan creem un nou projecte, o fem doble clic sobre un mòdul `.aspx` en el *Solution Explorer*, veurem com s'obre el dissenyador de pàgines ASP.NET. Aquest es compon d'un dissenyador visual, similar a qualsevol dissenyador web, i d'un editor de codi XHTML/CSS. En la part inferior les opcions *Diseño*, *Dividir* i *Código* permeten alternar entre tres vistes diferents: únicament els elements de disseny, el dissenyador i l'editor, o únicament l'editor de codi. En la figura 15 es pot veure la segona de les opcions, amb la part central dividida en dues seccions. La part superior és l'editor i la inferior el dissenyador.

Mentre es treballa en el dissenyador (la part inferior en la figura 15), és possible utilitzar elements habituals en el disseny de pàgines web. La barra de botons que hi ha en la part superior facilita la selecció de tipus i mides de lletra, colors de tinta i fons, alineació, fins i tot de les llistes numerades i sense numerar, etc. També es pot recórrer a finestres auxiliars, com *Aplicar estils*, *Propietats CSS* i *Administrar estils*, que faciliten tota la gestió dels atributs que estableixen l'estil dels elements continguts a la pàgina. Els menús *Format* i *Taula*, actius solament quan està actiu el dissenyador, ofereixen opcions addicionals d'aplicació de format i inserció de taules HTML.

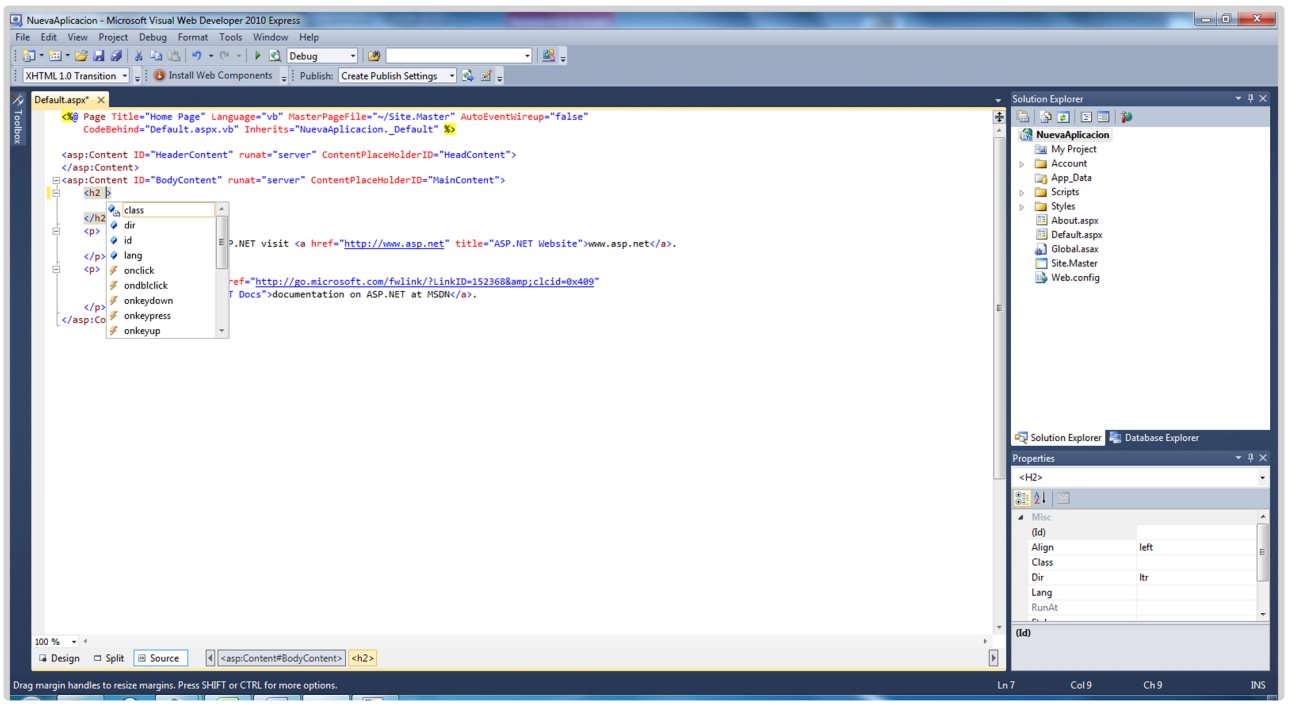
Figura 15. Dissenyador de pàgines i editor de codi combinats en una mateixa vista



En qualsevol moment és possible canviar del dissenyador a l'editor de codi i treballar sobre aquest. Els canvis duts a terme en el dissenyador es reflecteixen immediatament en el codi i viceversa. En escriure codi, quan introduïm el caràcter < per a iniciar una etiqueta[4] , l'editor ens oferirà ajuda immediata per mitjà de llistes d'elements XHTML, atributs CSS, valors que permet cada atribut, etc. En la figura 16, per exemple, es pot veure la llista de valors que permet l'atribut `style` d'una etiqueta `h2`.

[4] En anglès, *tag*.

Figura 16. No és necessari conèixer de memòria els elements i atributs de XHTML/CSS, l'editor ens els recorda



Encara que usant l'editor XHTML/CSS és possible introduir qualsevol element en una pàgina, per regla general resultarà molt més còmode fer-ho recorrent a la finestra *Toolbox* (vegeu la figura 17). Normalment, la trobarem oculta en el marge esquerre de l'entorn.

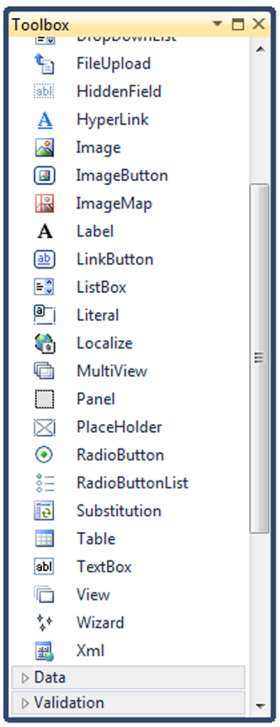


Figura 17. Quadre d'eines

En el *Toolbox*, els components apareixen agrupats en diverses categories: *Validation*, *Ajax Extensions*, *HTML*, etc. Els del grup HTML no són més que elements estàndard d'XHTML, de vegades amb algunes propietats preestablertes. La resta són components d'ASP.NET, és a dir, porcions de codi que s'executaran en el servidor, faran un cert treball i generaran com a resultat etiquetes XHTML/CSS.

La inclusió de qualsevol tipus de component en la pàgina és una operació realment senzilla: només cal agafar-lo de la finestra *Toolbox* i arrossegar-lo fins a la superfície del dissenyador. En aquest moment en l'editor de codi s'introduirà l'etiqueta corresponent, generalment del tipus `<asp:NomComponent atribut="valor"...>`, i en el dissenyador s'apreciarà l'aspecte del control.

Cada vegada que s'arrossega un component des del *Toolbox* fins a la superfície de disseny, el que s'obté és una còpia, un objecte (una instància d'una classe), amb uns atributs generals. Aquests es poden personalitzar modificant les propietats que exposen, utilitzant per a això la finestra *Properties* (figura 18).

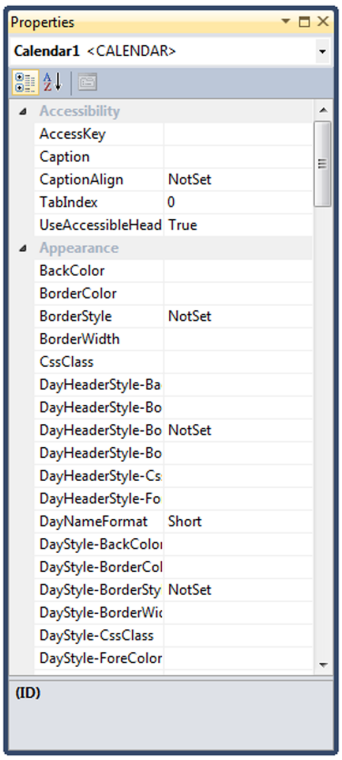


Figura 18. Llista de propietats

La modificació de certes propietats, com les que afecten l'estil del component, pot tenir un reflex immediat en l'aspecte de l'objecte en el dissenyador. En altres casos, no obstant això, les propietats tenen efecte únicament durant l'execució de l'aplicació, com per exemple, la propietat `AutopostBack`, amb la qual es regeix el comportament del component una vegada que està en el navegador del client, i determina si el canvi d'una propietat del component mateix provoca o no una comunicació amb el servidor.

3.1.5. L'editor de codi C#

A mesura que es va dissenyant la interfície de l'aplicació, col·locant components en el dissenyador i personalitzant-ne les propietats, l'habitual és que també es vagi codificant la lògica, establint el codi que s'ha d'executar quan s'enviï un formulari de dades, es canviï la selecció d'una llista, etc. La introducció del codi en el servidor, que en gran part estarà associat a esdeveniments generats per la pàgina o els components continguts en aquesta, es du a terme mitjançant un editor especifico per a C#.

Es pot obrir l'editor de codi de diferents maneres:

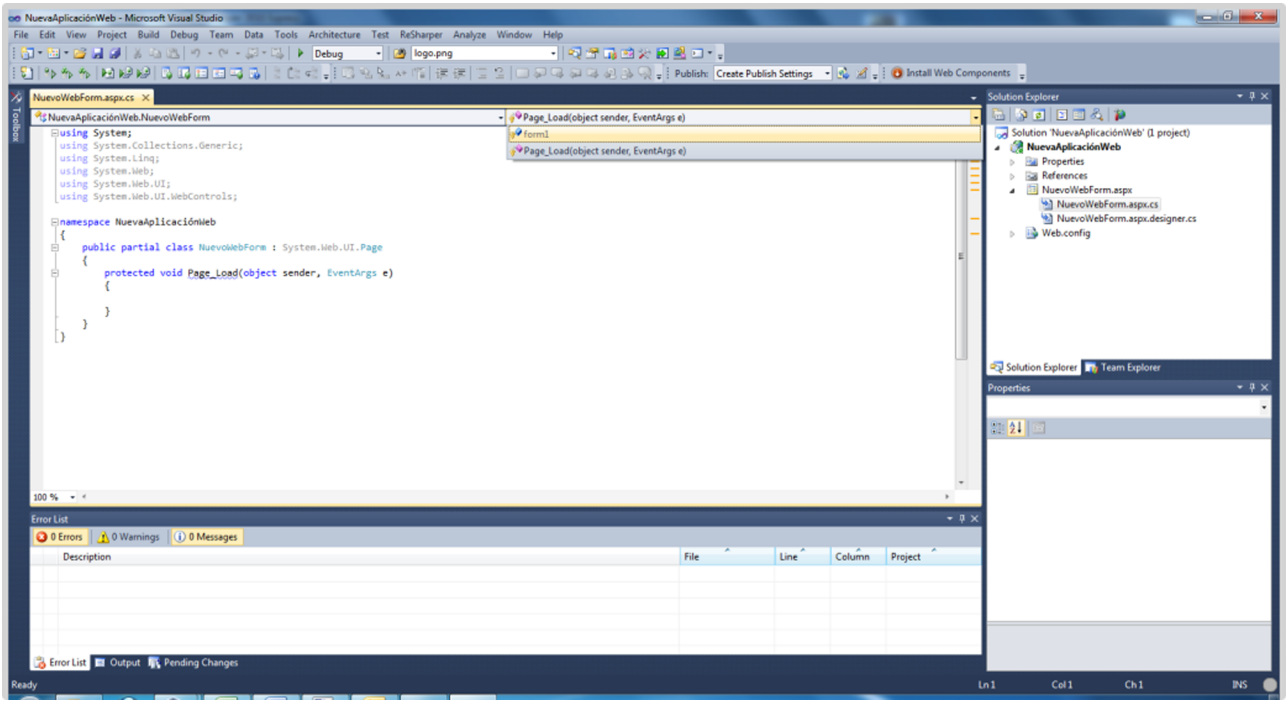
- Fent doble clic sobre un component per a associar codi al seu esdeveniment per defecte.

- Usant el botó *View Code* del *Solution Explorer* tenint triada qualsevol pàgina ASP.NET.

- Fent clic en el botó *Events* de la finestra *Properties* des del dissenyador, per a accedir a la llista d'esdeveniments del component que es tingui seleccionat, i després triant l'esdeveniment al qual es vol associar codi.

L'editor de codi, com es pot apreciar en la figura 19, té dues llistes desplegables en la part superior. La de l'esquerra permet seleccionar qualsevol dels elements existents en la pàgina, moment en el qual la llista de la dreta enumerarà els esdeveniments que s'apliquen a aquest objecte. La selecció d'un esdeveniment provocarà que l'editor generi la capçalera del mètode encarregat de respondre al succés, de manera que només cal introduir les sentències que es volen executar en aquest cas.

Figura 19. L'editor de codi de C#



Igual que l'editor XHTML/CSS, aquest ofereix també una sèrie d'ajudes enfocades a facilitar el treball del programador, que es tradueixen en finestres flotants amb informació sobre els membres de l'objecte el nom del qual s'ha introduït, la llista de paràmetres que accepta el mètode que es pretén invocar, valors que es poden assignar a una propietat, etc.

A diferència del que ocorre amb el codi XHTML/CSS, el codi C# no serà interpretat ni executat durant la fase de disseny de l'aplicació, per la qual cosa no tindrà cap efecte sobre la interfície d'usuari que es construeixi paral·lelament

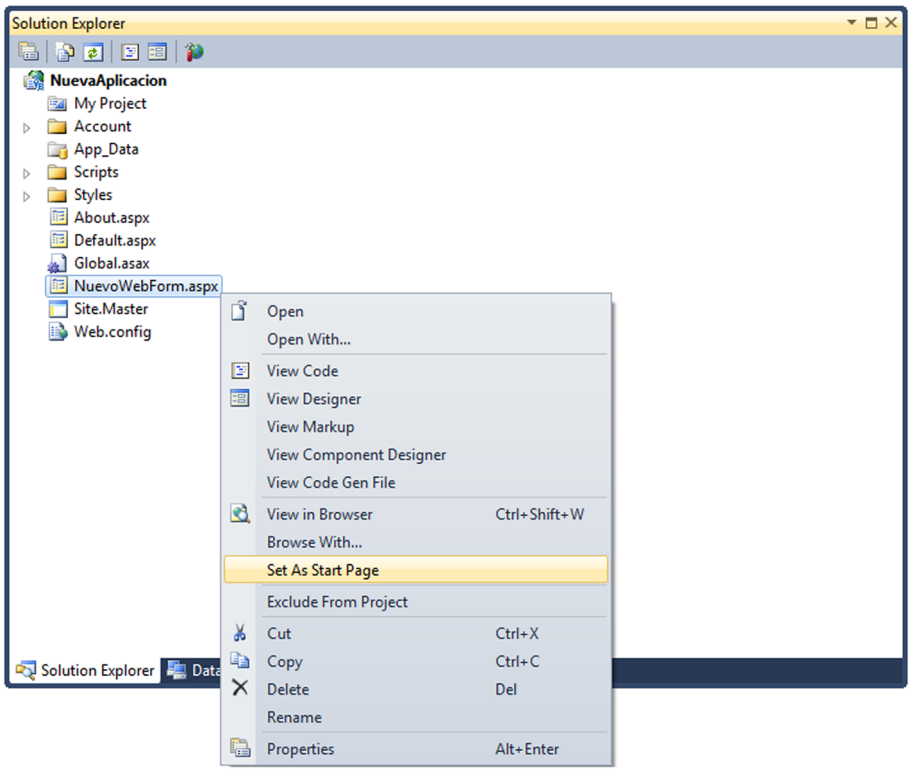
en el dissenyador. Es tracta de codi que es processarà en el moment en què l'aplicació sigui executada, i per aquest motiu s'emmagatzema en un mòdul independent.

3.1.6. Execució i depuració d'un lloc web

Dues de les tasques bàsiques de qualsevol desenvolupador web són l'execució i depuració del lloc en el qual treballa. Amb el Visual Studio, totes dues tasques es fan molt fàcilment, la qual cosa és molt útil per al programador.

Per a executar un lloc web primer hem d'especificar la pàgina en què s'iniciarà l'execució, ja que podem tenir diverses pàgines i en el Visual Studio estan totes al mateix nivell. Per a especificar el punt d'entrada de l'aplicació web premem el botó dret sobre la pàgina en qüestió i seleccionem l'opció *Set As Start Page*.

Figura 20. Establir com a pàgina principal



Ara, quan executem el lloc web automàticament, el Visual Studio carregarà la pàgina seleccionada. Per a executar el lloc web sense depuració, ho podem fer de maneres diferents:

- Seleccionant l'opció del menú *Debug* > *Start Without Debugging*.
- Prement Ctrl + F5.

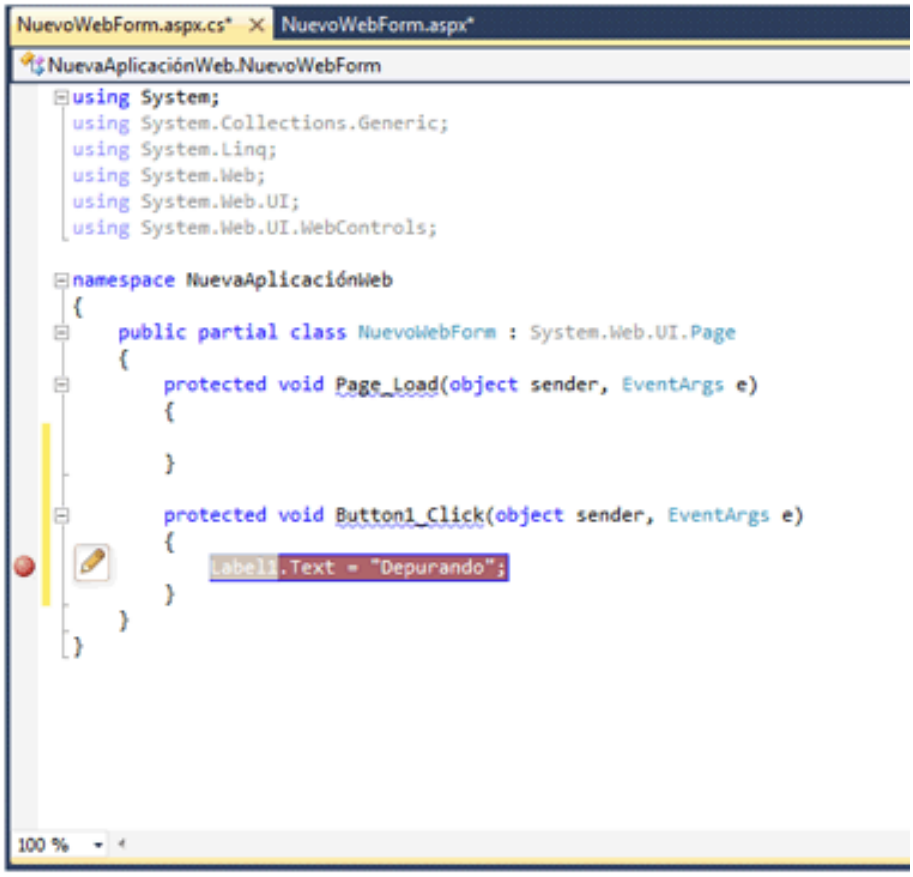
D'aquesta manera podrem veure la nostra aplicació en execució, però no la podrem depurar. Per a executar el nostre lloc web amb depuració disposem de diverses opcions:

- Prement el botó de la barra de menú superior.
- Seleccionant l'opció del menú *Debug* > *Start Debugging*.
- Prement F5.

El Visual Studio disposa de multitud d'eines per a la depuració del nostre codi. Les més importants i útils per al nostre curs són els *breakpoints* i els *watch* de variables.

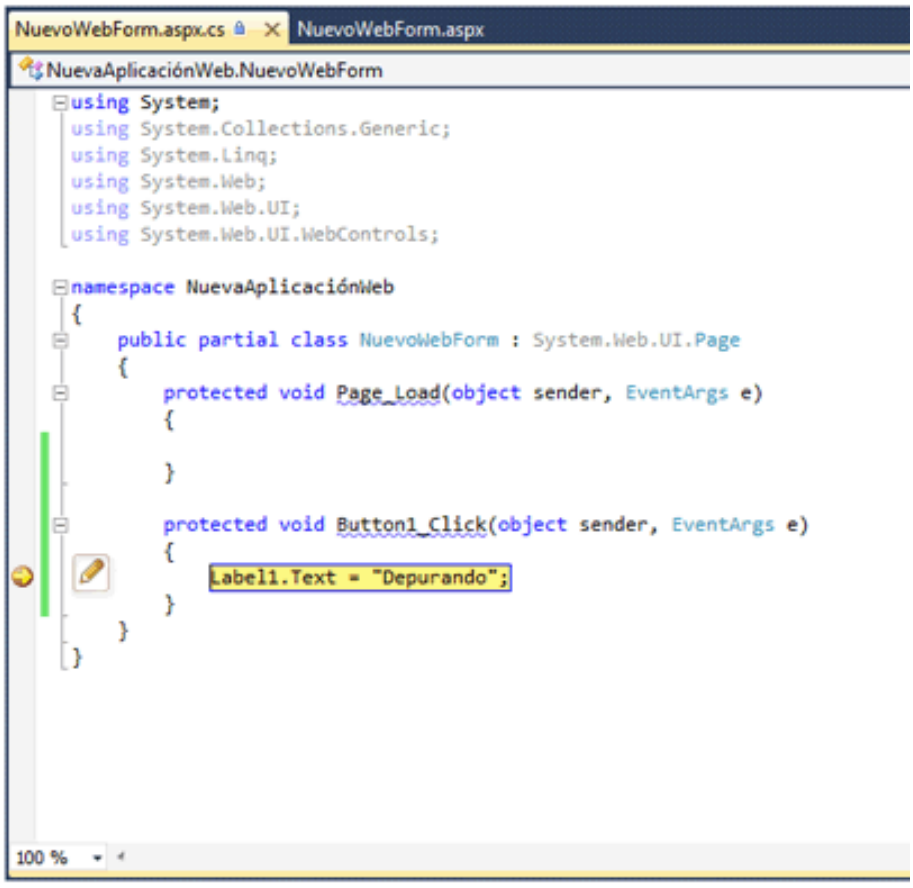
Un *breakpoint*, com ja devem saber, és la col·locació d'una interrupció en el flux d'execució. Per a col·locar un *breakpoint* en el nostre codi, únicament haurem d'anar a l'arxiu `.cs` i prémer amb el botó esquerre del ratolí sobre la columna esquerra de la fila que vulguem (figura 21).

Figura 21. Breakpoint en l'arxiu de codi .cs



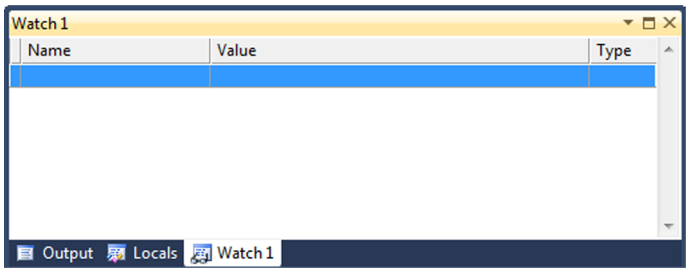
Si llancem l'execució amb depuració del nostre lloc i cliquem sobre el botó podrem veure com el flux es para en *el breakpoint* que hem inserit (figura 22).

