Björn G. D. Persson

Introduktion till ASP.NET 3.5

med C#

Webbaserad applikationsutveckling

mars 2009

Om denna sammanfattning

Syftet med denna sammanfattning (introduktion) är att (förhoppningsvis) ge en första inblick i hur man skapar webbformulär (*web forms*) med ASP.NET 3.5, något som till stor del även gäller i ASP.NET 2.0 (version 3.5 utökar främst med fler klasser/bibliotek). Introduktionen är på inget sätt uttömmande i ämnet och den bygger på mina tidigare sammanfattningar i samma ämne.

Jag börjar med att behandla hur man skapar ASPX-sidor på egen hand (projekt med inbäddad kod, *inline code*) för att sen visa hur man tar hjälp av *Visual Studio 2008* (VS2K8, VS.NET eller bara VS) för att utveckla projekt med bakomliggande kod (*code behind*). Sedan fortsätter jag med att titta på några vanliga kontroller i ASP.NET, s.k. webbserverkontroller, för att gå vidare med att titta på hur man presenterar och uppdaterar data från datakällor. Efter detta tittar jag på lite mer avancerade koncept så som inloggning och privata sidor, m.m.. Sist av allt avslutas med ett kapitel med ”Frågor och svar”.

För att kunna göra (och köra) exempel i introduktionen krävs *Internet Information Server* (IIS) och *.NET Framework 3.5* (med ASP.NET 3.5[1]) på datorn som webbserver exekverar på samt (lämpligen) *Visual Studio 2008*, *Express* eller *Professional Edition*. (Server och klient kan vara samma dator. ☺ Se sammanfattningen *Installera IIS, .NET Framework 2.0, databasklienter och verktyg* för hur IIS, *.NET Framework*, m.m. installeras samt lösningar på en del problem som kan uppstå.) Rekommenderat operativsystem för dator med webbserver är *Windows 2000* (*Professional* eller *Server*), *Windows XP Professional* (inte *Home Edition*![2]), *Windows Vista* (ej *Home Basic*!) eller *Windows Server 2003* (även om det senare kan ”strula” med säkerhetsinställningar i de två sista…). Beskrivningar som gäller VS bygger på *Visual Studio 2008 Professional Edition* i *Windows Vista Ultimate* (då jag har dessa installerade), men bör fungera med andra versioner av *Windows* och *Visual Web Developer 2008 Express Edition*, där den senare kan laddas ner gratis från Microsofts hemsidor.

Kod har skrivits med typsnitt av fast bredd (`Courier New`) för att göras mer lättläst samt längre exempel har inneslutits i en ram (se exempel nedan). Kod som måste skrivas på en rad, men som inte får plats på en rad i detta dokument, har radbrutits och en ➥–symbol visar på denna ”ofrivilliga” radbrytning.
I C# avslutas programsatser (*statements*) med semikolon (;), d.v.s. skriv gärna kod på flera rader om de överstiger ca. 80 tecken då det underlättar läsning av kod då man slipper skrolla i sidled för att läsa längre programsatser.

```
enVariabel = 1
             + 2;    //Kommentarer i kod skriv med fet stil
```

Spara papper, skriv inte ut sammanfattningen! Ladda ner PDF-filen istället.

Till sist… jag är en nörd, d.v.s. jag tycker ”det här med datorer” är roligt/intressant. Så ta inte allt jag skriver på så mycket allvar – försök att se livet, och smått humoristiska saker jag skriver/skriver om, på den ljusa sidan. ☺

Jag är givetvis tacksam för alla konstruktiva synpunkter på sammanfattningens utformning och innehåll. (Någon av de över 200 personerna, som laddat ner min utgåva av version 2.0 av denna sammanfattning, kunde meddelat mig var att den var rent struntprat i den versionen av dokumentet, d.v.s. att det fanns hur många fel som helst i den…☹)

Stockholm, mars 2009

Björn Persson, e-post: (se startsida på min personliga hemsida)
Personlig hemsida: `http://www.kiltedviking.net/`

[1] IIS bör installeras före *.NET Framework*, annars installeras (eventuellt) inte ASP.NET! Se sammanfattningen *Installera IIS, .NET Framework 2.0, databasklienter och verktyg* för vad som behöver göras om *.NET Framework* installeras före IIS.
[2] Det finns ingen IIS till *Windows XP Home Edition*, men det går att använda Visual Studio .NETs webbserver.