Figura 22. Depuració de l'aplicació



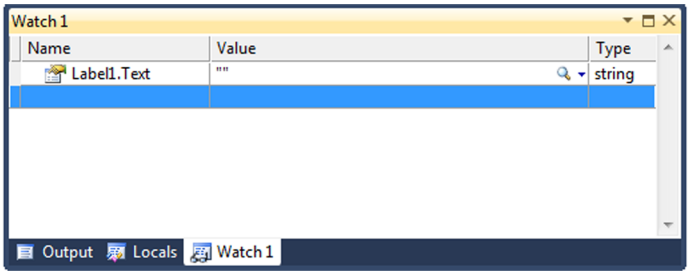
Una vegada el *breakpoint* ha saltat, podem veure el valor de totes les variables que tenim actualment en el nostre context. El valor de les variables el podem consultar en l'apartat inferior *Watch* (figura 23).

Figura 23. Finestra Watch de variables



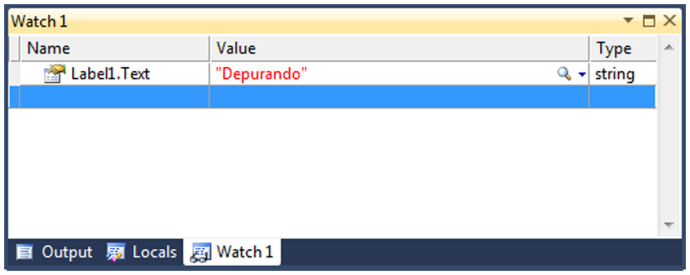
Si fem doble clic sobre la columna *nombre* podem inserir el nom de la variable que volem consultar i automàticament en la columna *valor* s'actualitzarà el contingut d'aquesta variable (figura 24).

Figura 24. Inspecció del valor d'una variable



Per a avançar el flux d'execució una vegada parat en el nostre *breakpoint*, disposem dels botons de la barra de menú superior. Si avancem una línia el flux d'execució veurem com la variable canvia de valor automàticament (figura 25).

Figura 25. Inspecció del valor d'una variable



Finalment, si volem reprendre l'execució normal de l'aplicació, únicament hem de prémer una altra vegada el botó .

3.1.7. Base de dades compacta amb el Visual Studio

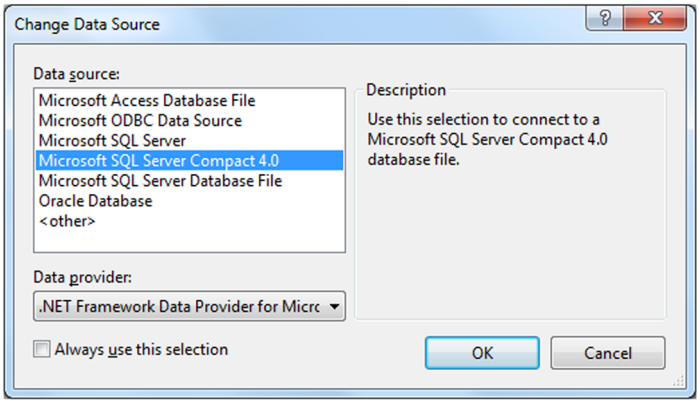
En la majoria d'aplicacions web és necessària una base de dades d'on s'agafa la informació. Per a aplicacions extenses pot ser necessària una base de dades convencional que necessiti un servidor de base de dades per a funcionar, però per a projectes petits com són els nostres, una base de dades compacta (d'un sol arxiu sense necessitat de servidor) és més que suficient. El Visual Studio ens permet la creació de tots dos tipus de base de dades des de l'entorn de programació mateix.

Creació de la base de dades

Nosaltres ens centrarem en la creació d'una base de dades compacta mitjançant el Visual Studio.

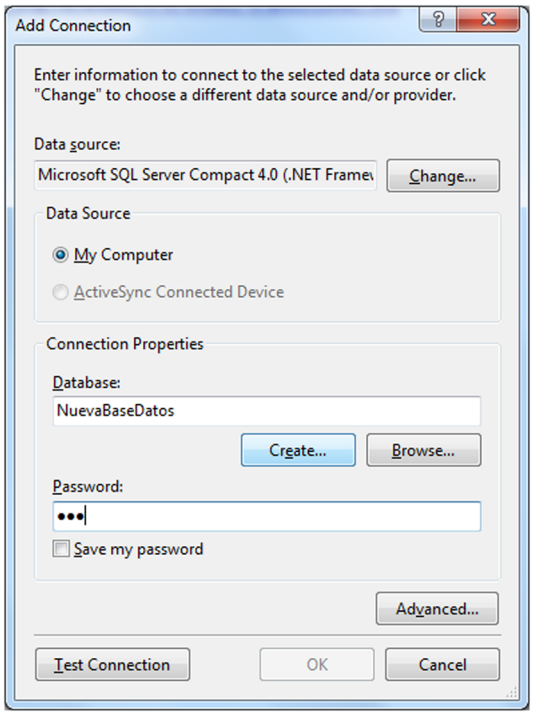
Primer de tot accedirem a l'apartat del menú *Tools* > *Connect to Database*. Seleccionem *Microsoft SQL Server Compact 4.0* i premem *OK* (figura 26).

Figura 26. Tipus de base de dades



En la finestra següent, assignem un nom a la nostra base de dades en l'apartat *Database*, com per exemple "NuevaBaseDatos". En l'apartat *Password* introduïm una contrasenya, com per exemple "uoc", i premem *Create* (figura 27). Una vegada creada la base de dades, premem *OK* i veurem com en el *Database Explorer* ens apareix una nova base de dades *NuevaBaseDatos.sdf* (figura 28).

Figura 27. Afegim la connexió a la base de dades



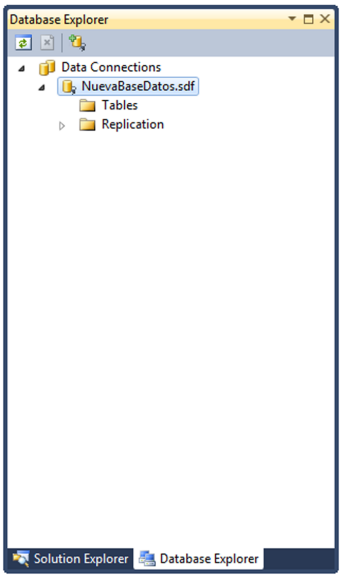
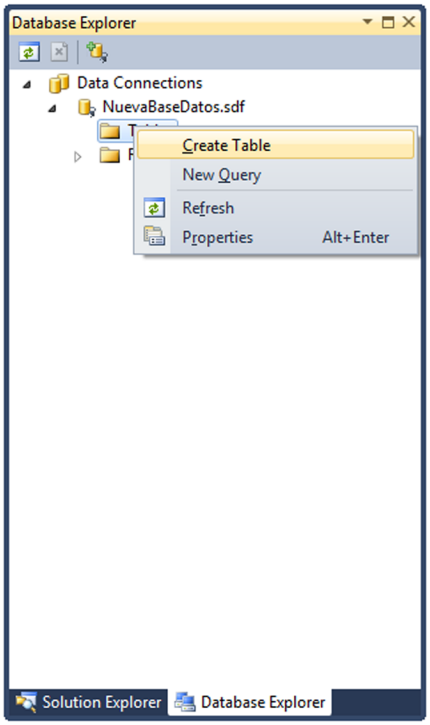


Figura 28. Base de dades compacta

Creació de taules

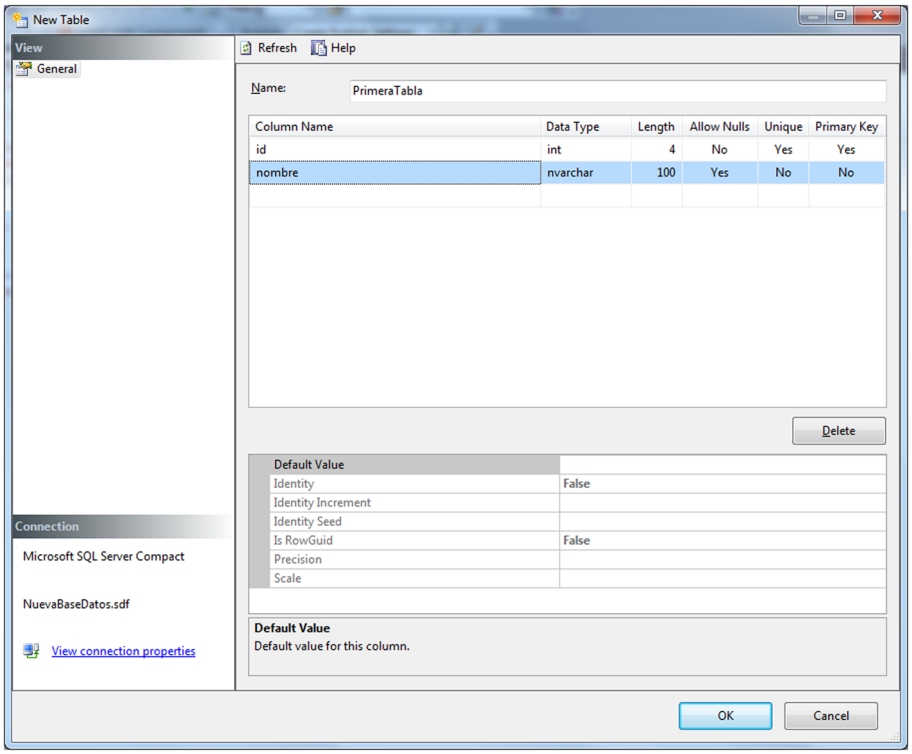
Per a la creació de taules, únicament haurem de prémer botó dret sobre *Tables* i seleccionar *Create Table* (figura 29).

Figura 29. Creació de taules



A continuació ens apareix una finestra en què podrem especificar el nom i els camps de la taula. Veiem que és molt semblant a altres gestors de bases de dades; per a cada columna hem d'especificar el nom del camp, el tipus, si ha de ser únic, si pot estar buit i si és clau primària. Una vegada emplenats els camps premem *OK* (figura 30). Això ens generarà la nostra primera taula (figura 31).

Figura 30. Formulari de taula nova



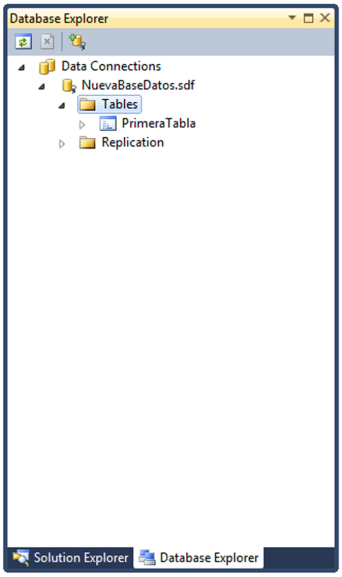
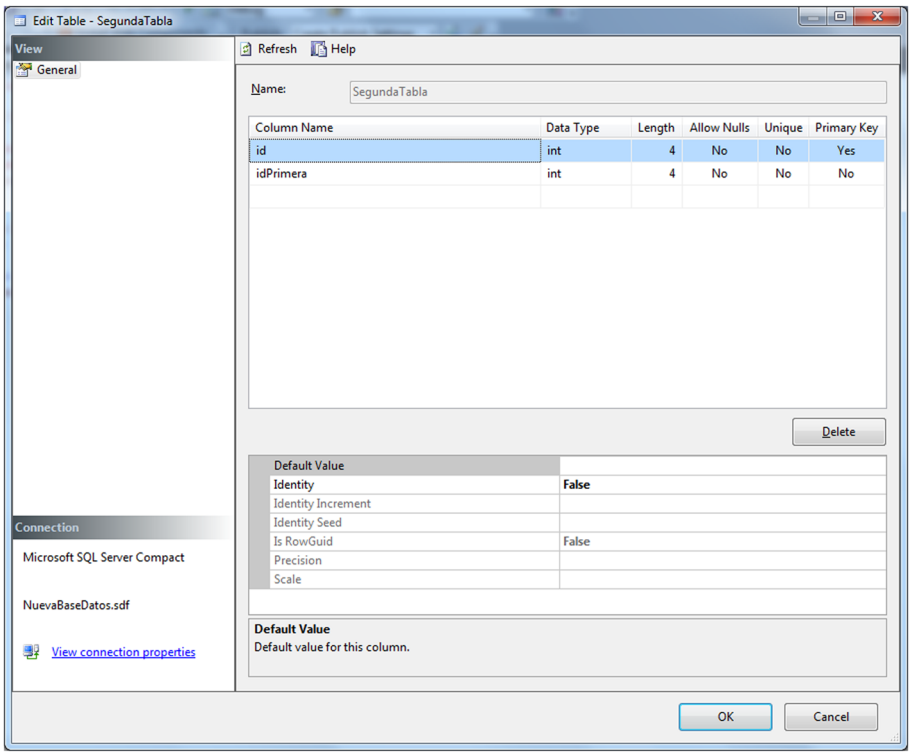


Figura 31. Nova taula

Ens queda poder crear claus foranes entre les nostres taules. Per a això creem una nova taula anomenada "SegundaTabla" (figura 32). Premem el botó dret sobre aquesta taula una vegada creada i seleccionem l'opció *Table Properties* (figura 33).

Figura 32. Edició de taula



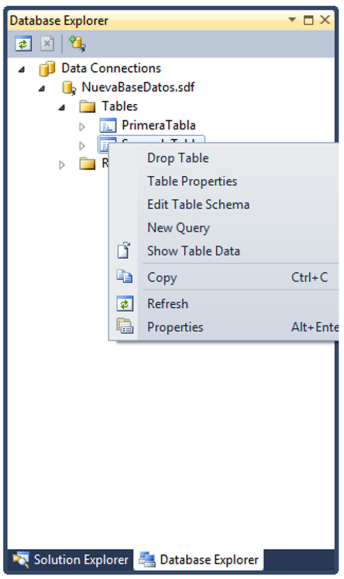
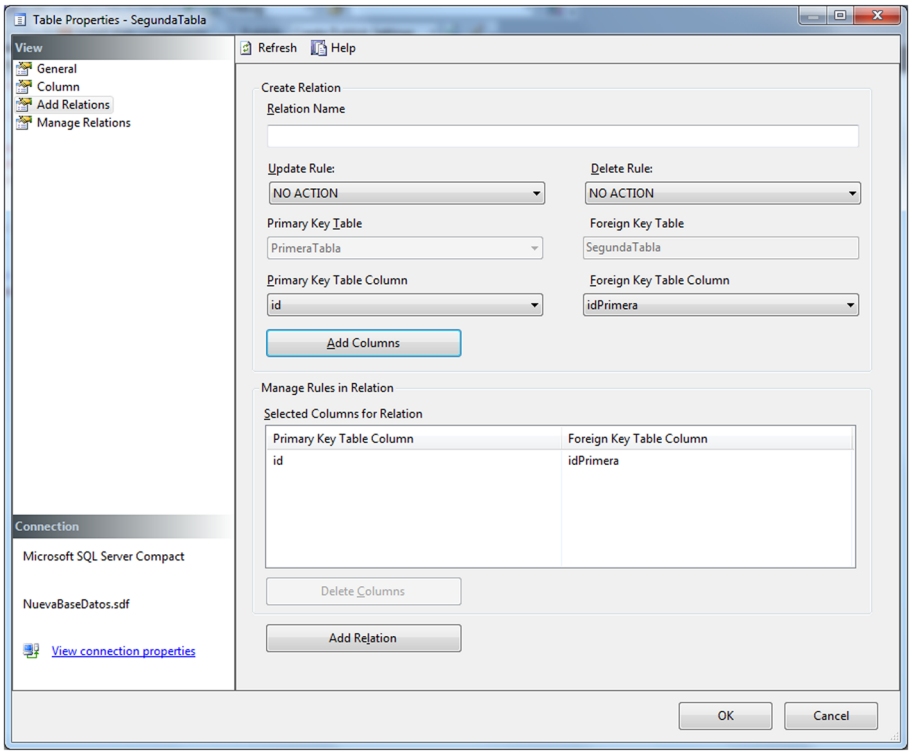


Figura 33. Edició de taula

Dins de les propietats anem a l'apartat *Añadir relaciones* i emplenem els camps d'aquest apartat tenint en compte que la columna esquerra correspon a la clau primària de la taula origen i la columna dreta correspon a la taula actual en què hem d'escollir la columna que serà clau forana. Premem *Add Columns* i automàticament s'afegirà la nova relació de clau forana entre les taules (figura 34).

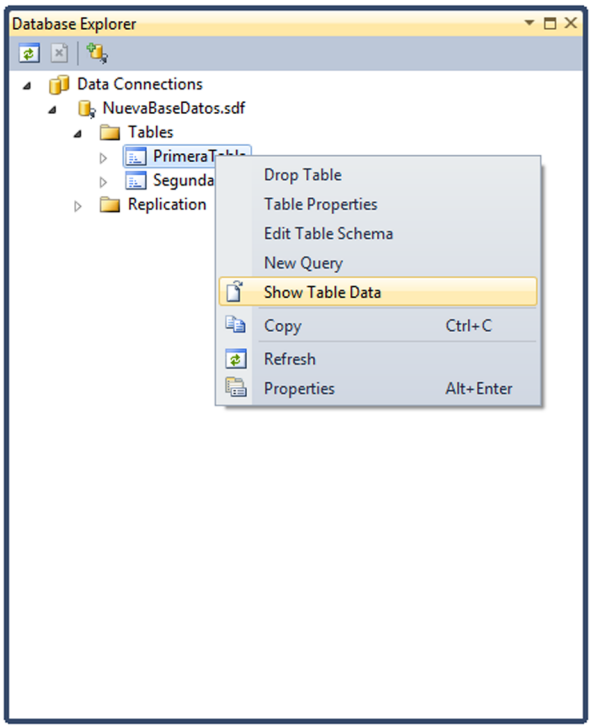
Figura 34. Afegim la clau forana



Inserció i edició de dades

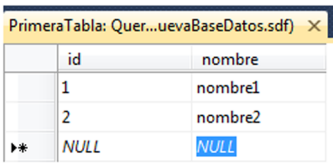
Una vegada creada la base de dades amb les seves taules relacionades, únicament ens queda inserir les dades necessàries. Per a això premem botó dret sobre la taula en què vulguem inserir dades i seleccionem l'opció *Show Data Table* (figura 35).

Figura 35. Afegim dades a la taula



Finalment veurem una taula que podrem editar afegint o modificant files a parer nostre (figura 36).

Figura 36. Afegim dades a la taula



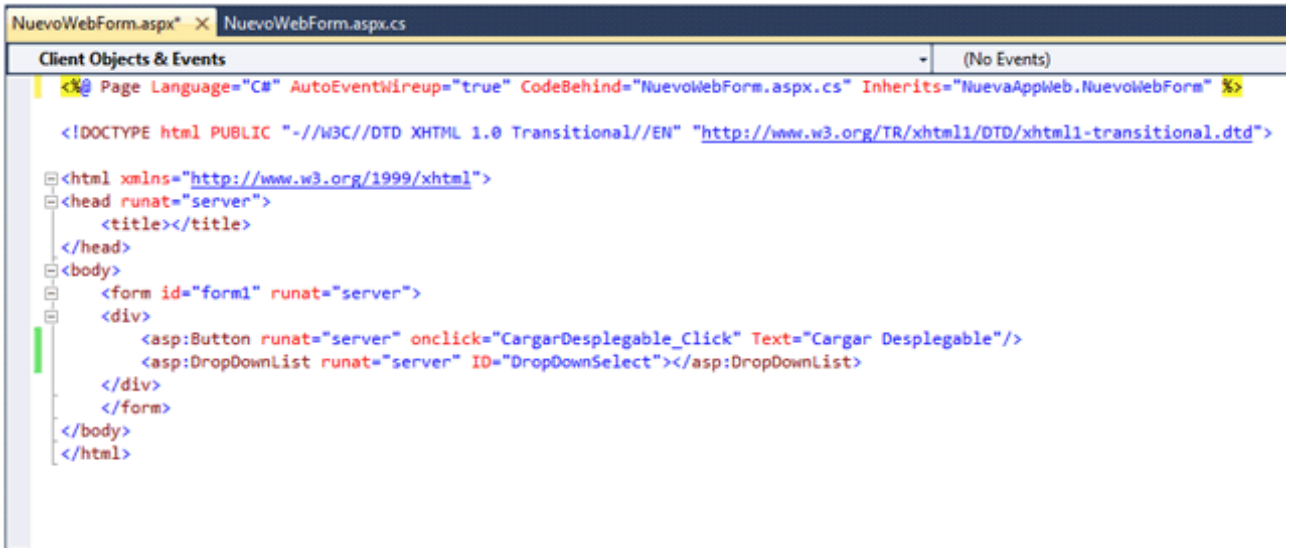
Maneig de la base de dades des de codi

1) Selecció de dades

Per a manejar la base de dades des del codi C#, ja sigui per a inserir, editar o consultar, el primer que hem de fer és obrir una connexió amb aquesta. A continuació veurem un petit exemple en què seleccionarem dades de la base de dades i les carregarem en un desplegable.

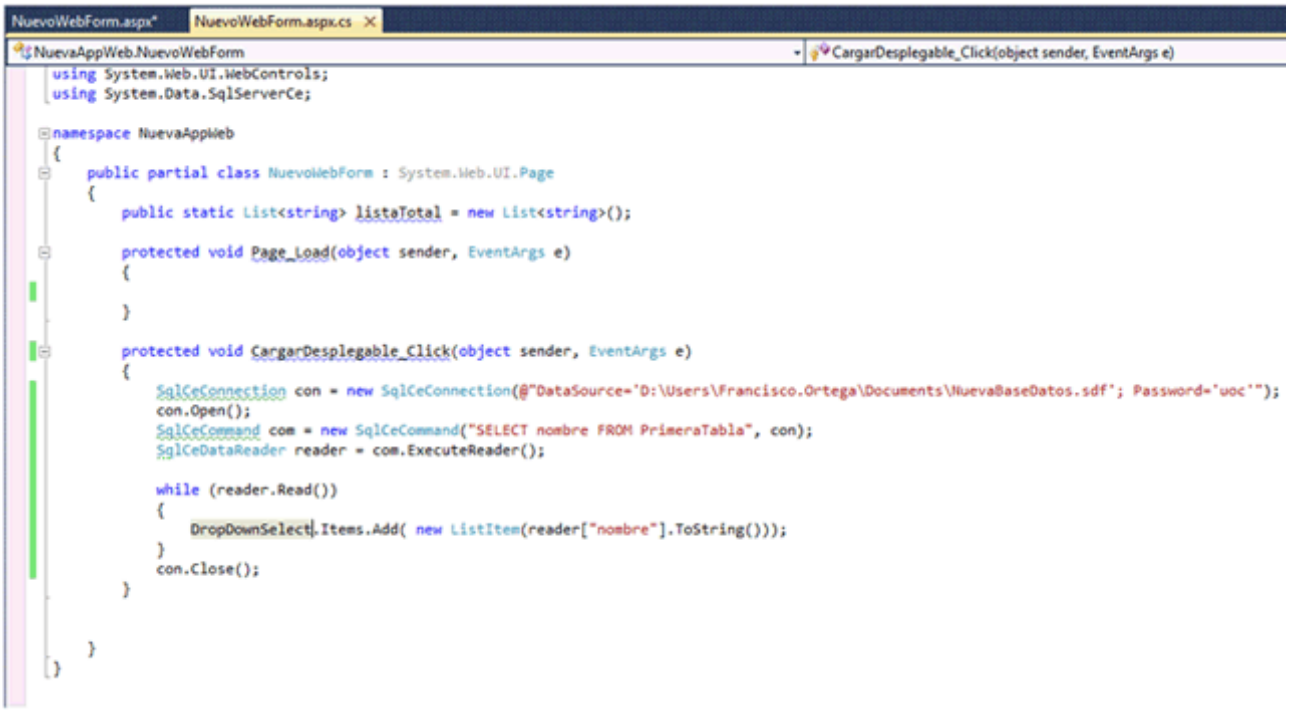
Primer de tot definim en l'arxiu `aspx` el desplegable en què es carregaran les dades i un botó per a llançar l'esdeveniment que consultarà la base de dades.