Innehållsförteckning

1 Introduktion till ASP.NET

ASP.NET är en ”uppföljare” till *Active Server Pages* (ASP) – de påminner om varandra men ASP.NET är ett steg i ”rätt riktning”. En viktig skillnaden är bl.a. att ASP.NET är kompilerad kod (i motsats till ASP:s, och PHP:s, tolkade skriptkod) och en annan att utveckling med ASP.NET är mer strukturerad – logik kan separeras från presentation. Annat som skiljer ASP.NET från ASP är att filändelsen är .ASPX[3] (istället för .ASP). ASP.NET (liksom *.NET Framework*) finns idag i följande versioner: 1.0, 1.1, 2.0, 3.0 och 3.5. (Några versioner måste installeras manuellt medan andra installeras med *Windows*, *Visual Studio .NET* eller s.k. *service pack* [SP] för *Windows*. En del kan installeras med *Windows/Microsoft Update*.)

Jag börjar med att behandla hur man skapar ASPX-sidor för hand (s.k. inbäddad kod, *inline code*) för att sen visa hur man gör det mesta med *Visual Studio .NET* (projekt med *code behind*). I sammanfattningens exempel kommer jag skapa ett projekt kallat IntroASPNET35CS (som står för ”Introduktion till ASP.NET 3.5 med C#”[4] ☺).

1.1 Några skillnader mellan ASP.NET 1.1, 2.0 och 3.x…

ASP.NET 2.0, liksom 3.x, fungerar på många sätt som tidigare[5] versioner av ASP.NET, d.v.s. det grundläggande funktionerna är de samma. De stora skillnaderna finns i hur de s.k. webbkontrollerna (se kapitlet *Webbserverkontroller* nedan) fungerar, främst att vi slipper skriva ytterligare en massa kod. En av de största förändringarna har skett bland datakontrollerna, vilket inte bara underlättar kodning, utan även förståelsen av hur de används/fungerar. Skillnaden mellan 2.0 och 3.x är främst att det introducerats fler klasser/bibliotek i 3.x, d.v.s. kontrollerna har i stort sett samma funktion i bägge versioner.

1.1.1 … samt Visual Studio .NET 2003, 2005 och 2008

Även i VS 2003 och 2005 så skiljer sig en del saker… några av dem är:

- Numera möjlighet att ändra sökvägen till var lösningsfiler (.SLN-filen) ska placeras när ett webbprojekt skapas (mot tidigare då VS måste konfigureras för att tala om var de skulle placeras). Alla övriga filer placeras, som tidigare, på webbserver.
- När ett webbformulär (ASPX-fil) öppnas så öppnas det i källkodsläge (i motsats till tidigare när det öppnades i designläge). Jag rekommenderar att webbformulär skapas (d.v.s. kontroller och XHTML-kod läggs till) i kodläge för att sen konfigurera dem i designläge.
- Som standard används XHTML istället för ”vanlig” HTML i webbsidor.
- Intellisense[6] fungerar bättre. ☺
- ... och i 2008 så är redigering helt enkelt mycket simplare/trevligare. ☺

1.2 Grundläggande om ASP.NET

ASPX-sidor kan skapas på två sätt:

- med inbäddad kod (*inline code*)

[3] M$ hade, eller verkar fortfarande ha, en period där dom tyckte det var häftigt att haka på ett X på allt…
[4] Det är mycket tänkbart att jag gör om sammanfattning för även VB.NET, eller kanske lägger till sådana exempel i denna. Jag har alltså följt trenden och övergått till C# då jag ”tvingats” av mina arbetsgivare...
[5] Det låter som om M$ har hållit på att utveckla ASP.NET i många, många år…☺
[6] *Intellisense* är den funktion som visar en lista på variabler, klasser, metoder, attribut, m.m. när vi börjar skriva kod, d.v.s. en mycket trevlig funktion som blivit ännu bättre i VS 2005/2008.

- eller bakomliggande kod (*code-behind*).

I det första fallet placeras programkoden (för sidans logik) direkt i ASPX-sidans kod, d.v.s. inbäddad inuti (X)HTML-koden (vilket bör undvikas). Detta sätt påminner om hur ”vanliga” ASP-sidor (ASP v.3 eller tidigare) skapades, men mer strukturerat. Med bakomliggande kod så placeras programkoden i en separat fil (ASPX.CS-fil), en fil som inkluderas (längst upp) i ASPX-filen. I båda fallen kompileras dock koden till en DLL-fil, d.v.s. slutresultatet blir ”det samma” (men utvecklingen sker mindre strukturerad med inbäddad kod). I VS 2008 så har man gått ett steg längre och separerar genererad kod (av VS) från utvecklares kod (din kod).