Figura 37



A continuació en l'arxiu `aspx.cs` definim l'esdeveniment del botó amb el codi de connexió i selecció de base de dades. Vegem como es defineix la biblioteca de SQL Server Compact amb la instrucció `using System.Data.SqlServerCe`. Segons la versió del Visual Studio és possible que no tingueu la biblioteca (*dll*) a la qual es fa referència. La trobareu en l'espai de fitxers de l'aula.

Figura 38



Comentarem línia per línia el codi que podem veure en l'esdeveniment de botó del servidor:

```
SqlCeConnection con = new SqlCeConnection (@"DataSource='D:\Users\Francisco.Ortega\
Documents\NuevaBaseDades.sdf'; Password='uoc'");
```

Primer de tot connectem amb la base de dades amb la classe `SqlCeConnection` i li passem la ruta de l'arxiu de base de dades creat anteriorment i la seva contrasenya.

```
con.Open();
```

És necessari obrir la connexió amb la base de dades per a poder començar a fer-hi operacions.

```
SqlCeCommand com = new SqlCeCommand("SELECT nom FROM PrimeraTaula", con);
```

Creem la consulta que farem en la base de dades amb l'objecte `SqlCommand` i li passarem la connexió que hem creat.

```
SqlCeDataReader reader = com.ExecuteReader();
```

Per a executar la consulta utilitzem la funció `ExecuteReader()`. Aquesta funció ens retorna els resultats de la consulta que emmagatzemem en un objecte de tipus `SqlCeDataReader`.

```
while (reader.Read())
```

```
{
    DropDownSelect.Items.Add( new ListItem(reader["nom"].ToString()));
}
```

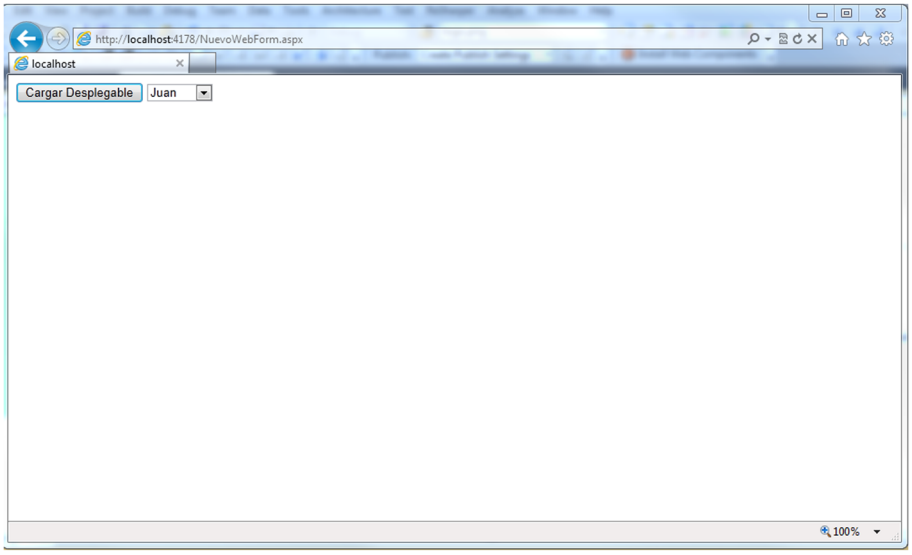
Anem recorrent el `SqlCeDataReader` amb un `while` extraient en cada iteració una fila del resultat amb la funció `Read()`. Una vegada extreta la fila accedirem als camps de la base de dades mitjançant la notació `reader["nomCamp"]`.

```
con.Close();
```

Una vegada fetes totes les operacions amb la base de dades tanquem la connexió per no sobrecarregar la base de dades.

Si executem l'aplicació podem veure com el desplegable es carrega amb la consulta de la base de dades en prémer el botó *Cargar Desplegable* (figura 39).

Figura 39



2) Edició de dades

Per a l'edició de dades de la base de dades, igual que en la selecció, hi haurem d'obrir una connexió.

Continuant amb l'exemple anterior afegirem un `TextBox`, on inserirem el valor que volem que reemplaci la selecció del desplegable.

Figura 40

A més, com podem veure, hem afegit un botó que llançarà l'esdeveniment d'edició del camp del desplegable amb el valor del `TextBox`.

Figura 41

Podem veure que la part de connexió i desconnexió amb base de dades és idèntica a la de l'exemple anterior, i únicament varia la creació i execució de la consulta.

```
SqlCeCommand com = new SqlCeCommand("UPDATE PrimeraTaula SET nom='" +
                   TextBoxEditar.Text + "' WHERE nom='" + itemSelected.Text + "'", con);
```

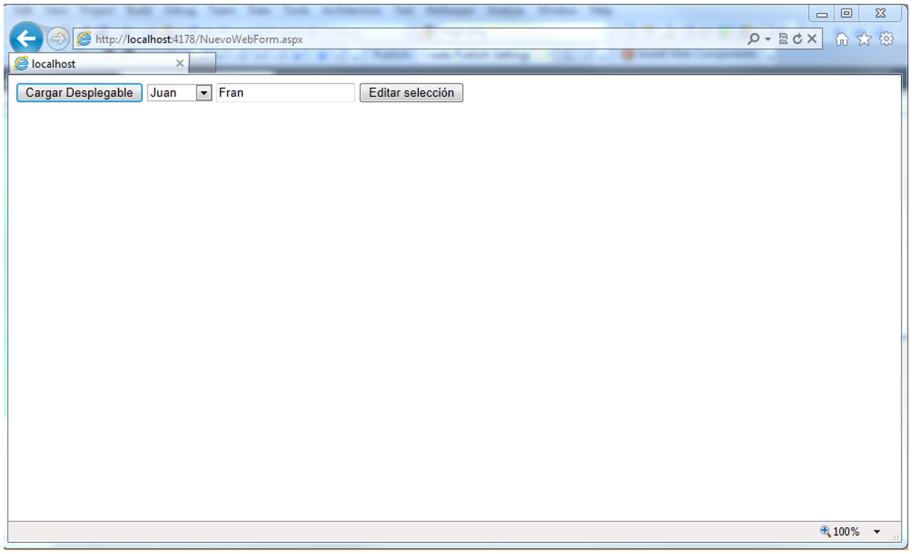
Utilitzant el mateix objecte `SqlCeCommand` creem `query` d'`Update` passant-li la connexió a la base de dades.

```
com.ExecuteNonQuery();
```

Per a una edició de dades s'ha d'utilitzar la funció `ExecuteNonQuery()`, en lloc d'`ExecuteReader()`.

Si executem l'aplicació, veurem com si inserim un valor en el `TextBox`, seleccionem un valor en el `select` i premem *Editar selección*, el valor del desplegable s'actualitza quan tornem a prémer *Cargar Desplegable*.

Figura 42



3) Inserció de dades

La inserció de dades és pràcticament idèntica a l'edició. Continuant amb l'exemple, afegirem un `TextBox` en què especificarem la cadena de caràcters que inserirem en la base de dades.

Figura 43

```
NuevoWebForm.aspx | NuevoWebForm.aspx.cs
Client Objects & Events                                   (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button runat="server" onclick="CargarDesplegable_Click" Text="Cargar Desplegable"/>
        <asp:DropDownList runat="server" ID="DropDownSelect"></asp:DropDownList>

        <asp:TextBox ID="TextBoxEditar" runat="server"></asp:TextBox>
        <asp:Button runat="server" onclick="EditarSeleccion_Click" Text="Editar selección"/>

        <asp:TextBox ID="TextBoxInsertar" runat="server"></asp:TextBox>
        <asp:Button runat="server" onclick="InsertarNombre_Click" Text="Insertar nombre"/>
    </div>
    </form>
</body>
</html>
```

A més, afegim un botó que llançarà l'esdeveniment d'inserció en la base de dades.

Figura 44

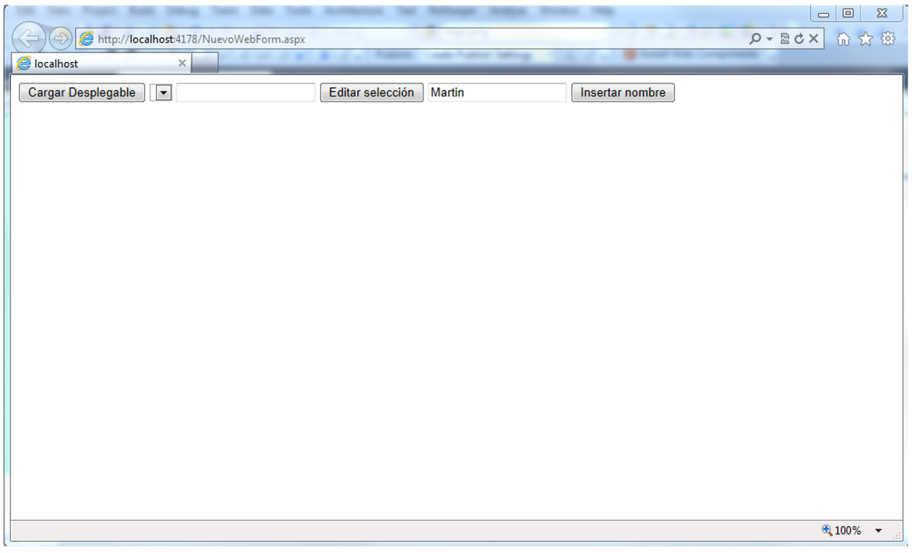
```
        protected void InsertarNombre_Click(object sender, EventArgs e)
        {
            SqlCeConnection con = new SqlCeConnection(@"DataSource='D:\Users\Francisco.Ortega\Documents\NuevaBaseDatos.sdf';
                                                        Password='uoc'");
            con.Open();

            if (!String.IsNullOrEmpty(TextBoxInsertar.Text))
            {
                SqlCeCommand com = new SqlCeCommand("INSERT INTO PrimeraTabla (nombre)VALUES(@nombre)", con);
                com.Parameters.AddWithValue("@nombre", TextBoxInsertar.Text);
                com.ExecuteNonQuery();
            }

            con.Close();
        }

    }
}
```

El codi que varia pel que fa a l'edició de dades és el que fa referència a la creació de la `query` d'inserció.

```
SqlCeCommand com = new SqlCeCommand("INSERT INTO PrimeraTaula (nom)VALUES(@nom)", con);
                   com.Parameters.AddWithValue("@nom", TextBoxInserir.Text);
```

Primer creem la `query` d'inserció especificant els valors que inserirem amb la notació `@nomCamp`. A continuació especifiquem aquests valors amb la funció `Parameters.AddWithValue()` passant-li el `@nomCamp` identificat i el valor que volem inserir. Si executem l'aplicació, inserim un valor en el `TextBox` i premem el botó *Insertar nombre*, veurem com s'insereix la cadena en la base de dades. Per a poder veure-ho, recarregarem la combinació prement *Cargar Desplegable*.

Figura 45



4. Introducció a ASP.NET

ASP.NET és una de les tecnologies que integra .NET mitjançant la qual els desenvolupadors poden crear aplicacions web.

En aquest apartat s'explicaran les característiques més destacables d'ASP.NET, es farà una breu introducció històrica des dels principis d'ASP.NET fins a ASP.NET 4.0. A continuació, es començarà a veure una introducció a la tecnologia ASP.NET, explicant el que són els WebForm, els controls de servidor o les MasterPage.

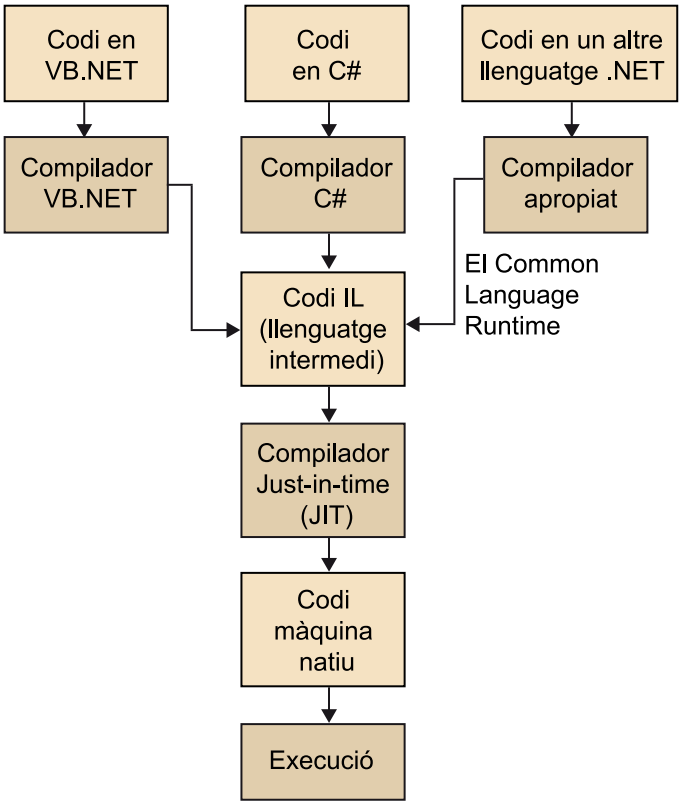
4.1. Característiques principals d'ASP.NET

En ASP.NET hi ha set factors o característiques clau que el diferencien dels altres productes de Microsoft i de les plataformes de la competència. Aquestes característiques són les següents:

1) ASP.NET està integrat amb l'entorn de desenvolupament .NET Framework. L'entorn .NET Framework és un conjunt de milers de classes, estructures, interfícies, etc., que està organitzat segons la seva funcionalitat. La manera d'utilitzar les classes de .NET en ASP.NET és el mateix que en qualsevol altre tipus de plataforma .NET. En altres paraules, la utilització de l'entorn de desenvolupament .NET és la mateixa en una aplicació web que en una d'escriptori. Aquesta característica és veritablement important perquè implica que Microsoft ofereix les mateixes eines als desenvolupadors web que als programadors d'aplicacions client.

2) ASP.NET és compilat, no interpretat. Les aplicacions ASP.NET, igual que totes les aplicacions .NET, sempre es compilen. Aquesta compilació passa per dues etapes ben diferenciades. En la primera etapa, el codi C# es transforma en un llenguatge intermedi anomenat Microsoft Intermediate Language (MSIL). Aquesta etapa permet que les plataformes .NET siguin multillenguatge, és a dir, que es puguin programar en diferents llenguatges de programació com C# o Visual Basic. La segona etapa de compilació transforma el codi MSIL en codi màquina. En la figura 46 podem veure un diagrama amb les diferents etapes de compilació.

Figura 46. Etapes de compilació d'ASP.NET



3) ASP.NET és multillenguatge. Com hem vist, gràcies a la primera etapa, qualsevol plataforma .NET pot ser multillenguatge. Únicament necessitem un compilador que transformi el llenguatge en el qual programem al llenguatge intermedi MSIL. Els dos llenguatges de programació per excel·lència en .NET són C# i Visual Basic.

4) ASP.NET és orientat a objectes. En ASP.NET es poden explotar totes les característiques d'una programació orientada a objectes. Per exemple, es poden crear classes reutilitzables, estandarditzar el codi utilitzant interfícies o estendre classes mitjançant l'herència.

5) ASP.NET és compatible amb tots els navegadors. Un dels principals desafiaments als quals s'enfronta un desenvolupador web és la gran quantitat de navegadors que hi ha en el mercat. ASP.NET aborda aquest problema de manera bastant innovadora: anima els desenvolupadors a utilitzar una sèrie de controls de servidor. Aquests controls són funcionals en clients només si aquest suporten totes les seves característiques.

6) ASP.NET és fàcil de desplegar i configurar. Un dels maldecaps principals als quals s'ha d'enfrontar un desenvolupador és el desplegament d'una aplicació web en un servidor real.

7) Un altre aspecte destacable d'ASP.NET és la **configuraciófàcil**, ja que tota la plataforma està unificada en un sol arxiu `web.config`. En aquest arxiu podem configurar des de permisos d'accés a cadenes de connexió de base de dades.

4.2. L'evolució d'ASP.NET

Quan Microsoft va llançar al mercat ASP.NET ràpidament es va convertir en l'estàndard per a desenvolupament web amb tecnologia .NET i en un fort competidor per a les altres plataformes de desenvolupament web.

Des de llavors, Microsoft ha creat diverses versions d'ASP.NET. En els subapartats següents s'explica com ASP.NET ha anat evolucionant als llarg dels anys.

4.2.1. ASP.NET 1.0 i 1.1

La idea central d'ASP.NET va ser crear un model de pàgines web anomenat *formularis web*. Bàsicament, quan un navegador sol·licita una pàgina ASP.NET, crea en el servidor un objecte `Page` i tants objectes com controls tingui aquesta pàgina. D'aquesta manera, el desenvolupador pot modificar aspectes de la pàgina en temps d'execució.

4.2.2. ASP.NET 2.0

La versió 2.0 va mantenir el mateix model de formularis web i es va concentrar en l'addició de característiques noves d'alt nivell com:

- Pàgines mestres. Les MasterPage o pàgines mestres[5] són plantilles per a la reutilització de codi, com per exemple en encapçalaments o peus de pàgina.

- Navegació. S'inclouen diferents controls de navegació com l'arbre, el mapa del lloc o els fils d'Ariadna.

- Seguretat. Inclou suport per a l'emmagatzematge de credencials d'usuari i nous controls de servidor per a l'accés[6] , el registre d'usuari o el recordatori de contrasenya.

- Origen de dades. La versió 2.0 permet enllaçar controls de servidor amb diferents fonts de dades com bases de dades o XML.

(5)En anglès, *master pages*.

(6)En anglès, *login*.

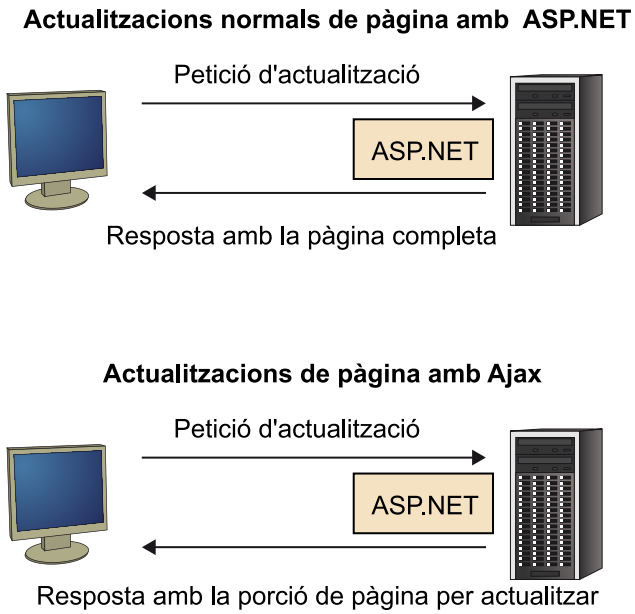
4.2.3. ASP.NET 3.5

La principal novetat que es pot destacar de la versió 3.5 és la introducció de dues noves tecnologies: Linq i Ajax.

La primera tecnologia és un conjunt d'ampliacions per a C# o Visual Basic que permet manipular dades en memòria (per exemple, una llista) de la mateixa manera que ho faríem amb consultes a una base de dades.

En les versions anteriors d'ASP.NET, per a cada petició des del navegador, el servidor havia de processar tota la pàgina sencera i tornar-la a enviar al servidor. Aquest procés resultava lent i ineficient, a més de ser poc interactiu des del punt de vista de l'usuari. Per a solucionar aquest problema es va implementar la tecnologia Ajax per a ASP.NET, que mitjançant JavaScript és capaç de fer recàrregues parcials d'una pàgina web. La figura 47 il·lustra les diferències entre les peticions tradicionals i les peticions Ajax.

Figura 47. Peticions tradicionals i peticions Ajax



4.2.4. ASP.NET 4.0

En la versió 4.0 d'ASP.NET, les noves característiques aportades són una mica més subtils. Alguna són:

- Renderització en XHTML. Després de processar la pàgina corresponent a la petició, s'envia al client en l'estàndard XHTML.

- Control de ViewState. S'incorpora l'activació/desactivació del ViewState. El ViewState és un objecte que ASP.NET va transmetent entre les crides asíncrones en què s'emmagatzema informació de la pàgina.

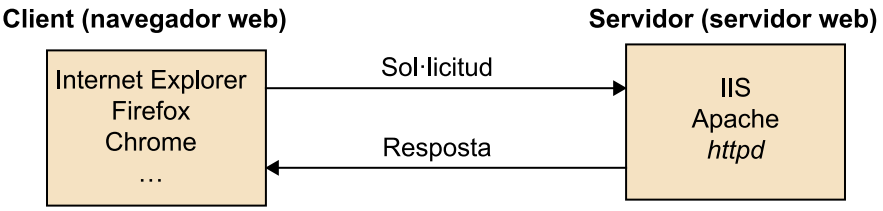
- Control de gràfics. S'incorpora un control per a la realització de diferents gràfics com gràfics de línies, barres, corbes, superfícies, circulars, d'anells, i gràfics de punt.

- Encaminament. Tecnologia per a poder redirigir peticions web igual que fan la majoria dels patrons de disseny model-vista-controlador.

4.3. Estructura d'una aplicació ASP.NET

L'estructura d'una aplicació web clàssica consisteix en una arquitectura client-servidor segons la qual hi ha un servidor i un o diversos clients que hi accedeixen (figura 48).

Figura 48. Arquitectura client-servidor



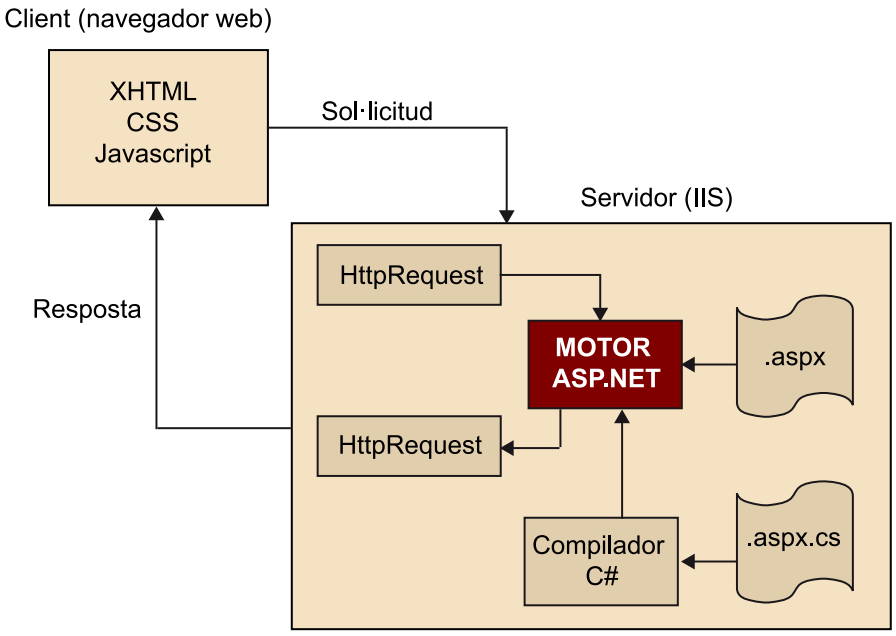
Aquesta estructura és utilitzada també en una aplicació ASP.NET. A més, les aplicacions ASP.NET són executades en el servidor per una capa de programari, coneguda habitualment com a *motor d'ASP.NET*, que és l'encarregada d'interpretar la sol·licitud que li facilita el servidor web i generar un objecte `HttpRequest` amb tota la informació. En aquest objecte estaran contingudes, per exemple, totes les dades introduïdes en els formularis per l'usuari, el navegador utilitzat o la informació d'estat de l'aplicació.

Cada pàgina d'un projecte ASP.NET es compon almenys de dos mòduls: un amb extensió `.aspx`, en el qual està contingut el codi XHTML estàtic, i un altre amb l'extensió `.aspx.cs`, que allotja el codi C# corresponent a la lògica de l'aplicació.

La primera vegada que es rep una sol·licitud per a una pàgina, el motor d'ASP.NET la compila i obté una versió llesta per a executar-se, que s'allotja en memòria. En les sol·licituds següents la resposta serà immediata, ja que es disposa d'una versió compilada.

En la figura 49 es pot veure un esquema de blocs dels elements que formen una aplicació ASP.NET. Com es pot veure, el client sempre obtindrà com a resposta una combinació d'XHTML, CSS i JavaScript, llenguatges vàlids per a qualsevol navegador web amb independència de la plataforma que s'utilitzi per a fer la petició.

Figura 49. Arquitectura ASP.NET



4.3.1. Tipus d'arxius en ASP.NET

Les aplicacions web en ASP.NET poden tenir diferents tipus d'arxius, alguns dels quals es mostren a continuació:

- **`.aspx`**: són les pàgines web en ASP.NET. Contenen la interfície d'usuari. El punt d'entrada per als usuaris sempre és un d'aquests arxius.

- **`.ascx`**: són els controls d'usuari en ASP.NET. Són similars a les pàgines web, però no són accessibles directament des del navegador, sinó que sempre formen part d'una pàgina. S'utilitzen per a reutilitzar codi i evitar-ne així la repetició.

- **`web.config`**: arxiu de configuració en XML. Des d'aquí es configuren aspectes de seguretat, memòria, connexions a bases de dades, etc.

- **`global.asax`**: és el fitxer global de l'aplicació. Aquí es poden definir variables globals, accessibles des de qualsevol pàgina, i esdeveniments propis de l'aplicació.

- **`.cs`**: arxiu de codi C#. Ens permet separar la lògica de l'aplicació de la interfície d'usuari.

A més d'aquests arxius, les aplicacions ASP.NET en poden contenir d'altres de comuns en moltes aplicacions web, com arxius JavaScript, HTML, CSS o imatges.

4.4. Què són els WebForm?

Les Pages o, com oficialment es coneixen, els WebForm, són la part més important d'una aplicació ASP.NET. Aquests WebForm generen tot el codi HTML de resposta a qualsevol petició que sol·licita un client des del navegador. Cada WebForm que afegim a la nostra aplicació serà una pàgina accessible per l'usuari. Un WebForm està compost per dos arxius de codi, un arxiu `.aspx`, on hi ha tot el codi de la interfície d'usuari, i un arxiu `.aspx.cs`, amb tot el codi relacionat amb el WebForm com esdeveniments, validacions, etc.

A continuació podem veure un exemple d'un WebForm en què es mostren tots dos fitxers:

Figura 50. Arxius .aspx i .cs d'un WebForm

NuevoWebForm.aspx

Client Objects & Events

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
         Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

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

NuevoWebForm.aspx.cs | NuevoWebForm.aspx

NuevaAplicaciónWeb.NuevoWebForm

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NuevaAplicaciónWeb
{
    public partial class NuevoWebForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Internament, un WebForm o Page es genera gràcies a la classe `Page`, com hem pogut veure tant en la capçalera del fitxer `.aspx` com en l'herència del fitxer `.cs`. Posteriorment es dedicarà un apartat a explicar més profundament aquesta classe, una de les més importants de la biblioteca de classes de .NET.

4.4.1. Afegir un WebForm al nostre lloc web

Per a afegir un WebForm a un lloc web creat prèviament, únicament hem de seguir els passos següents:

- Premem el botó dret sobre l'arrel del projecte, o sobre qualsevol carpeta en què vulguem que s'allotgi el WebForm, per a obrir el menú contextual, i a continuació premem *Add* > *New item*.

- A continuació seleccionem la plantilla *WebForm*, donem un nom a la nostra pàgina i premem *Add*.

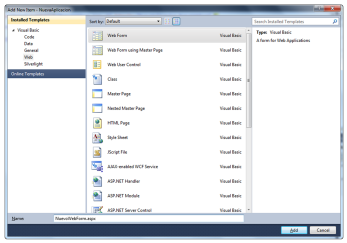


Figura 51. Afegim un WebForm

- Si despleguem en el *Solution Explorer* el nostre WebForm podem veure els arxius `.aspx` i `.aspx.cs`.

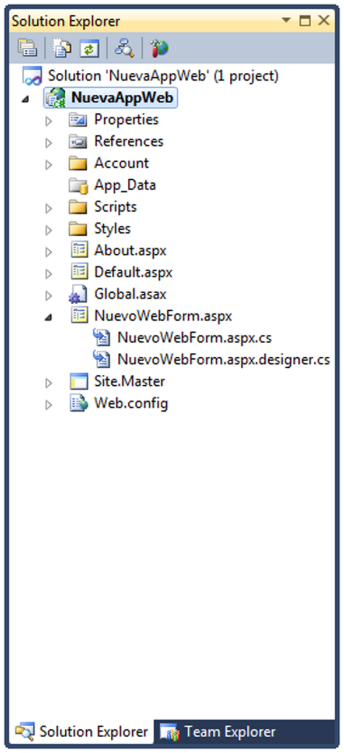


Figura 52. Afegim un WebForm

- Finalment podem explorar tots dos arxius en l'Editor de codi corresponent fent doble clic en cadascun.

4.5. La classe `Page`

Cada pàgina d'una aplicació web en ASP.NET hereta de la classe `Page`. Mitjançant aquesta herència les nostres pàgines adquireixen una sèrie de propietats i mètodes que podem utilitzar en els fitxers `.cs`.

A continuació enumerarem alguna de les propietats més importants:

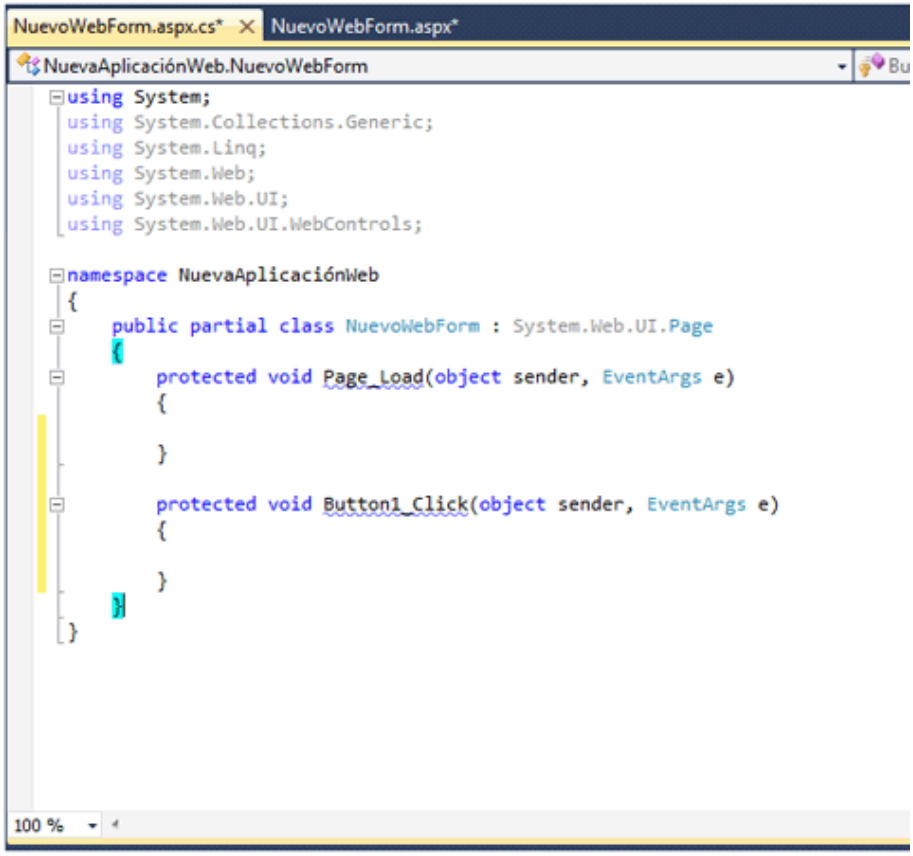
- **`IsPostBack`**: booleà que indica si és la primera vegada que la pàgina s'ha carregat.
- **`EnableViewState`**: booleà. Ens permet decidir si volem desar la informació d'estat dels controls entre peticions.
- **`Application`**: col·lecció que conté informació compartida entre tots els usuaris del lloc web.
- **`Session`**: col·lecció que conté informació d'un usuari en particular que es manté entre diferents peticions.
- **`Request`**: col·lecció que conté informació sobre la petició actual, com informació del navegador o els paràmetres enviats.
- **`Response`**: objecte `HttpResponse` relacionat amb la resposta a una petició determinada. Es pot utilitzar per a encaminar l'usuari a una altra pàgina, enviar *cookies* al navegador, etc.

Un cas molt comú en què s'utilitza una propietat de la classe `Page` és en l'encaminament del flux cap a una altra pàgina web. Concretament, s'utilitza la propietat `Response`.

Per a veure'n un exemple inserirem en un WebForm únicament un Button amb el seu esdeveniment de servidor respectiu.

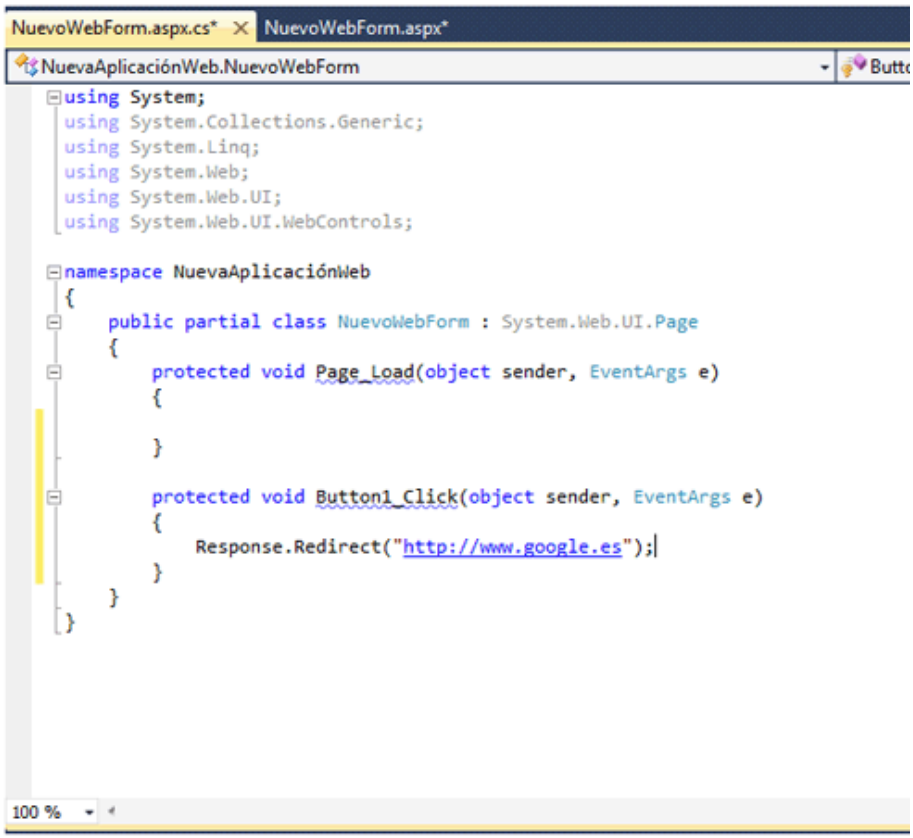
Figura 53. Codi .aspx d'un WebForm

Figura 54. Codi .cs d'un WebForm



En l'esdeveniment `click` del botó fem un encaminament cap a una altra pàgina web de la manera següent:

Figura 55. Codi .cs d'un WebForm



Si ho executem, veurem que quan premem en el botó (figura 56) el flux s'encamina cap a l'URL que hem posat en el `Response.Redirect` (figura 57).

Figura 56. Execució d'un WebForm

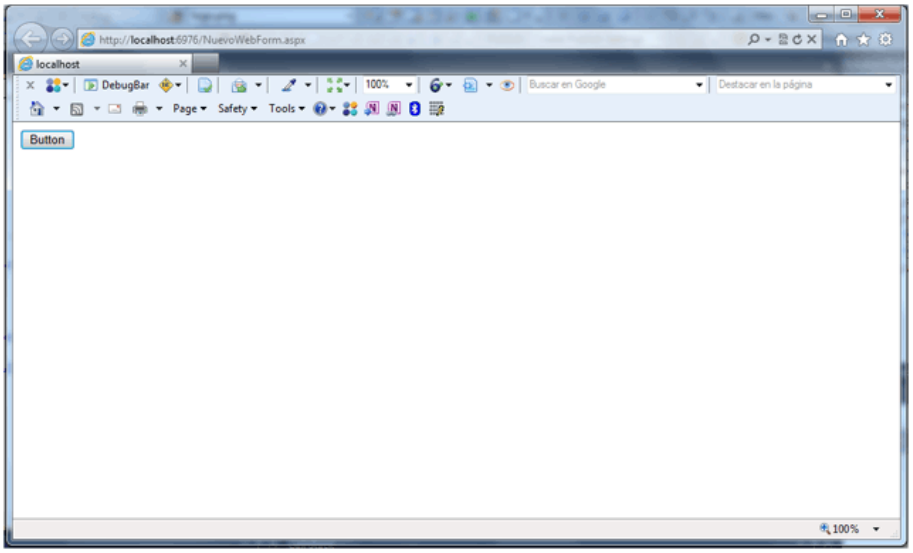
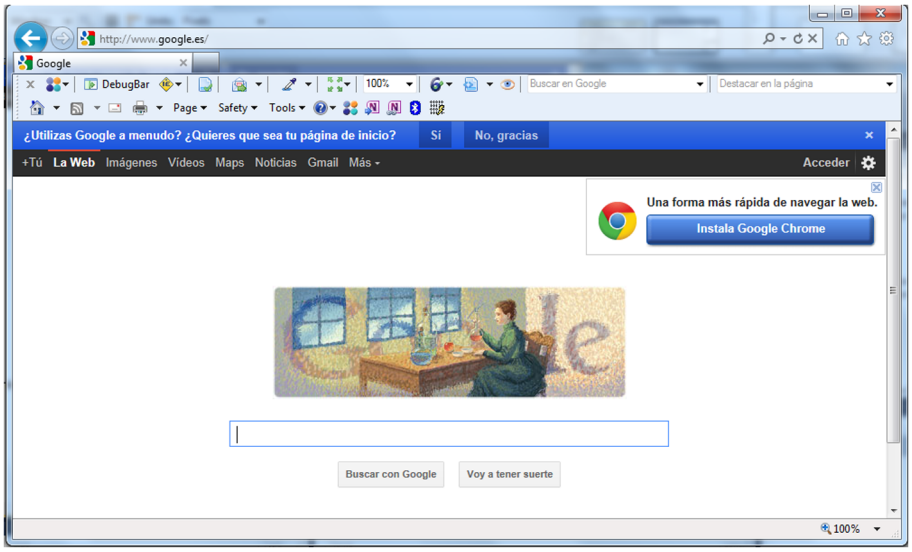


Figura 57. Execució d'un WebForm



Aquest exemple ens pot servir per a fer encaminament entre les pàgines del nostre lloc i així poder navegar-hi.

4.6. El ViewState

En una aplicació d'escriptori de Windows, una part de la memòria de l'ordinador és utilitzada per a emmagatzemar-ne l'estat. En una aplicació web, el funcionament és totalment diferent. Un client es connecta al servidor web, sol·licita una pàgina i la connexió finalitza quan la pàgina es lliura, i es descarten tots els objectes que hi hagi en memòria. Per aquesta raó, sorgeix el problema de l'emmagatzematge de certa informació de l'estat de l'aplicació entre diferents peticions d'un client.

Una de les maneres d'emmagatzemar aquesta informació en ASP.NET és fer-ho en el ViewState. ASP.NET insereix automàticament el ViewState com un camp ocult en cada renderització HTML d'una pàgina. D'aquesta manera es conserven les dades entre diverses peticions de servidor.

A continuació (figures 58, 59 i 60), podem veure un exemple d'un comptador que emmagatzema les vegades que un usuari prem un botó.

Figura 58. Codi .cs d'un ViewState

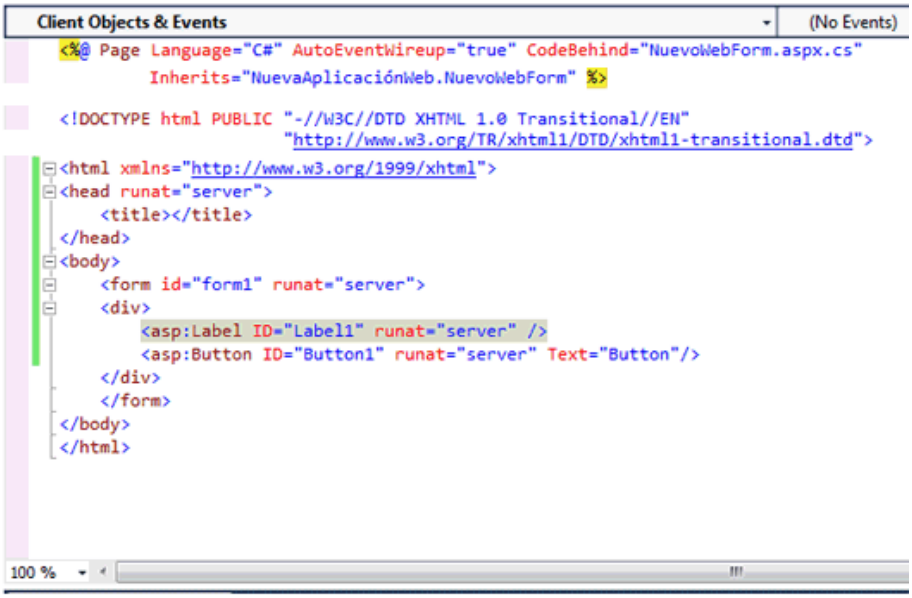


Figura 59. Codi .cs d'un ViewState

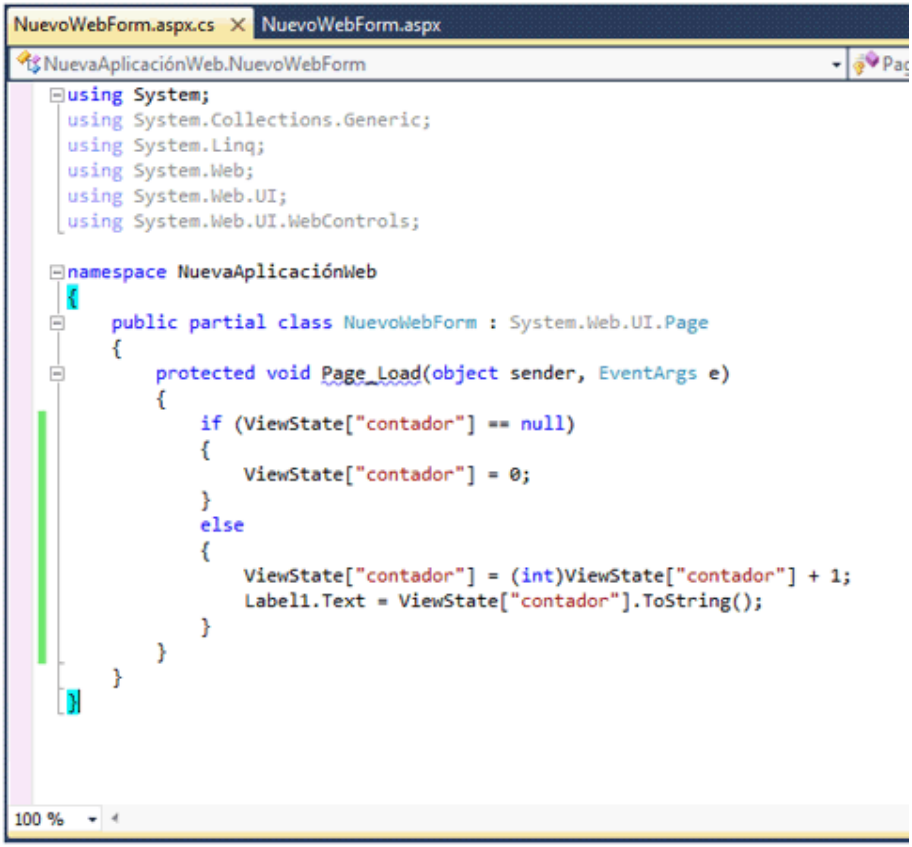
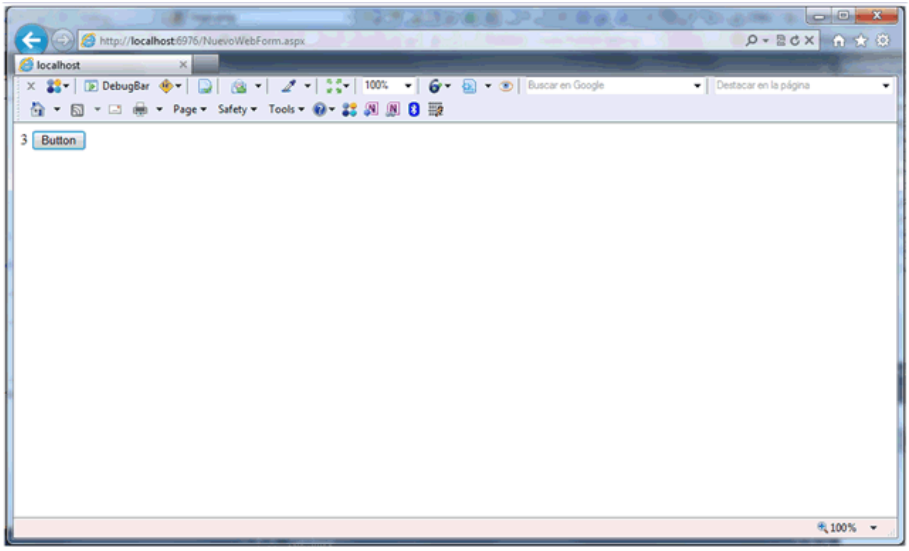


Figura 60. Execució d'un ViewState



4.7. Controls de servidor

ASP.NET va introduir fa diversos anys un model innovador per a la creació de pàgines web. Anteriorment, per al disseny de pàgines web dinàmiques s'introduïen etiquetes en el fitxer HTML mateix i, segons la interpretació d'aquestes etiquetes, es produïa una sortida d'HTML o una altre. Aquesta tècnica, a més de resultar tediosa, produïa un codi poc extensible i reaprofitable.

ASP.NET va solucionar aquest problema introduint el concepte de *controls de servidor*.

Els **controls de servidor** es creen i es configuren com a objectes. Una vegada configurats per part del desenvolupador generen automàticament el seu codi HTML propi.

Algunes característiques dels controls de servidor són:

- Generen la seva interfície d'usuari pròpia. Es genera el codi HTML en temps d'execució i s'envia al client.

- Retenen el seu estat. Desen informació d'ells mateixos entre diferents peticions.

- Disposen d'esdeveniments de servidor. Per exemple, un botó pot executar una funció de servidor en ser premut.

ASP.NET ofereix una gran quantitat de controls de servidor, que es poden classificar en diferents grups. Tots els trobarem en el Toolbox de Visual Studio. Alguns dels grups més importants són:

a) Controls de servidor HTML: grup en què trobem els elements HTML estàndard com els enllaços (`<a>`), els *inputs* (`<input>`), etc. Per a convertir un element HTML estàndard en un control de servidor, únicament hem d'indicar l'atribut `runat=server`.

b) Controls web: controls amb les mateixes funcionalitats que alguns elements HTML, però que disposen de propietats i mètodes que en faciliten la declaració i accés. Alguns exemples són Hiperlink, ListBox, Button, etc.

c) Controls Rich: controls que amplien les funcionalitats dels elements HTML i fins i tot són capaços de generar codi JavaScript propi. Un exemple és el TreeView.

d) Controls de validació: controls que permeten aplicar normes o regles a les entrades de dades dels usuaris.

e) Controls de dades: controls dissenyats per a mostrar grans quantitats de dades i per a l'edició, ordenació i paginació d'aquestes. L'exemple més clar és el control GridView.

f) ASP.NET Ajax Controls: aquests controls permeten al programador utilitzar la tecnologia Ajax sense necessitat d'escriure codi JavaScript en el costat del client.