Som standard så är `default.xxx` (ersätt `xxx` med `htm`, `asp` eller `aspx`) namnet på ”standardfiler” (eller s.k. indexfiler) i IIS.[7] En ”standardfil” är den fil som laddas om man inte anger ett filnamn i en webbadress (URL), d.v.s. det bör finnas en ”standardfil” i varje mapp på en webbserver. T.ex. så är nedanstående adresser samma i IIS:[8]

```
http://www.eki.mdh.se/studentguide/
http://www.eki.mdh.se/studentguide/default.htm
```

1.2.1 Skapa projekt

Innan vi kan lägga till några filer så måste vi skapa ett webbprojekt och om vi skapar ett projekt på en webbserver så skapas även en s.k. webbapplikation. En webbapplikation är alla filer i en mapp och filer i dess undermappar, som inte ingår i en annan webbapplikation. Webbapplikationer skapas bl.a. för att skydda dem, och dess resurser, från varandra.

1. Välj *Web Site…* från menyn *File→New*. Markera *ASP.NET Web Site* i dialogrutan *New Web Site*, välj språk (att använda i projekt) i listrutan *Language* och fyll i nedanstående beroende på vilken typ av webbserver du ska använda för att testa ASP.NET i:

IIS, lokal eller på annan dator:
Location: HTTP `http://<adress-till-dator>/sokvag/pa/dator`

(t.ex. `http://localhost/dotnet35/IntroASPNET35CS` om lokal dator eller `http://dator.doman.se/dotnet35/IntroASPNET35CS` om webbserver på annan dator)

Inbyggd testserver i VS:
Location: *File System* `<sökväg-till-projektmapp-på-dator>`

När projektet skapas så skapas även två filer som standard: `Default.aspx`[9] och `web.config`. Vi återkommer till dessa, d.v.s. låt dem vara för nu.

1.2.2 Lägga till webbsida med inbäddad kod

Dags att skapa en enkel webbsida med inbäddad kod för att visa på hur det fungerar. (Efterföljande exempel använder enbart bakomliggande kod.)

2. Lägg till en ny webbsida (webbformulär) genom att högerklicka på projektet (rad i fet stil som börjar med http://…, om du använder IIS enligt beskrivning ovan) i *Solution Explorer* (oftast längst upp till höger i VS) samt välj *Add New Item*….

[7] Andra webbservrar (t.ex. Apache) använder filnamnet `index.xxx` (ersätt `xxx` med `htm` eller `html` … eller `php`).
[8] Vill bara informera om att webbservern www.eki.mdh.se aldrig varit IIS (i.a.f. inte på senare tid – jag minns inte så långt tillbaka ☺). Men det är principen som gäller här...
[9] Denna `Default.aspx` använder bakomliggande kod.

3. Välj **Web Form** i listrutan *Templates*, byt namn på fil till `inbaddad.aspx` i textrutan *Name* samt kontrollera att kryssrutan *Place code in separate file* **inte** är markerad innan du klickar på knappen Add.
4. Öppna filen (om den inte öppnades automatiskt – byt även till källkodsläge, genom att klicka på fliken *Source* längst ner i editorn, om koden inte visas).

Det är nu dags att lägga till dynamisk kod. ☺

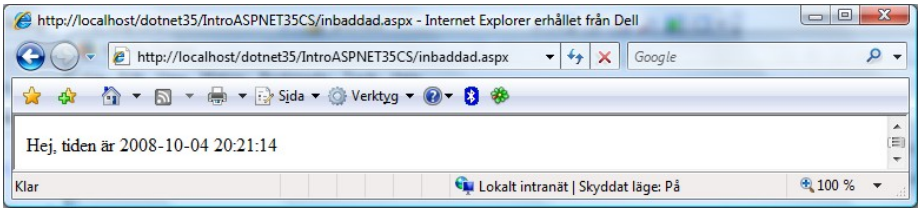
1.2.2.1 Lägga till dynamisk kod

Dynamisk kod är vad jag kallar kod som exekveras för att generera HTML-kod dynamiskt, d.v.s. skriptkod. Skriptkod kan infogas på ett antal olika sätt. Det klassiska (och det som kan användas för att ”infoga” det dynamiska i HTML-koden) är som med ASP v.3 (eller tidigare), d.v.s. mellan `<% %>`-taggarna. Denna typ av taggar bör dock undvikas till förmån för andra taggar, bl.a. etiketter (*labels*). Jag återkommer till denna andra typ av taggar… Först ett exempel med taggarna vi bör undvika. ☺

5. Lägg till nedanstående kod någonstans mellan HTML-taggarna `<body …>`[10] och `</body>`. Eftersom vi skriver ut HTML-koden dynamiskt så måste vi även skriva ut eventuella HTML-taggar också. Därför skriver vi ut stycketaggarna (`<p>`), lite ledtext samt aktuellt datum och tid. (Detta slipper vi om vi använder etiketterna jag återkommer till.)

```
<%
  Response.Write("<p>Hej, tiden är " + DateTime.Now + "</p>");
%>
```

För att titta på webbsidan så högerklickar man på den i *Solution Explorer* samt väljer *View in Browser* från menyn som visas.



1.2.2.2 HTML-formulär

I ”vanlig” ASP krävdes en hel del logik för att hantera HTML-formulär. Men i ASP.NET har det blivit lättare – mycket av logiken sköts av ASP.NET. För att kunna hantera denna logik så har Microsoft (bl.a.[11]) introducerat något som kallar webbserverkontroller (*web server controls*, eller bara webbkontroller). I HTML-koden lägger vi till prefixet `asp:` först i taggen samt använder attributet med värdet `runat="server"` för att tala om att servern ska hantera logiken.

I nedanstående exempel visas först HTML-koden för en ”vanlig” textruta och sedan motsvarande webbkontroll för jämförelse.

```
<input name="HtmlText" type="text" value="Text i textruta">

<asp:TextBox id="TextBox1" runat="server">Text i textruta</asp:TextBox>
```

[10] Det kan finnas ett eller flera attribut i inledande body-tagg…
[11] Med bl.a. menar jag att det finns andra typer av kontroller – kontroller som jag inte kommer behandla här. ☺

HTML-koden (som skickas till webbläsare) för webbkontroll ovan blir följande:

```
<input name="TextBox1" type="text" value="Text i textruta" id="TextBox1" />
```

För att webbformulär (*web forms*) ska fungera så måste vi placera webbkontrollerna inom formulärtaggen. I formulärtaggen måste (eller bör) vi även använda attributen `id` och `runat` enligt exempel nedan.

```
<form id="FormNamn" runat="server">
  <!-- Kontroller i formuläret -->
</form>
```

En viktig detalj är att formulär i ASP.NET oftast skickas till den webbsida där formuläret finns, vilket är skälet till att vi kan utelämna ACTION-attributet i FORM-taggen. ASP.NET lägger för övrigt till attributen ACTION och METHOD. D.v.s. (händelse-)hantering av formulär sker främst i samma webbsida som formuläret finns i. (I ASP var det vanligt att man hade en ”vanlig” HTML-sida med formulär och en ASP-sida för att hantera data från formuläret.)

1.2.2.3 Hantera händelser från HTML-formulär

Tanken med de nya webbkontrollerna och ASP.NET (s.k. *web forms*) är att utvecklingen av ett användargränssnitt för webben ska påminna om det för ”vanliga” Windows-applikationer (s.k. *win forms*). D.v.s. formuläret ska bygga på händelser (så som *Click*, d.v.s. att användare klicka på en knapp). Det är främst en typ av händelse, som genereras av webbkontroller, och händelsen Page_Load som är intressant (men inte de enda…).

Händelsen Page_Load kan t.ex. användas för att fylla webbkontroller med data från databaser. Vi behöver dock inte skriva logik för att se till att markerat alternativ i listrutor är det som användare valt när formulär skickades när formulär visas igen.

Metoderna som ska hantera händelserna, bl.a. Page_Load, placeras inom en script-tagg – oftast innan öppnande html-tagg. Eventuellt resultat från exekvering, som ska visas i webbsida, placeras i variabler. På detta sätt separeras logik från presentation.

```
<script language="CS" runat="server">
  //Här placeras händelsehanterare och andra metoder
</script>
<html xmlns=...>
  <!-- HTML-koden för webbsidan -->
```

Alla fördefinierade metoder och metoder för hantering av händelser (vad jag kan se nu) har två parametrar: källa för händelse (av typen `Object`) och argument från händelse (av typen `EventArgs` eller någon av dess subklasser).[12] I exempel nedan visas metoden `Page_Load()` för att hantera händelsen Page_Load.