4.7.1. Controls de servidor HTML

Com hem dit, els controls HTML són exactament iguals que els elements HTML comuns. El podem trobar tots en el Toolbox, dins de l'apartat *HTML* (figura 61).

Si arrosseguem qualsevol control HTML i el deixem anar en el dissenyador de pàgines veurem com automàticament es genera el codi HTML corresponent.

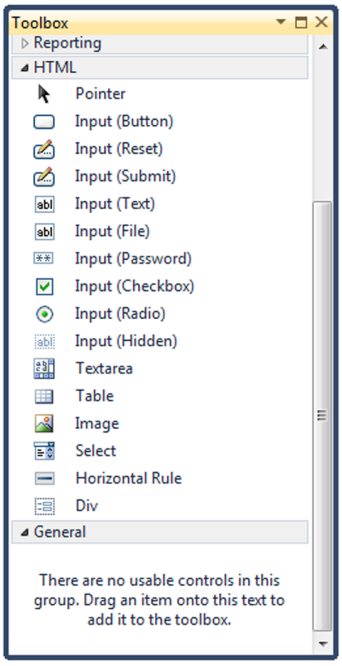


Figura 61. Controls HTML

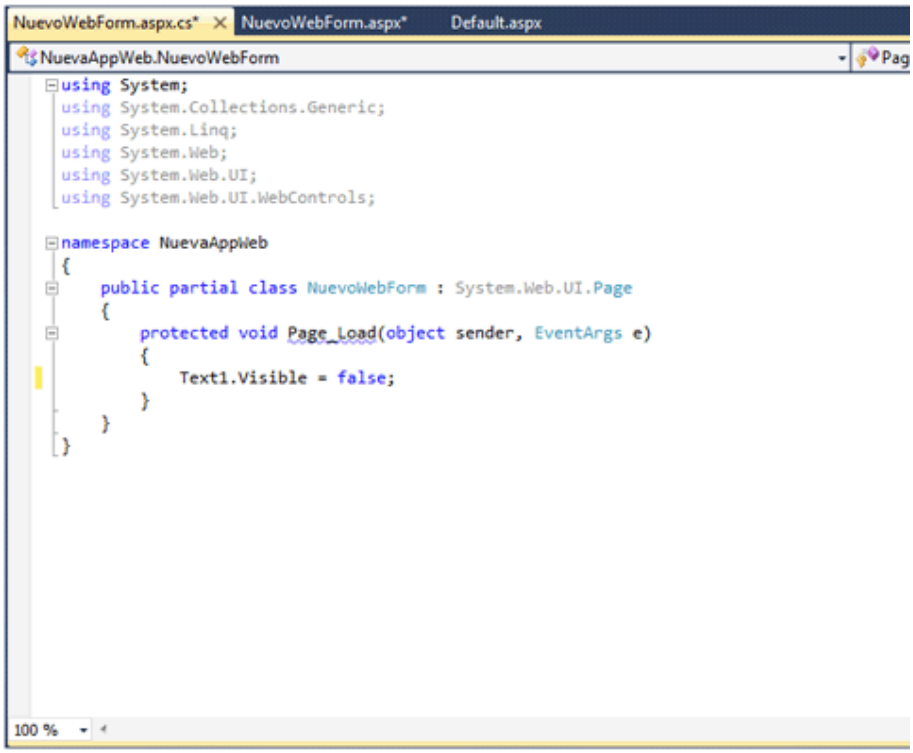
Figura 62. Codi .aspx d'un control HTML

Per a convertir l'element HTML en un control de servidor, únicament hem d'afegir l'atribut `runat = server`.

Figura 63. Codi .aspx d'un control HTML

Des d'aquest moment hi podrem accedir des de l'arxiu de codi (`.cs`) mitjançant el seu `id` i modificar-ne les propietats.

Figura 64. Codi .cs d'un control HTML



Si executem l'aplicació, podrem veure que el nostre element no és visible, ja que hem aplicat la propietat `Visible = false`.

4.7.2. Controls web

Els controls de servidor HTML proporcionen una manera relativament ràpida de migrar HTML a ASP.NET, però no és necessàriament la millor. D'una banda, els controls HTML i els seus atributs no sempre tenen uns noms intuïtius, i tasques com aplicar estil al control poden resultar tedioses.

Per a resoldre aquests problemes ASP.NET posa a disposició nostra els controls web, que bàsicament generen el mateix HTML que els controls de servidor HTML, però disposen d'una configuració molt més senzilla.

Els controls web estan disponibles en el Toolbox dins de l'apartat *Standard* (figura 65).

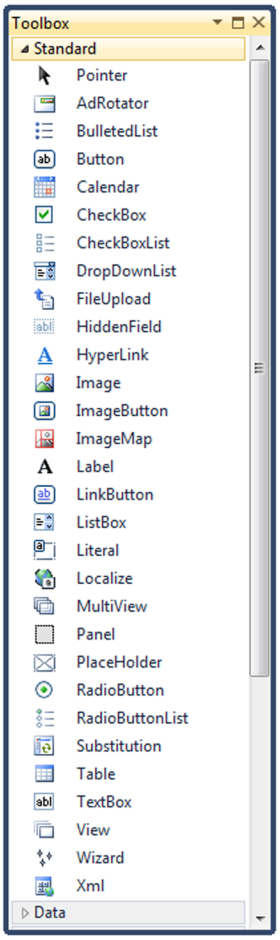


Figura 65. Controls web

Si arrosseguem, per exemple, el control web Button i el deixem anar en l'editor de codi veurem que, com amb els controls HTML, es genera un bloc de codi per defecte.

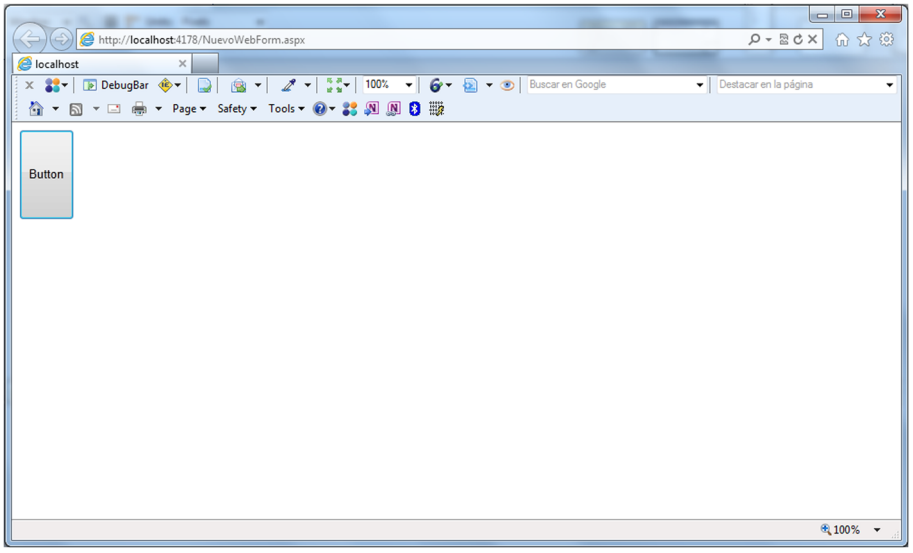
Figura 66. Codi .aspx d'un control web

```
Client Objects & Events                                   (No Ev
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
         Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Button" />
    </div>
    </form>
</body>
</html>

100 %
```

Si modifiquem una de les propietats, com per exemple `Height`, i executem el lloc web, veurem com automàticament s'aplica el valor que hem indicat a l'alçada del botó.

Figura 67. Codi .cs d'un control web

Figura 68. Execució de l'exemple d'un control web



Com hem pogut observar en l'exemple dels controls web, podem accedir a les propietats des del dissenyador de pàgines, mentre que en l'exemple dels controls HTML ho fèiem des de l'arxiu adjunt `.cs`. Aquest és un altre dels avantatges d'utilitzar els controls web en lloc dels controls HTML.

4.7.3. Controls Rich

Els controls Rich o *Rich controls* ens aporten complexes interfícies d'usuari. Aquests controls ens ofereixen funcionalitats molt més àmplies que els controls HTML o web. En realitat cada control Rich és un conjunt de controls web que utilitzem com un de sol. L'exemple més clar és el del control Calendar, que està compost per Buttons, Labels o Hyperlinks.

Els controls Rich els trobem en l'apartat *Standard* del Toolbox. Concretamentn alguns dels controls Rich més importants són:

- Calendar: calendari que permet navegar entre mesos, dies o anys.
- Multiview i View: permeten implementar diferents vistes en una mateixa pàgina i navegar-hi.
- AdRotator: bàner per a mostrar una sèrie d'imatges.

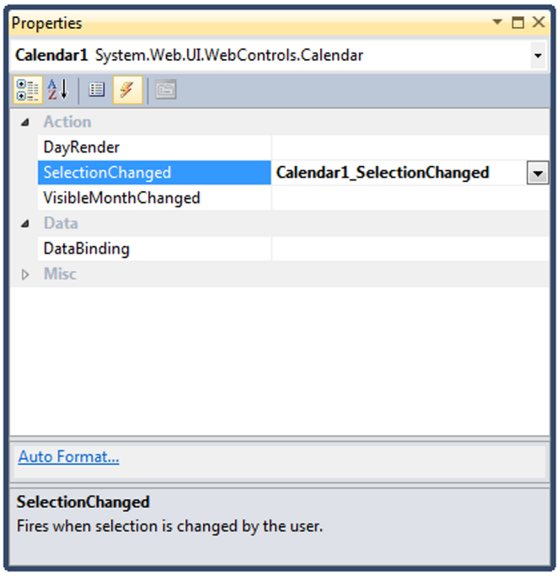
De nou arrossegarem un control d'aquest tipus, concretament un Calendar, i el deixarem anar en el dissenyador. Veiem que, encara que internament està compost per diversos controls web, únicament es genera una etiqueta Calendar (figura 69). Fem el mateix amb un control web Label, que ens servirà per a mostrar el dia seleccionat.

Figura 69. Codi .aspx d'un control Rich

Igual que en els controls web, podem modificar propietats o afegir esdeveniments tant des del dissenyador com des de l'arxiu adjunt `.cs`.

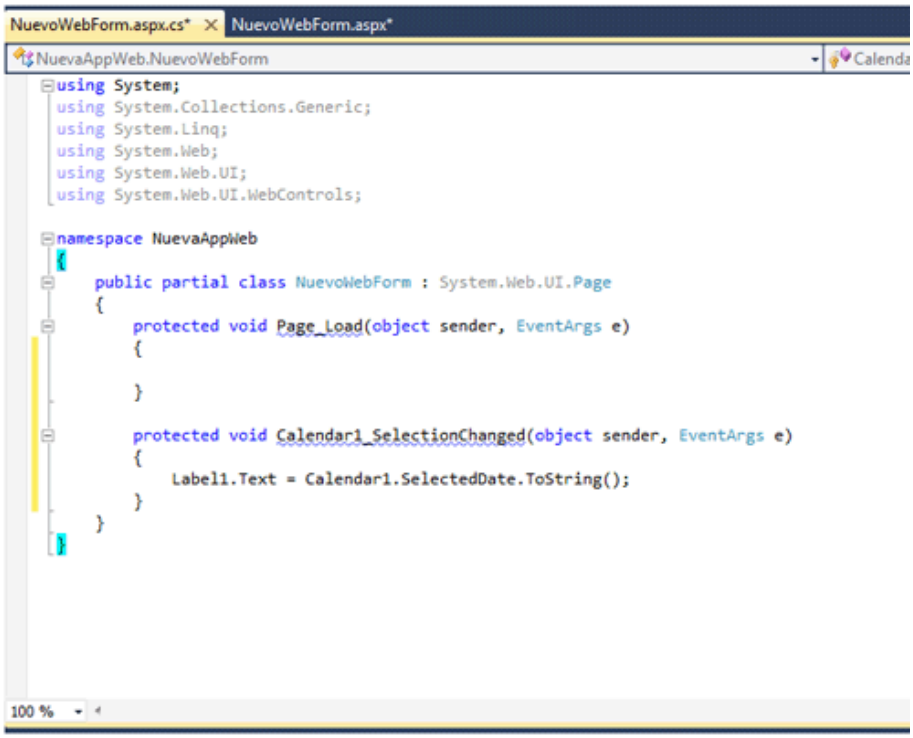
A continuació, en el dissenyador, en la finestra de propietats des de l'apartat d'esdeveniments, fem doble clic sobre l'esdeveniment `SelectionChanged`. Això ens generarà automàticament un esdeveniment que es dispararà en canviar el dia seleccionat.

Figura 70. Finestra de propietats del control



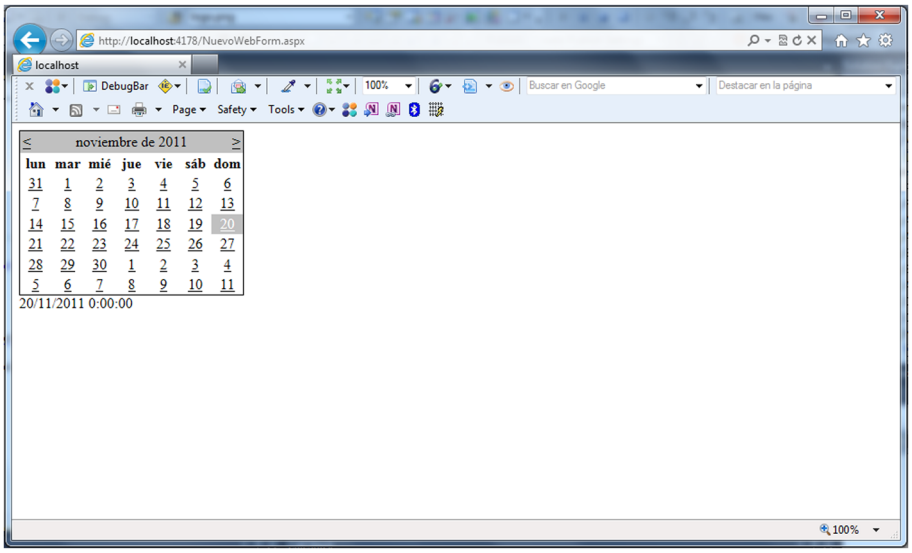
Dins de la funció `SelectionChanged` escrivim el codi que ens actualitzarà el control web Label amb la data seleccionada.

Figura 71. Codi .cs d'un control Rich



Executem i comprovem que mentre anem canviant la selecció, l'etiqueta es va actualitzant correctament.

Figura 72. Execució de l'exemple d'un control Rich



4.7.4. Controls de validació

Un de les funcionalitats més comuna de les pàgines web és recollir dades inserides pels usuaris. Sovint, una pàgina web demana dades a un usuari, que finalment seran desades en una base de dades. En la majoria de casos aquestes dades han de ser validades per a assegurar que no s'emmagatzema informació incoherent. Idealment, la dada entrada per l'usuari ha de ser validada en el client, mitjançant JavaScript, i en el servidor, en el nostre cas mitjançant ASP.NET.

Per no haver d'escriure el codi de validació manualment, que seria una tasca bastant laboriosa, ASP.NET disposa de sis controls de validació:

- `RequiredFieldValidator`: prova si el control relacionat està buit quan s'envia el formulari.
- `RangeValidator`: comprova que l'entrada de l'usuari estigui dins d'un rang determinat.
- `CompareValidator`: prova que el valor del control compleixi una condició determinada (més gran que, més petit que, igual que).
- `RegularExpressionValidator`: comprova que el valor d'un control determinat compleixi una expressió regular.
- `CustomValidator`: permet al desenvolupador especificar una funció client i una de servidor que validaran l'entrada de l'usuari.
- `ValidationSummary`: proporciona una regió amb els errors de tots els validadors de la pàgina.

Aquests controls fan la major part del treball i el programador solament els ha de configurar perquè funcionin correctament. Podem trobar tots aquests controls en el Toolbox dins de la secció *Validation* (figura 73).

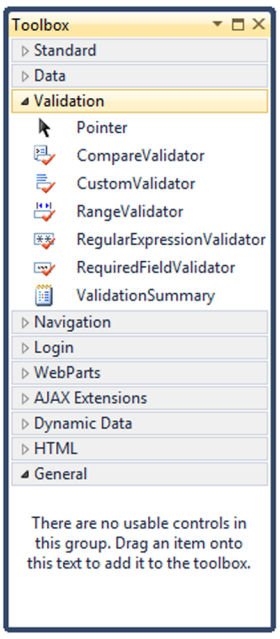


Figura 73. Controls de validació

Per a provar com funciona un control de validació, primer hem d'inserir el control en la pàgina, de manera que arrossegarem un TextBox des del Toolbox fins a l'editor.

Figura 74. Codi .aspx d'un control de validació

```
Client Objects & Events                                    (No Event
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
         Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>

100 %
```

A continuació arrosseguem el control de validació que vulguem aplicar, com per exemple, un RangeValidator, i en configurem els atributs següents:

- `ErrorMessage`: error que apareixerà quan la validació falli.
- `MaximumValue`: valor màxim que es pot introduir.
- `MinimumValue`: valor mínim que es pot introduir.
- `ControlToValidate`: ID del control sobre el qual s'aplicarà la validació.

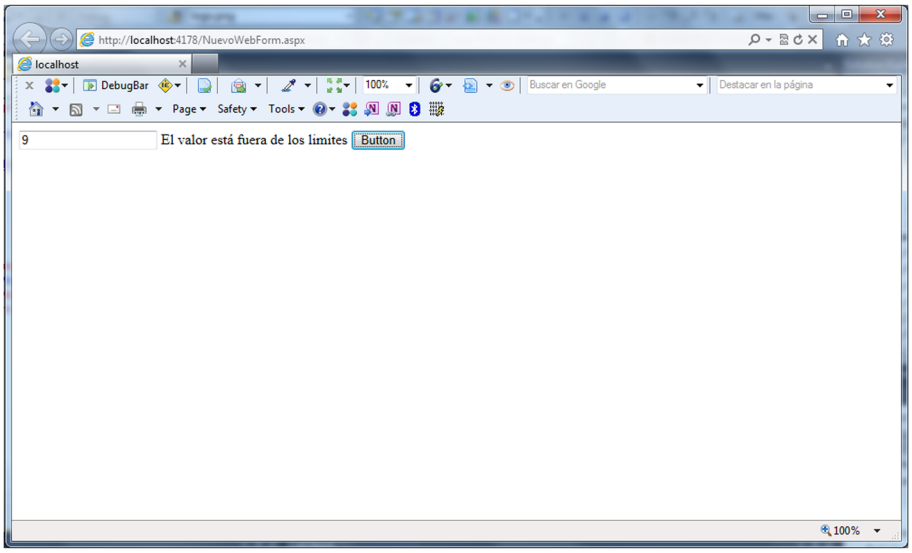
Finalment, arrosseguem un Button a la nostra pàgina, amb el qual farem l'enviament del formulari.

Figura 75. Codi .aspx d'un control de validació

```
Client Objects & Events                                    (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
        Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:RangeValidator ID="RangeValidator1" runat="server" ErrorMessage="El valor está fuera de los límites"
            MaximumValue="50" MinimumValue="10" ControlToValidate="TextBox1"></asp:RangeValidator>
        <asp:Button ID="Button1" runat="server" Text="Button" />
    </div>
    </form>
</body>
</html>

100 %
```

Si executem la nostra pàgina i introduïm valors en el TextBox veurem com ens apareix el missatge "El valor está fuera de los límites" quan enviem el formulari amb un valor erroni.

Figura 76. Execució de l'exemple d'un control de validació



4.7.5. Controls de dades

Els controls de dades ens permeten mostrar grans quantitats de dades amb suport de característiques d'alt nivell com ordenació, paginació, etc. El control per excel·lència i més utilitzat d'aquest grup és el GridView.

Aquest control és extremadament flexible i ens mostra les dades en forma de files i columnes. Amb aquest control es cobreixen les funcionalitats més comunes en el maneig de dades, com són la paginació, l'ordenació o la selecció d'elements.

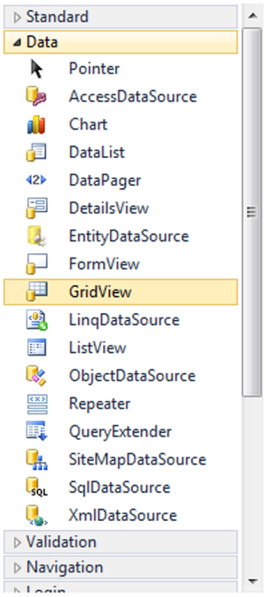


Figura 77. Controls de dades

Exemple pràctic d'ús del GridView

Per a saber millor com funciona el GridView, en veurem un petit exemple pràctic.

Igual que els altres controls, trobarem el GridView disponible en el Toolbox (figura 77). Arrosseguem un GridView a l'editor i veiem com automàticament genera el codi `xml`.

Figura 78. Codi .aspx d'un control de dades

```
Client Objects & Events                                    (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
         Inherits="NuevaAppWeb.NuevoWebForm" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
    </div>
    </form>
</body>
</html>
100 %
```

A continuació hem de proporcionar l'origen de les dades al control. La manera més senzilla és construir una llista d'objectes, de manera que creem una classe anomenada `Persona`.

Figura 79. Exemple d'un GridView

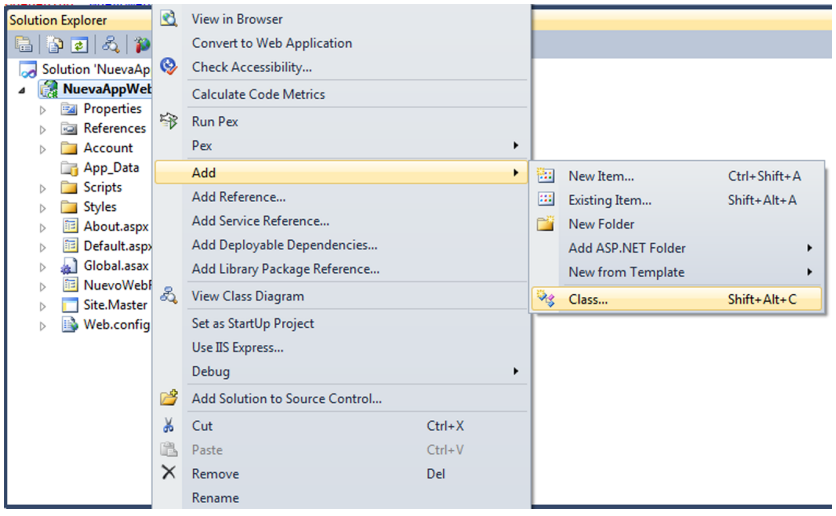


Figura 80. Exemple d'un GridView

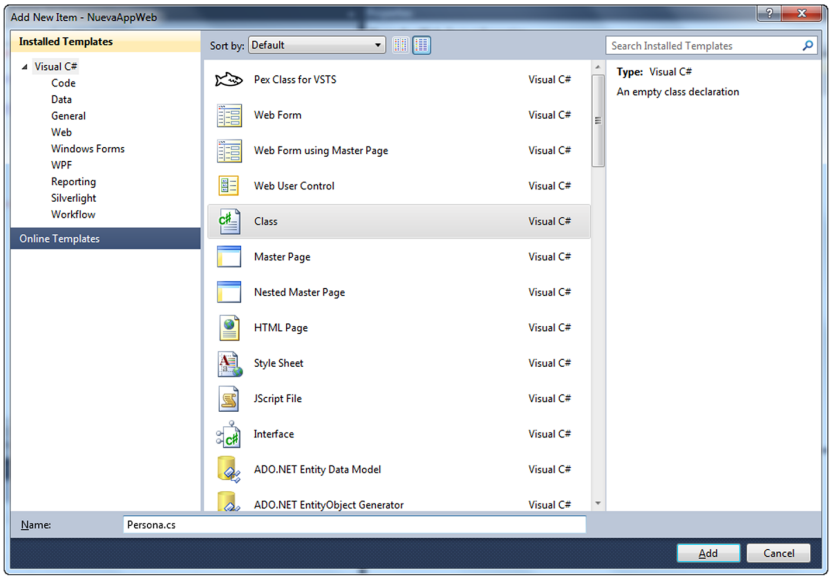
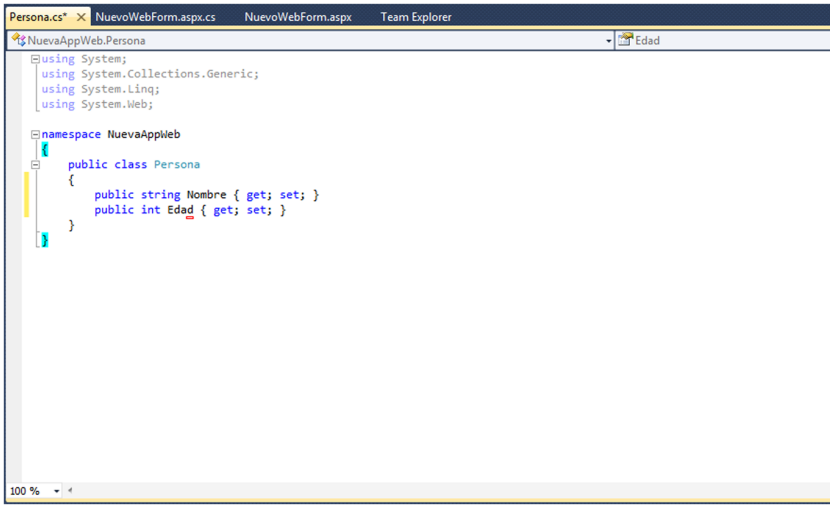


Figura 81. Exemple d'un GridView

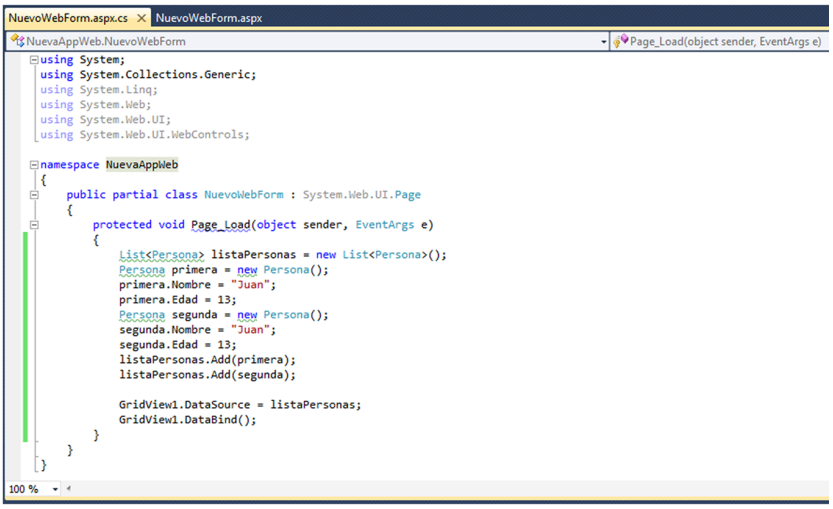


A continuació, en l'esdeveniment `Page_Load` de l'arxiu `.cs` generem una llista d'objectes `Persona`.

Figura 82. Exemple d'un GridView

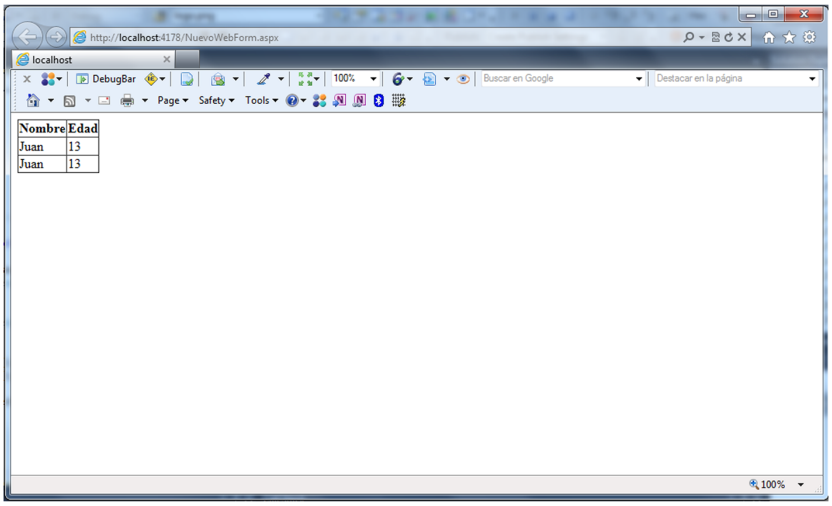
Finalment, relacionem la llista anterior creada amb l'origen de dades o `DataSource` del GridView.

Figura 83. Exemple d'un GridView



Si executem el nostre lloc web podem veure com automàticament es crea una Grid en què les columnes són les propietats de la classe `Persona` i les files corresponen a cadascun dels objectes de la llista.

Figura 84. Exemple d'un GridView



4.8. Controls d'usuari

Una aplicació web ben construïda divideix el codi en diferents blocs independents. Com més modular és l'aplicació, més fàcil és mantenir-ne el codi, solucionar errors i reutilitzar components. Per a l'organització del codi en diferents components ASP.NET posa a disposició del desenvolupador els UserControl o controls d'usuari. Els UserControl s'assemblen molt als WebForm, ja que estan compostos per una porció de codi amb etiquetes HTML i controls de servidor (l'arxiu `.ascx`) i un arxiu de codi amb la lògica i els esdeveniments.

Les úniques diferències entre els UserControl i els WebForm són:

a) Els UserControl utilitzen l'extensió `.ascx` en lloc d'`.aspx` i la classe del codi `.cs` hereta de `UserControl` en lloc de `Page`.

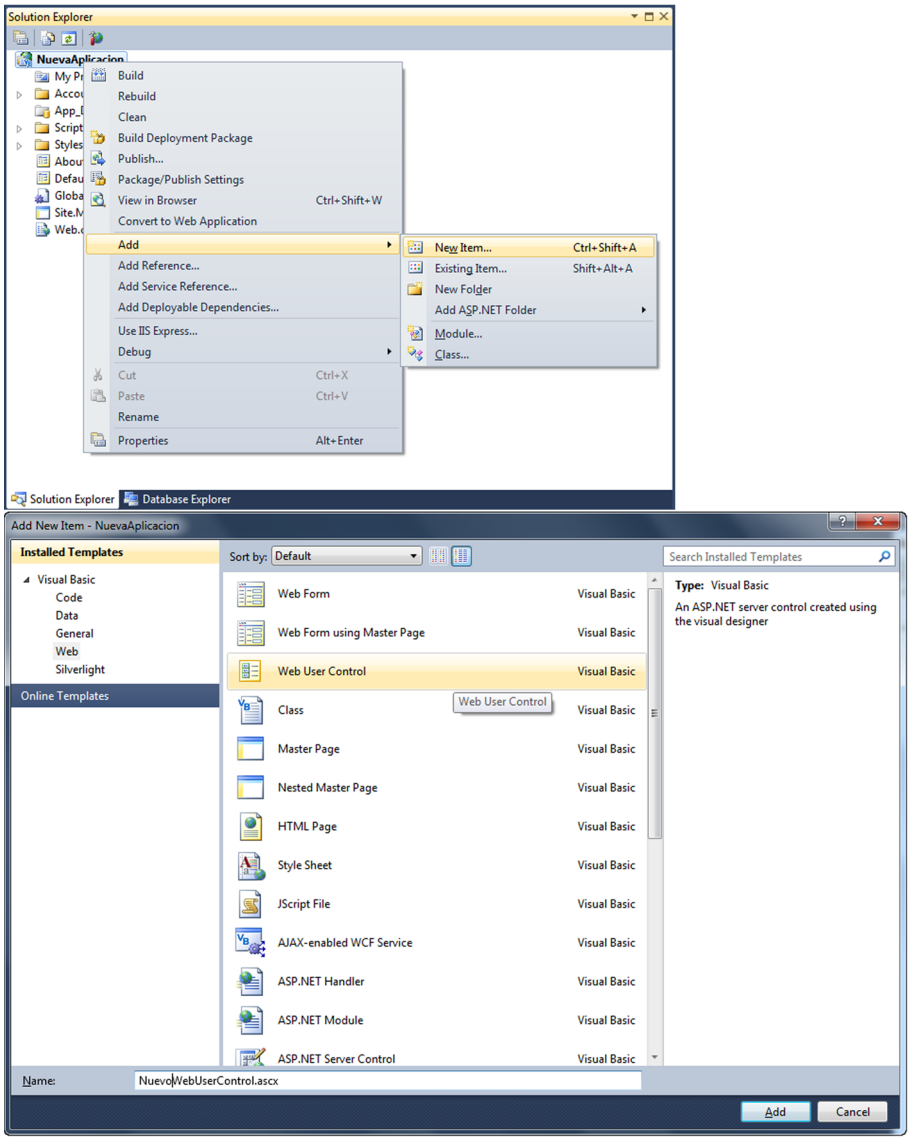
b) L'arxiu `.ascx` comença amb la directiva `<%@Control%>` en lloc de `<%@Page%>`.

c) Els UserControl no poden ser sol·licitats directament per un navegador. Sempre han d'estar embeguts dins d'un WebForm.

4.8.1. Creació d'un UserControl

La manera de crear un UserControl en el Visual Studio és molt similar a la manera de crear un WebForm. Hem de seleccionar *Add* > *Newitem* i escollir *Web User Control* de la llista. Li donem el nom, per exemple, de `NouControl.ascx`.

Figura 85. Creem un UserControl

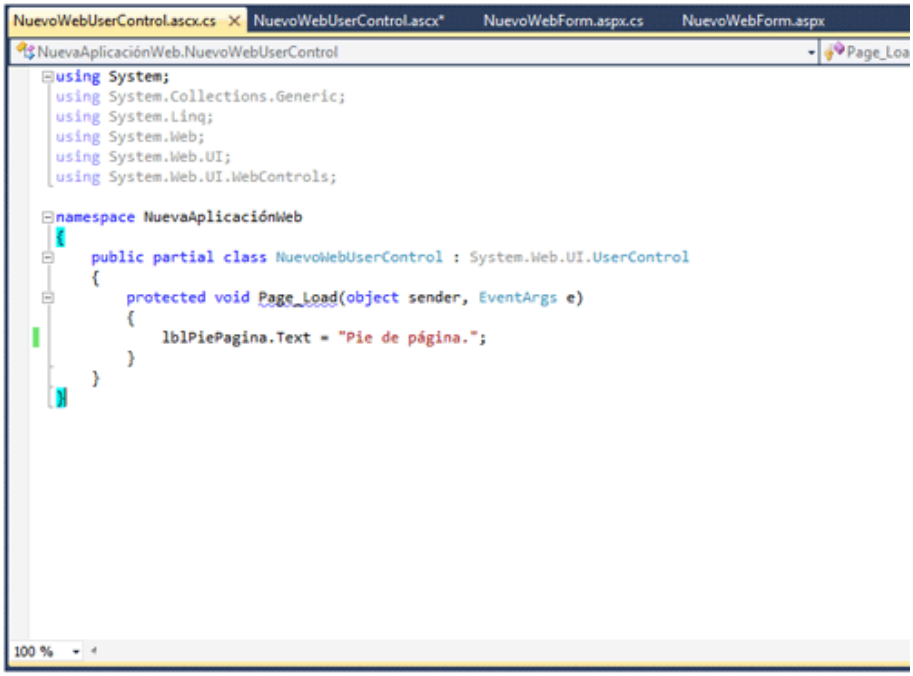


Modifiquem l'arxiu `.ascx` perquè quedi de la manera següent:

Figura 86. Codi .aspx d'un UserControl

En l'arxiu de codi relacionat veiem com els esdeveniments són molt semblats als dels WebForm. Modifiquem també aquest arxiu perquè quedi de la següent forma:

Figura 87. Codi .cs d'un UserControl



Per a poder provar aquest UserControl és necessari inserir-lo en un WebForm. Es tracta d'un procés de dos passos. En primer lloc necessitem agregar la directiva `Register` a la pàgina que contindrà el UserControl just després de la directiva `Page`.

```
<%@ Page Language = "C#" AutoEventWireup = "true" CodeBehind = "NouWebForm.aspx.cs"
                    Inherits = "NovaAplicacioWeb.NouWebForm" %>
<%@ Register TagPrefix = "customControls" TagName = "pie" Src = "NouWebUserControl.ascx" %>
```

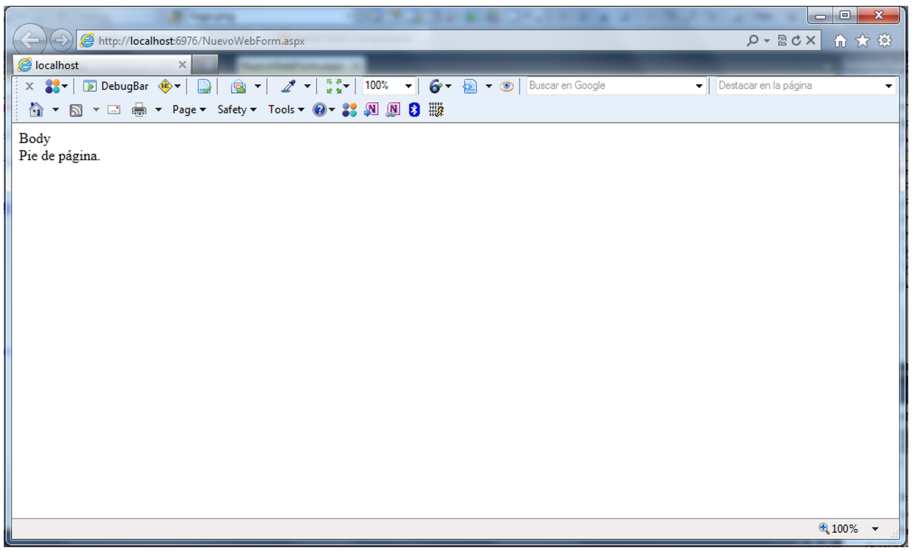
En aquesta directiva, com podem veure, s'ha d'especificar un prefix i un nom. La directiva `Src` serveix per a informar de la localització del UserControl.

A continuació hem d'inserir el UserControl en el WebForm com si fos un control de servidor, però mitjançant la directiva següent, especificant-ne el prefix i el nom:

```
<div>
   <customControls:peu runat = "server"></customControls:peu>
</div >
```

Si executem la pàgina podrem veure com el UserControl es mostra com si fos un control més (figura 88).

Figura 88. Execució de l'exemple del UserControl



4.9. Pàgines mestres

Quan dissenyem un lloc web, normalment disposem d'una sèrie d'elements comuns en totes les pàgines: el logotip de l'organització, un peu amb informació, una sèrie d'enllaços generals, etc.

Una novetat que va introduir ASP.NET són les pàgines mestres. Gràcies a aquestes es redueix el temps de disseny i el manteniment del lloc.

Una **pàgina mestra** és una combinació de codi HTML i controls de servidor, com qualsevol altra pàgina d'ASP.NET, amb la peculiaritat que s'emmagatzema en un arxiu amb extensió `.master` en lloc de l'extensió habitual `.aspx`.

Una altra diferència respecte als WebForm se centra en la capçalera de l'arxiu `.master`.

En una pàgina mestra, la directiva `Page` és substituïda per la directiva `Master`.

```
<%@ Master Language = "C#" AutoEventRireup = "true" CodeBehind = "Site1.master.cs"
                        Inherits = "NovaAppWeb.Site1" %>
```

Com hem dit, en una pàgina mestra podem introduir contingut HTML, components de servidor i qualsevol altre contingut que hi pugui haver en una pàgina ASP.NET. El que distingirà la pàgina mestra d'altres és l'aparició com a part del contingut d'un o més, components `ContentPlaceHolder`.

Figura 89. Codi .aspx d'una pàgina mestra

4.9.1. Components `ContentPlaceHolder` i `Content`

En dissenyar una pàgina mestra anirem deixant uns buits o contenidors, representats per a components `ContentPlaceHolder`. Després, en cada pàgina individual s'introduirà el contingut específic aportat pels components `Content`.

El component `ContentPlaceHolder`, per tant, apareixerà en l'arxiu `.master`. La propietat que més ens interessa d'aquest component és l'`ID`, a la qual assignarem un identificador o nom, que serà el que serveixi per a enllaçar-lo amb el control `Content`.

Figura 90. Codi .aspx d'una pàgina mestra

```
Client Objects & Events                                    (No Events)
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs"
           Inherits="NuevaAppWeb.Site1" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>

100 %
```

Figura 91. Codi .aspx d'una pàgina de contingut

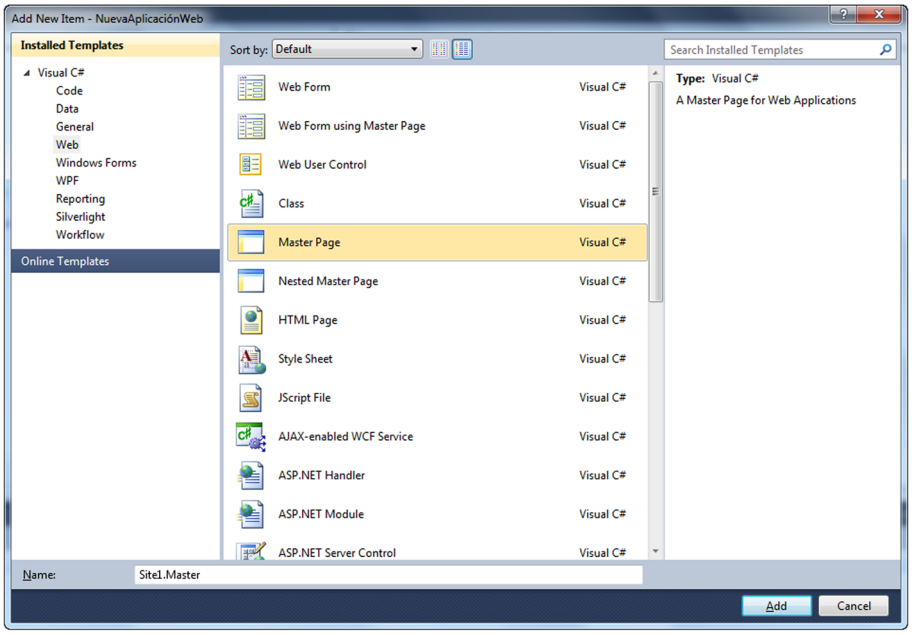
```
Client Objects & Events                                    (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
         Inherits="NuevaAppWeb.NuevoWebForm" MasterPageFile="~/Site1.Master"%>

<asp:Content ID="ContentHead" ContentPlaceHolderID="head" runat="server">

</asp:Content>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

</asp:Content>

100 %
```

Veiem que per a referenciar la pàgina mestra en el WebForm específic s'utilitza la propietat `MasterPageFile`, que apareix com a atribut en la directiva `Page` d'aquelles pàgines en què interessi incloure el disseny base.

4.9.2. Exemple pràctic

Ara que coneixem a grans trets els elements principals de les pàgines mestres, podem veure com les podem utilitzar en el disseny d'un lloc senzill. Començarem creant un nou projecto web buit. A continuació obrim el quadre de dialogo *Add* > *New item* amb el botó dret del ratolí sobre el nostre projecte i seleccionem *Master Page* (figura 92).

Figura 92. Afegim una plantilla de MasterPage



Aquesta pàgina mestra per defecte contindrà únicament un contenidor `ContentPlaceHolder`.

Figura 93. Codi .aspx d'una pàgina mestra

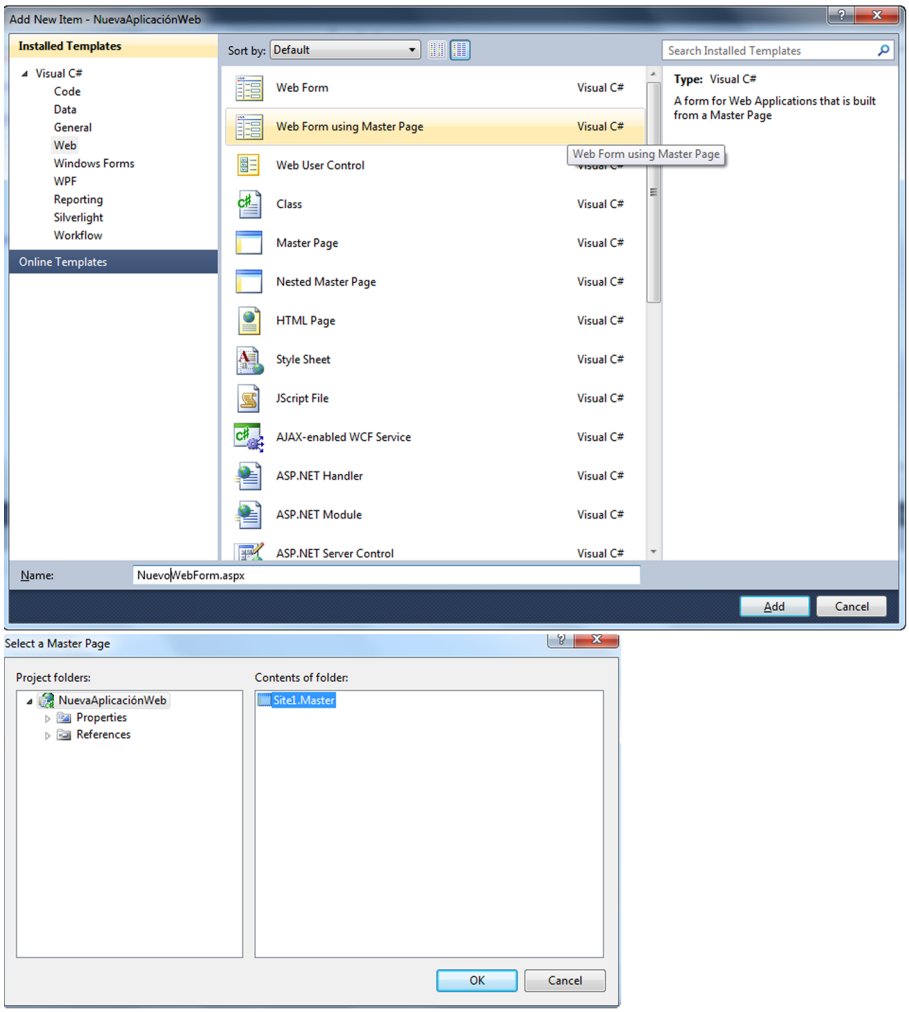
```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs"
           Inherits="NuevaAplicaciónWeb.Site1" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

Introduïm un contingut per defecte en el `ContentPlaceHolder` que es mostrarà sempre que no el sobreescriguem amb el component `Content` que veurem a continuació, i agreguem els elements propis de la pàgina mestra, com la capçalera o el peu.

Figura 94. Codi .aspx d'una pàgina mestra

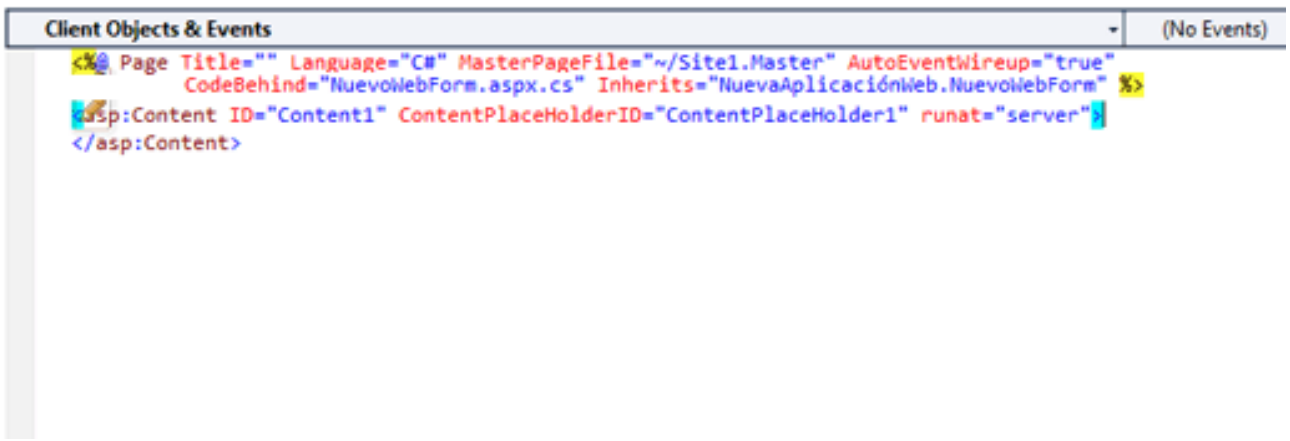
A continuació obrim una altra vegada el menú *Add* > *New* item i seleccionem *Web Forms using Master Page*. S'obrirà una finestra que mostrarà en la llista dreta les pàgines mestres disponibles. Triem la que acabem de crear.

Figura 95. Afegim la plantilla de pàgina de contingut



Per a cada pàgina que afegim al projecte i que enllacem a la pàgina mestra, haurem d'agregar un contingut en l'objecte `Content` corresponent. Per a relacionar el component `Content` amb el seu `ContentPlaceHolder` corresponent únicament hem d'indicar en el primer l'`ID` del segon.

Figura 96. Codi .aspx d'una pàgina de contingut



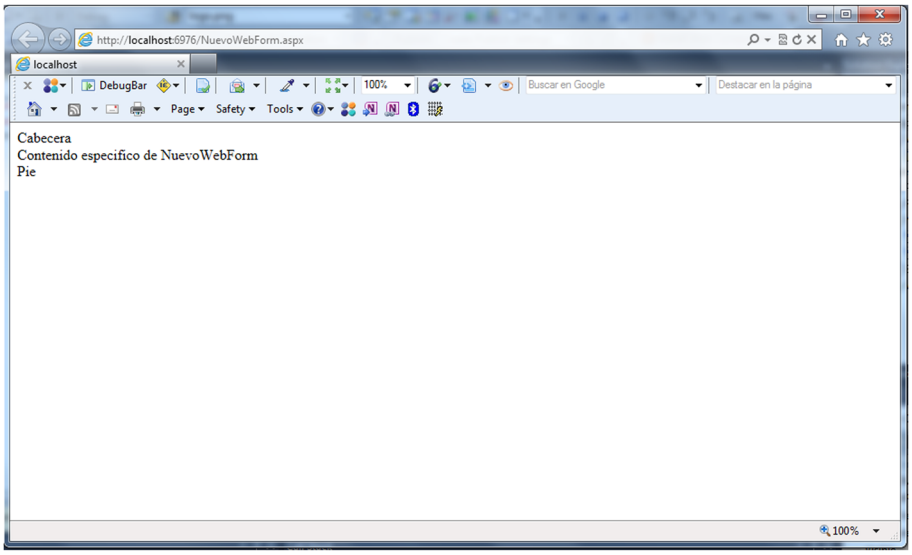
Introduïm el contingut que volem que aparegui en l'espai del `ContentPlaceHolder`.

Figura 97. Codi .aspx d'una pàgina de contingut

```
Client Objects & Events                                    (No Events)
<%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true"
         CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    Contenido especifico de NuevoWebForm
</asp:Content>
```

Amb aquests passos simples ja tindrem una pàgina accessible des del navegador amb una capçalera, un contingut i un peu. Si executem l'aplicació, veurem la pàgina que hem creat inclosa dins de la pàgina mestra (figura 98).

Figura 98. Execució de l'exemple de MasterPage



5. ASP.NET Ajax

En altres mòduls hem vist com podem implementar la tècnica Ajax mitjançant JavaScript, i més concretament mitjançant l'objecte `XMLHttpRequest` que hi ha en tots els navegadors actuals. Aquesta manera de construir peticions asíncrones, com hem vist, és bastant laboriosa i implica invertir molt temps de treball.

ASP.NET posa a la disposició del desenvolupador ASP.NET Ajax per a facilitar la tasca de la implementació Ajax en els nostres llocs web. ASP.NET Ajax consta de dues parts principals: una part del costat del client i una part del costat del servidor.

La part del costat del client és un conjunt de biblioteques JavaScript. Bàsicament, aquestes biblioteques afegeixen funcionalitats a JavaScript per a facilitar el desenvolupament de les nostres aplicacions.

La part de servidor d'ASP.NET Ajax, que és la que de debò ens interessa, funciona en un nivell superior. Inclou controls i components que utilitzen les biblioteques JavaScript del costat del client sense que n'hàgim de tenir coneixement.

5.1. L'ScriptManager

L'ScriptManager és el cervell del model de servidor ASP.NET Ajax. Es tracta d'un control web que no té cap aspecte visual; no obstant això, fa la tasca clau del connectar els controls de servidor Ajax amb les biblioteques JavaScript.

Per a agregar l'ScriptManager a una pàgina, el podem trobar en el Toolbox en l'apartat *Ajax Extensions* (figura 99). Una vegada arrossegat i deixat anar en l'editor obtindrem el codi següent (figura 100):

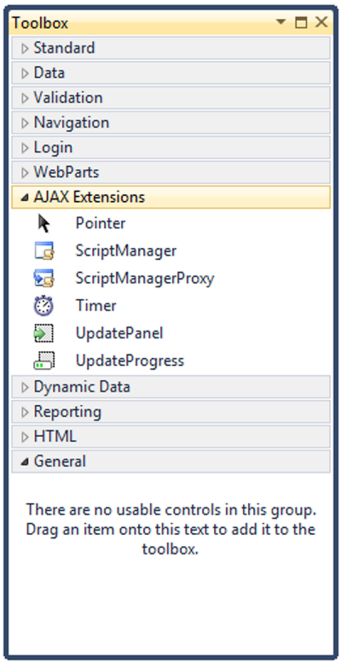


Figura 99. Caixa d'eines

Figura 100. Codi .aspx d'un ScriptManager

```
Client Objects & Events                                    (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
         Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div>

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

100 %
```

Cada pàgina que utilitza controls ASP.NET Ajax requereix una instància i solament una de ScriptManager.

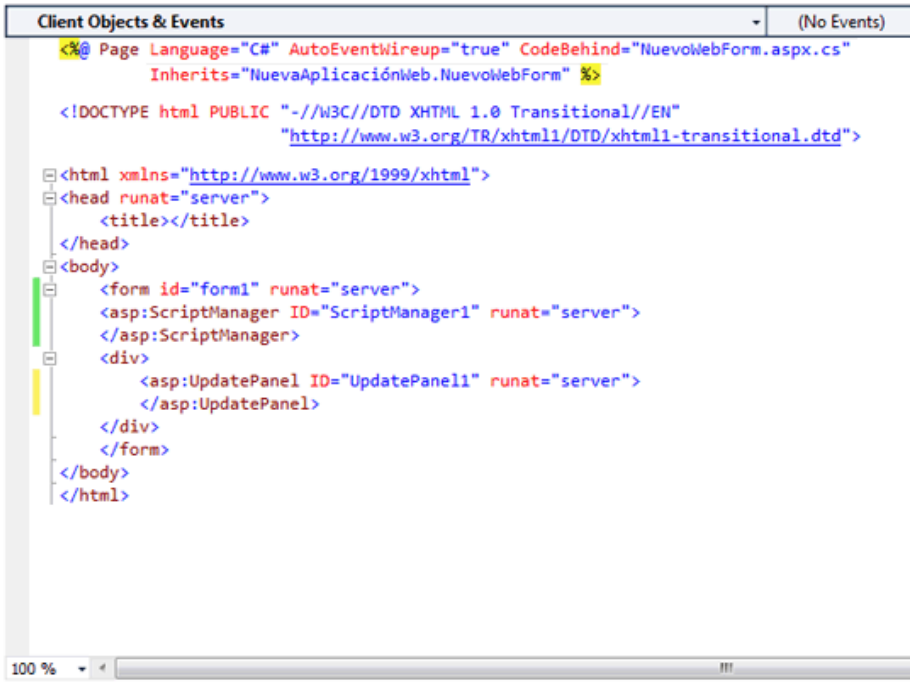
5.2. Renderitzacions parcials amb UpdatePanel

Sens dubte, el control Ajax més important de tots és UpdatePanel. Aquest control ens permet fer renderitzacions parcials de la nostra pàgina, és a dir, podem recarregar parts de la nostra pàgina sense haver de recarregar la pàgina sencera. L'escenari habitual d'un UpdatePanel és el següent:

- L'usuari llança un esdeveniment com el clic d'un botó dins d'un UpdatePanel.
- ASP.NET Ajax intercepta l'esdeveniment i fa la petició asíncrona de la pàgina al servidor.
- En el servidor es processa la pàgina de manera normal i es retorna al navegador.
- El navegador rep la pàgina i el codi JavaScript s'encarrega de recarregar solament l'UpdatePanel on es trobava el botó.

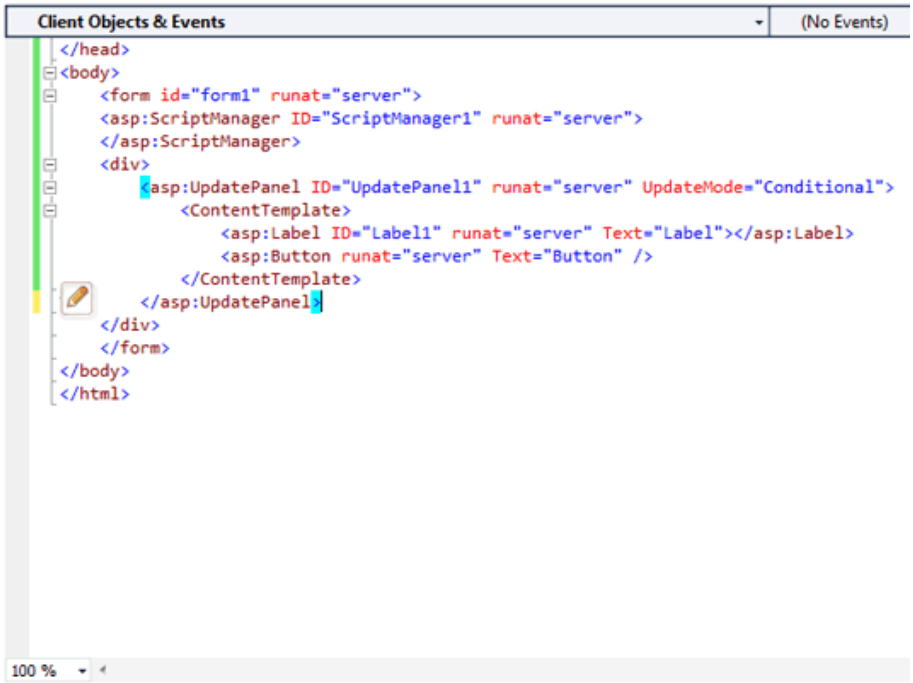
L'UpdatePanel treballa en conjunt amb el control ScriptManager i, per tant, quan utilitzem UpdatePanel hi ha d'haver abans un ScriptManager. Per tant, per a fer un exemple primer, inserim un ScriptManager a la nostra pàgina. A continuació, en el mateix apartat *Ajax Extensions* del Toolbox trobarem l'UpdatePanel. En inserir-lo a la nostra pàgina, es generarà el codi següent (figura 101):

Figura 101. Codi .aspx d'un UpdatePanel



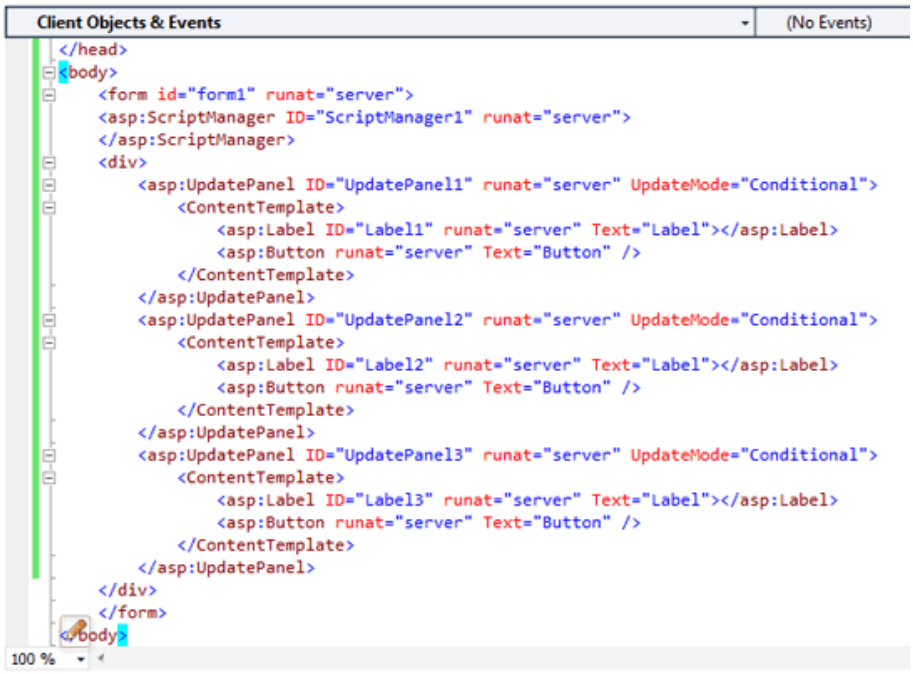
A continuació, inserim una Label dins de l'UpdatePanel per a anar mostrant el resultat de la crida asíncrona i un Button per a anar llançant aquesta crida.

Figura 102. Codi .aspx d'un UpdatePanel



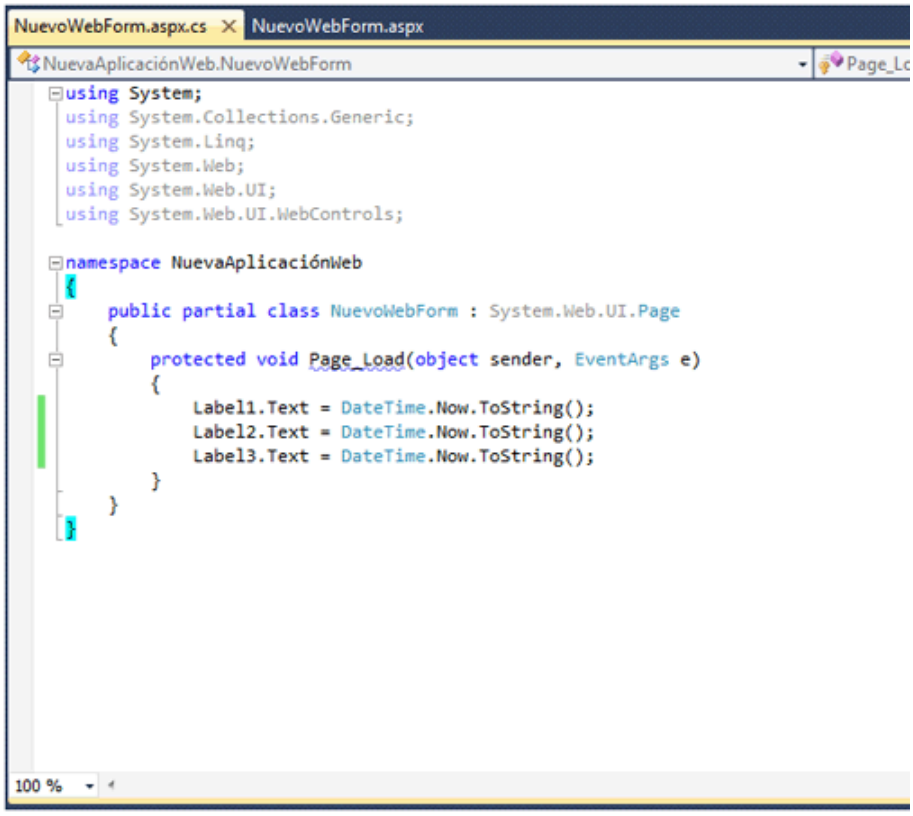
Fem el mateix procés per a agregar dos UpdatePanel més amb un Label i un Button cadascun.

Figura 103. Codi .aspx d'un UpdatePanel



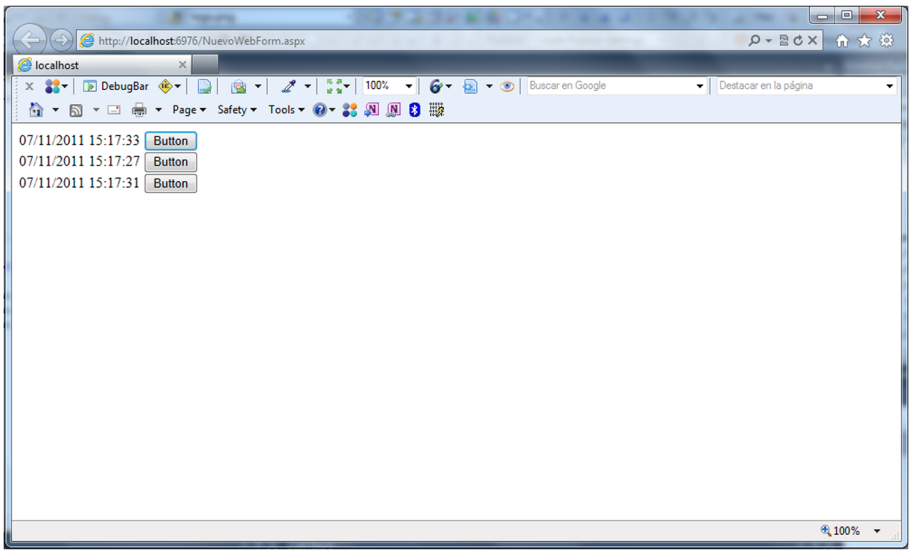
En l'arxiu `.cs` de la pàgina escrivim el codi següent per a actualitzar cadascun de les Labels:

Figura 104. Codi .cs d'un UpdatePanel



Si executem la pàgina veurem com per a cada clic de qualsevol Button solament es refresca l'UpdatePanel on està contingut, encara que hàgim variat els tres Labels.

Figura 105. Execució de l'exemple d'UpdatePanel



5.3. Indicació d'estat amb UpdateProgress

Mentre un navegador renova una pàgina mostra a l'usuari algun tipus d'indicació, com per exemple una icona animada o un canvi progressiu de color en la barra d'adreces, que permet saber que es troba ocupat, esperant rebre resposta del servidor. Per defecte, les aplicacions Ajax no tenen aquest tipus de comunicació de retorn, que resulta important, ja que sense aquesta la persona que utilitza el navegador no sap si la comunicació està en curs o és que l'aplicació senzillament ha deixat de funcionar.

Introduir una indicació d'estat en una aplicació ASP.NET Ajax resulta molt senzill, ja que hi ha un component anomenat *UpdateProgress* que s'encarrega de col·locar en la pàgina l'element de notificació que vulguem. Aquesta notificació, a més, es pot mantenir oculta fins al moment en què el temps d'espera superi un cert límit, i torna a desaparèixer automàticament quan la resposta ha arribat. Les dues propietats clau d'aquest component són `AssociatedUpdatePanelID` i `DisplayAfter`. Amb la primera, l'UpdateProgress s'associa amb un UpdatePanel de la pàgina, mentre que la segona estableix el nombre de mil·lisegons d'espera després dels quals es farà visible la indicació.

Igual que altres components, UpdateProgress és un contenidor inicialment buit. Perquè faci el seu treball, és necessari introduir-hi algun contingut: una etiqueta de text, un petit gràfic, etc.

Partint de l'exemple del subapartat anterior afegirem a l'interior del primer UpdatePanel un UpdateProgress i, dins d'aquest, una simple etiqueta de text amb un missatge.

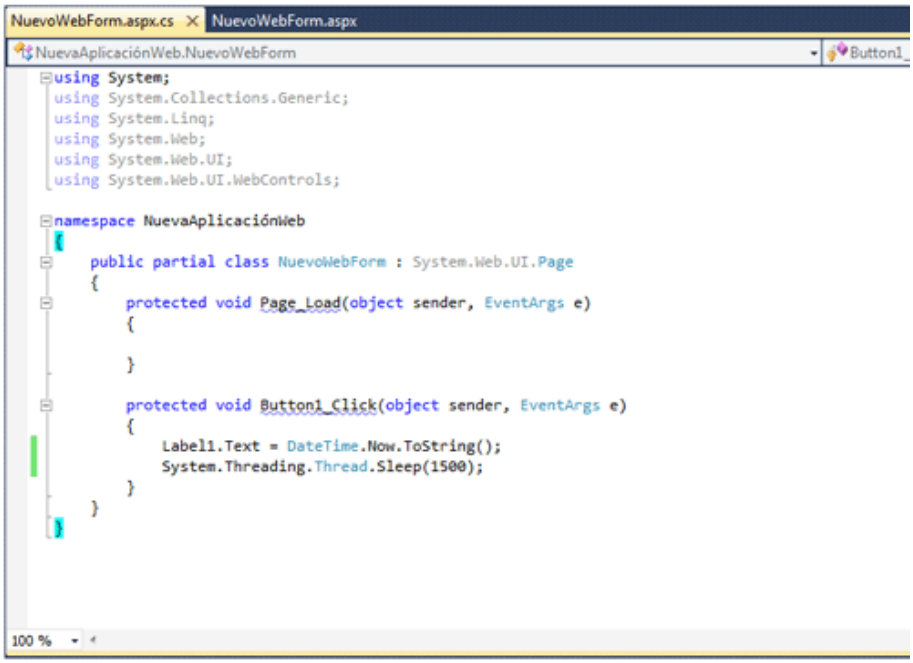
Figura 106. Codi .aspx d'un UpdateProgress

```
Client Objects & Events                                        (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
        Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server" />
                <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click"/>
                <asp:UpdateProgress ID="UpdateProgress1" runat="server">
                    <ProgressTemplate>
                        <asp:Label ID="Label2" runat="server" Text="Cargando..."/>
                    </ProgressTemplate>
                </asp:UpdateProgress>
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
100 %
```

Si executem el projecte, no descobrirem en principi cap diferència, ja que l'UpdateProgress està programat per a mostrar el missatge si la resposta triga més de mig segon a arribar. En un entorn real es podria donar el cas que la petició trigués més de mig segon, però provant en local, la resposta és gairebé immediata.

Per a simular aquest temps d'espera escriurem el codi següent, que retarda el fil d'execució:

Figura 107. Codi .cs d'un UpdateProgress



Ara, en executar de nou l'aplicació, es podrà apreciar perfectament el funcionament del component UpdateProgress.

5.4. ASP.NET Ajax Control Toolkit

Els controls UpdatePanel i UpdateProgress són bastant útils. No obstant això, són els únics controls Ajax que es troben en ASP.NET. Per a ampliar la gamma de controls Ajax Microsoft i la comunitat ASP.NET van desenvolupar ASP.NET Ajax Control Toolkit.

ASP.NET Ajax Control Toolkit està format per desenes de controls que utilitzen les biblioteques JavaScript d'ASP.NET Ajax per a crear efectes sofisticats. Dos factors que ASP.NET Ajax Control Toolkit té al seu favor són:

- És completament gratuït.
- Se n'inclou el codi font complet, i gràcies a això el programador podrà personalitzar els controls que el componen.

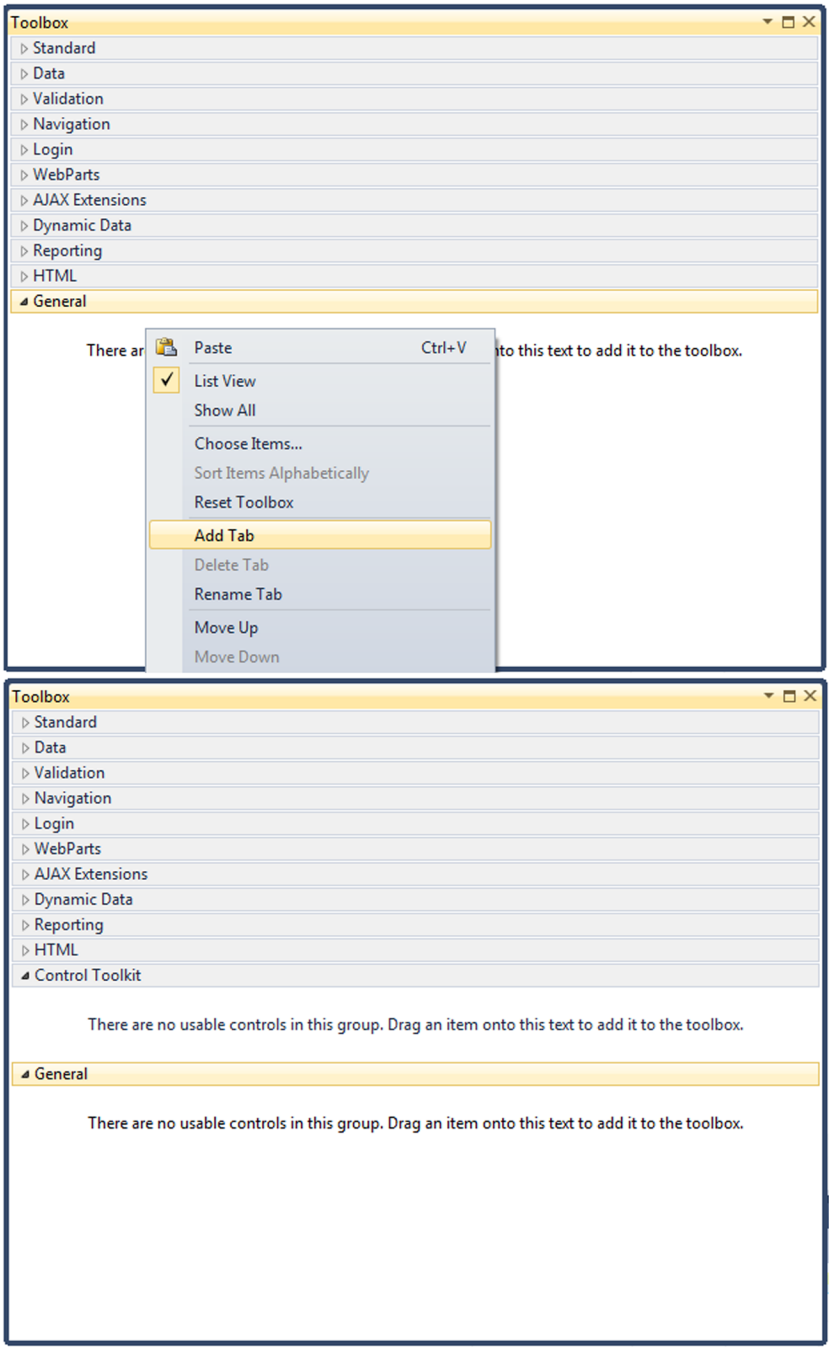
5.4.1. Instal·lació d'ASP.NET Ajax Control Toolkit

Per a baixar l'ASP.NET Ajax Control Toolkit, anirem a l'adreça Ajax Control Toolkit.

Una vegada hàgim baixat l'arxiu `.zip` simplement descomprimim l'arxiu `AjaxControlToolkit.dll`. Per a aconseguir integrar ASP.NET Ajax Control Toolkit amb el nostre lloc web, hem de seguir una sèrie de passos:

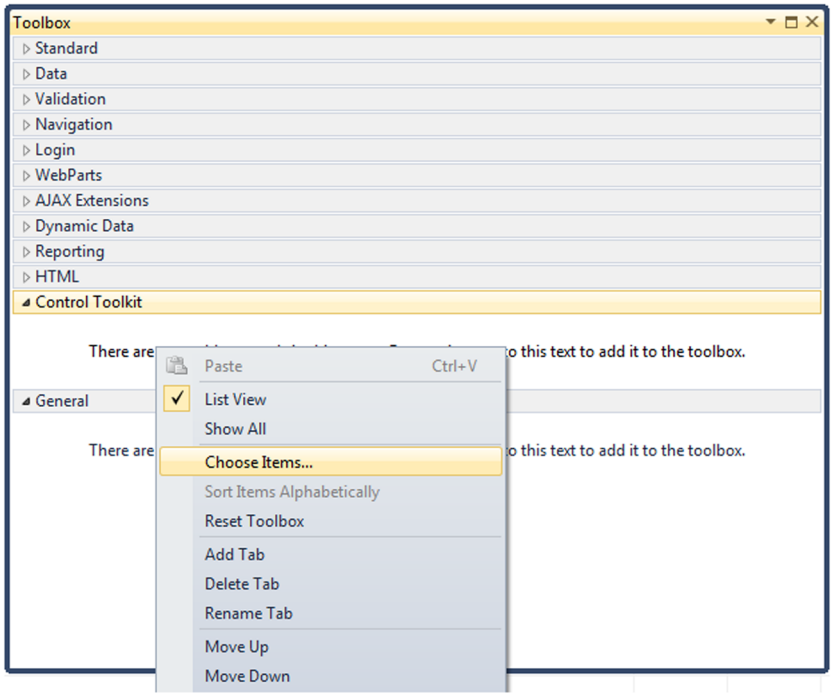
1) Primer de tot és necessari crear un apartat nou en el nostre Toolbox prement amb el botó dret sobre aquest i seleccionant *Add Tab* (figura 108). El nou apartat l'anomenem "Control Toolkit".

Figura 108. Afegim una secció per a Control Toolkit



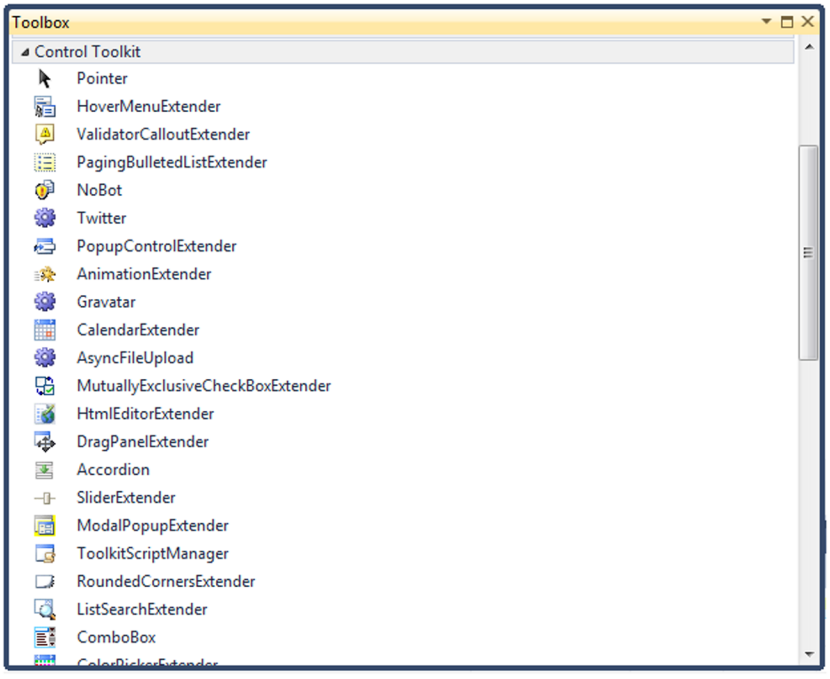
En aquest moment tenim un nou apartat en el Toolbox, però no conté cap control. Per a afegir els controls premem amb el botó dret sobre *Control Toolkit* i seleccionem *Choose items* (figura 109). A continuació, explorem en el nostre sistema i escollim l'arxiu `AjaxControlToolkit.dll` que vam extreure anteriorment.

Figura 109. Afegim una secció per a Control Toolkit



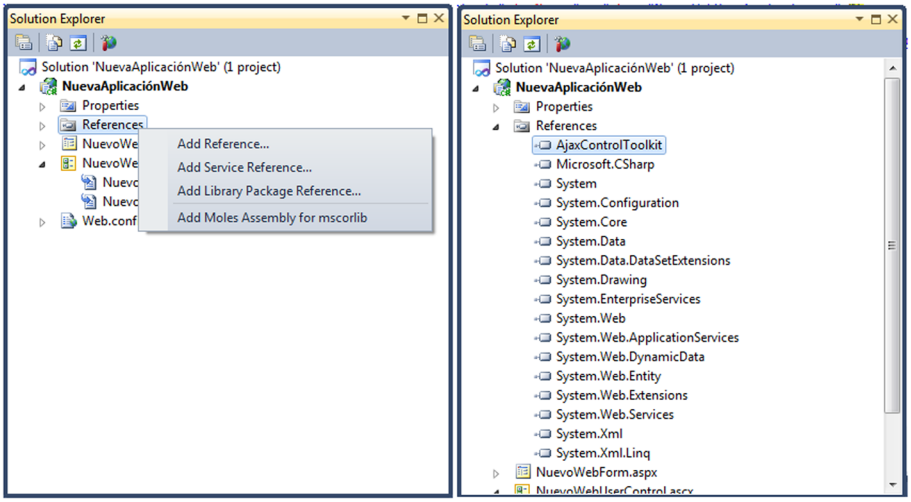
Automàticament, es crearan tots els controls en el nostre nou apartat Control Toolkit (figura 110).

Figura 110. Afegim una secció per a Control Toolkit



2) El segon pas important és afegir la biblioteca `AjaxControlToolkit.dll` al nostre lloc web. Premem amb el botó dret sobre la carpeta *References* i seleccionem *Add Reference*. A continuació, explorem en el nostre sistema i escollim l'arxiu `AjaxControlToolkit.dll` (figura 111).

Figura 111. Afegim una referència a Control Toolkit



Una vegada afegida la biblioteca dels controls l'hem de registrar en cada WebForm o UserControl que vulguem amb la directiva `Register`.

```
<%@ Register Assembly = "AjaxControlToolkit" Namespace = "AjaxControlToolkit" TagPrefix = "asp" %>
```

A partir d'ara podrem utilitzar qualsevol control d'ASP.NET Ajax Control Toolkit mitjançant l'etiqueta `<asp: NomControl></asp: NomControl>`.

La biblioteca ASP.NET Ajax Control Toolkit, com ja hem dit, disposa de multitud de controls. L'objectiu del mòdul, com ja hem anat comprovant, no és una anàlisi exhaustiva de tots els controls, sinó veure uns exemples bàsics al més genèrics possibles. En els dos subapartats següents veurem dos controls d'exemple per a veure com s'utilitzen: Accordion i AutoCompleteExtender.

5.4.2. Accordion

Un control Accordion és un contenidor en forma de pila amb diversos panells en què se'n pot veure només un cada vegada. Cada panell té una capçalera i un contingut. Quan es fa clic en la capçalera d'un, el panell s'expandeix i es mostra el contingut alhora que els altres panells es minimitzen.

Per a fer un petit exemple amb el control Accordion crearem un nou WebForm, al qual afegirem la directiva:

```
<%@ Register Assembly = "AjaxControlToolkit" Namespace = "AjaxControlToolkit"
                        TagPrefix = "asp" %>
```

Com qualsevol altre control Ajax, els controls de la biblioteca ASP.NET Ajax Control Toolkit necessiten la presència d'un control ScriptManager.

A continuació arrossegarem el control Accordion de l Toolbox, amb la qual cosa es crearà el codi següent (figura 112):

Figura 112. Codi .aspx d'un Accordion

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
                </asp:ScriptManager>
    <div>
        <asp:Accordion ID="Accordion1" runat="server">

        </asp:Accordion>
    </div>
    </form>
</body>
</html>
```

Seguidament afegirem el codi següent per a construir el nostre Accordion (figura 113):

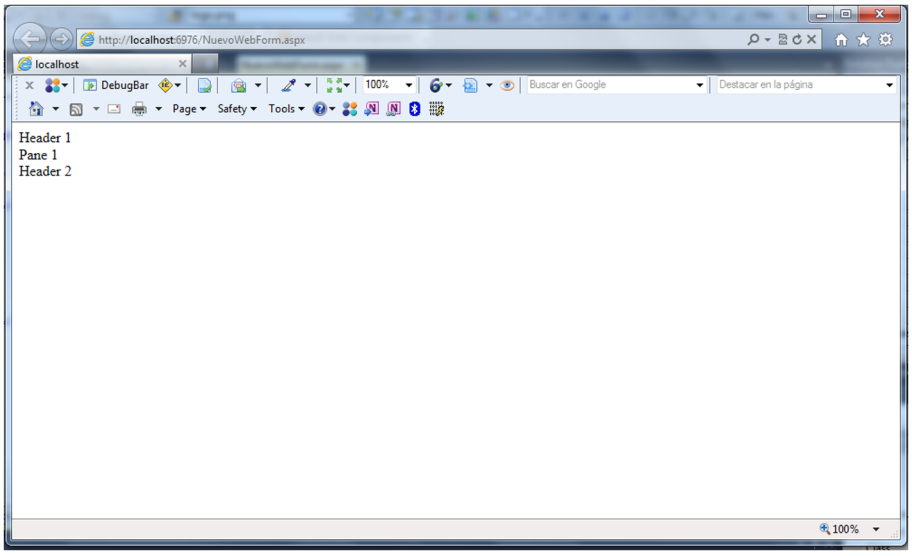
Figura 113. Codi .aspx d'un Accordion

```
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
                </asp:ScriptManager>
    <div>
        <asp:Accordion ID="Accordion1" runat="server">
            <Panes>
                <asp:AccordionPane ID="AccordionPane1" runat="server">
                    <Header>
                        Header 1
                    </Header>
                    <Content>
                        Pane 1
                    </Content>
                </asp:AccordionPane>
                <asp:AccordionPane ID="AccordionPane2" runat="server">
                    <Header>
                        Header 2
                    </Header>
                    <Content>
                        Pane 2
                    </Content>
                </asp:AccordionPane>
            </Panes>
        </asp:Accordion>
    </div>
    </form>
```

Veiem com cada secció del nostre Accordion es correspondrà amb un `AccordionPane` i cada `AccordionPane` conté un `Header` (que serà el que es mostri com a capçalera) i un `Content` (que serà el que es mostri quan el panell s'expandeixi).

Si executem la nostra web podrem comprovar el funcionament de l'Accordion (figura 114).

Figura 114. Execució de l'exemple de l'Accordion



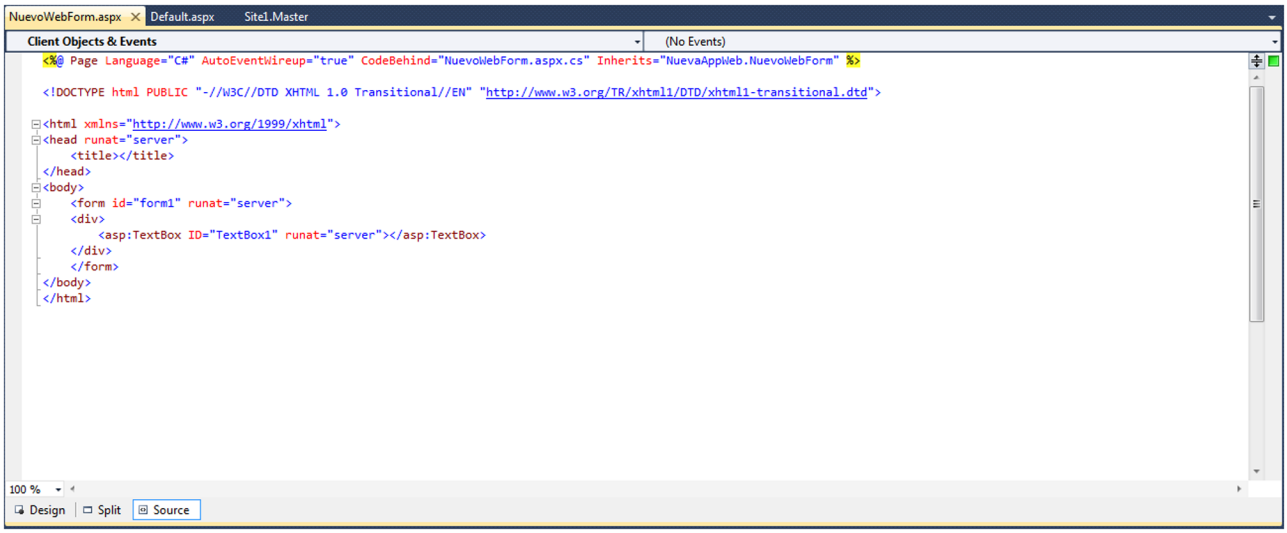
5.4.3. AutoCompleteExtender

L'exemple anterior de l'Accordion mostra un control totalment nou que té característiques Ajax. En la biblioteca ASP.NET Ajax, Control Toolkit no és el cas més comú, ja que el més normal són controls que es complementen amb els ja existents en ASP.NET per a afegir-los funcionalitats Ajax.

Una d'aquestes extensions de control és AutoCompleteExtender, que es complementa amb controls com TextBox, i ens va oferint una llista de valors possibles mentre l'usuari introdueix caràcters. Si l'usuari fa clic en un dels elements de la llista, el valor es copia dins del quadre de text.

Per a veure la funcionalitat de l'AutoCompleteExtender crearem un petit exemple. Primer de tot inserirem un TextBox en un nou WebForm.

Figura 115. Codi .aspx d'un AutoCompleteExtender



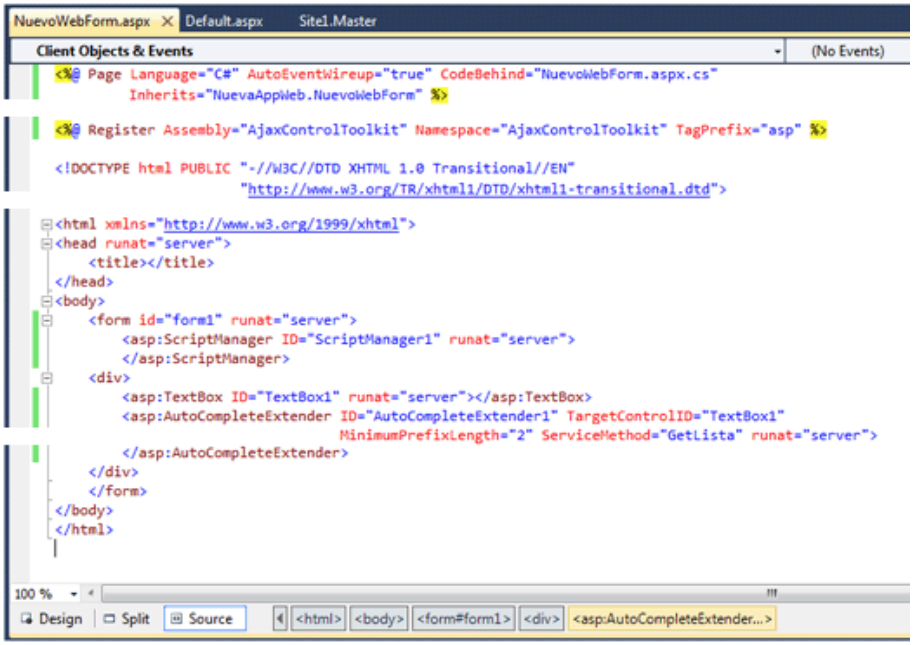
A continuació, i com en tots els controls Ajax, inserirem un ScriptManager que controlarà la part JavaScript del control. En aquest moment ja estem en condicions d'agregar el control AutoCompleteExtender a continuació del nostre *TextBox*. Únicament hem de configurar tres propietats del control:

- `TargetControlID`: identificador del control sobre el qual s'aplicarà l'extensió.

- `MinimumPrefixLength`: a partir que l'usuari escrigui *x* caràcters, es mostrarà la llista.

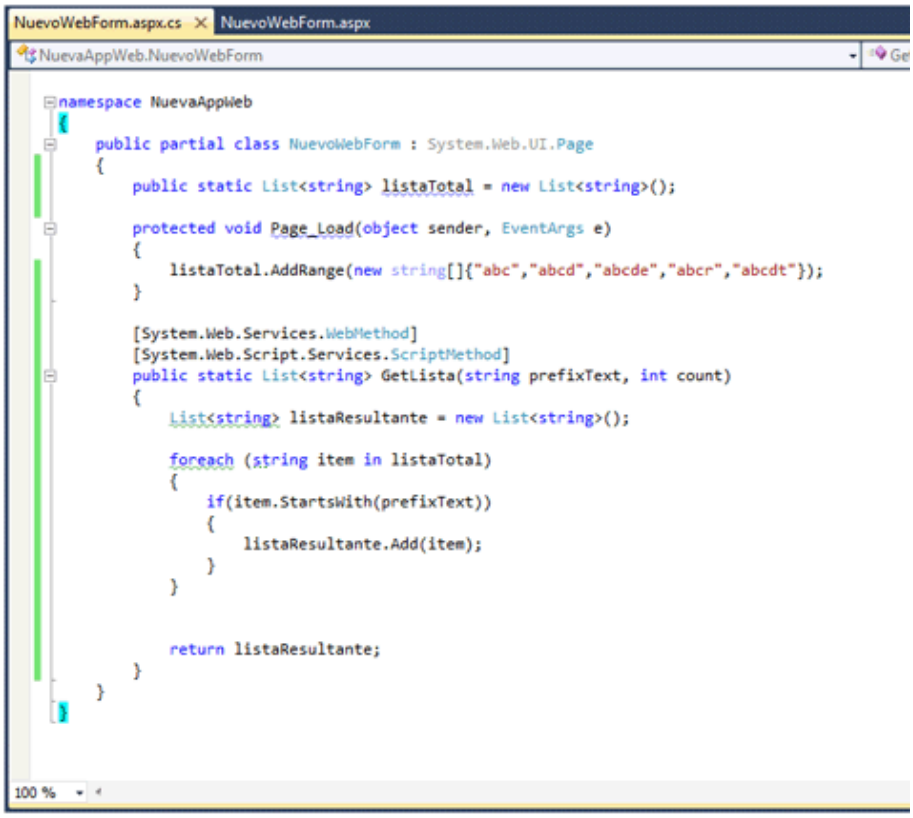
- `ServiceMethod`: mètode que s'encarregarà de retornar els ítems que es mostraran en la llista

Figura 116. Codi .aspx d'un AutoCompleteExtender



El mètode que retornarà la llista de *strings* que es mostrarà ha de tenir les anotacions de `WebMethod` i `ScriptMethod` perquè sigui accessible directament des de JavaScript.

Figura 117. Codi .cs d'un AutoCompleteExtender



Veiem com tenim una llista amb tots els *string* "llistaTotal" i únicament hem de comprovar en la funció `GetLlista` si cada element d'aquesta llista comença per la cadena entrada per l'usuari.

Si executem l'aplicació web veurem com a mesura que l'usuari insereix caràcters la llista es va actualitzant.

Figura 118. Execució de l'exemple de l'AutoCompleteExtender