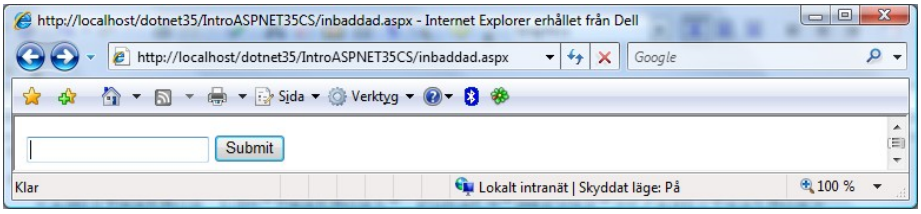
[12] Vi behöver inte använda oss av objekten i parametrarna… men dom måste finnas där. ☺

```
<script language="CS" runat="server">
  void Page_Load(Object source, EventArgs e)
  {
    //T.ex. initiering av kontroller med data från databas
  }
</script>
```

Ett typiskt exempel på en händelse är när användaren klickar på en knapp. I HTML-formulär är det främst Submit-knappen för formulär som gäller. I nedanstående exempelformulär finns en textruta, en knapp och en av nyheterna i ASP.NET – en etikett[13]. Tanken är att vi ska skapa en händelsehanterare för knapptryckningen som läser texten från textrutan och skriver ut den i etiketten.

Koden för formuläret visas nedan (och den resulterande webbsida).[14]

```
<form ID="form1" runat="server">
<div>
  <asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
  <asp:Button id="Button1" onclick="Control_Click" text="Submit"
    runat="server" />
  <asp:Label id="Label1" runat="server" />
</div>
</form>
```



Som vi ser av koden ovan så har alla webbkontroller attributet och värdet `runat="server"`. Detta talar om för ASP.NET att den ska hantera dess logik. Vi använder även attributet ID för att namnge webbkontrollerna.[15] På detta sätt kan vi använda värdet på attributet ID för att referera till kontrollerna i skriptkod (se mer i koden för händelsehanteraren nedan).

Textkontroller är annars ”inget speciellt” – om vi vill att text som standard ska placeras i textrutan (då webbsida visas första gången) så kan vi placera texten mellan den öppnande och avslutande taggen. Knappen skiljer sig en aning från en ”vanlig” HTML-knapp. Vi har i attributet `onclick` talat om viken metod i webbsidan som ska exekveras när formuläret skickas tillbaka till webbsidan-/servern. Definitionen för metoden visas i koden nedan. Etiketten är, som sagt, en ny typ av kontroll. Egentligen är det bara en plats i dokumentet vi kan referera till som ett namn (se kod nedan).

Koden för att hantera klickhändelsen visas nedan (och resulterande webbsida).

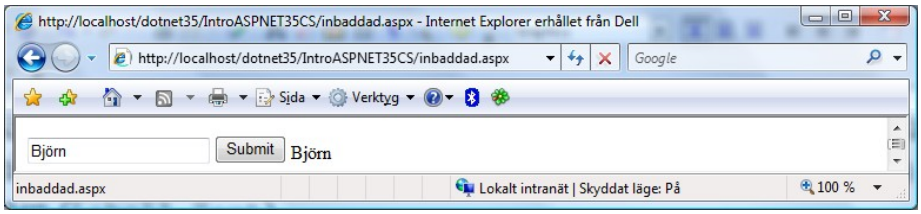
```
<script runat="server" language="cs">
```

[13] En etikett är egentligen bara en webbkontroll som ”konverteras” till vanlig text när ASP.NET ”exekverar” kontrollen. Men när vi designar webbsidan med VS.NET så ”agerar” den som en etikett i ett Windows-formulär.
[14]Observera att eftersom vi inte skrivit metoden ännu så kan du **inte** titta på webbsidan ännu!
[15] Inte alla HTML-taggar har attribut som NAME, etc.. Men genom att använda attributet ID (som alla HTML-taggar har) så kan vi ge alla kontroller unika namn. (Värden i attributet ID ska enligt HTML-specifikationen vara unika för alla kontroller.)

```
    void Control_Click(Object source, EventArgs e)
    {
      Label1.Text = TextBox1.Text;
    }
  </script>
```

För att titta på webbsidan så högerklickar vi, som tidigare, på den i Solution Explorer och väljer View in Browser från menyn som visas.

Som vi ser i koden så använder vi namnen på kontrollerna (eller värdet på attributet ID) för att referera till kontrollerna. I koden ovan så tilldelar vi innehållet i textrutan (`TextBox1.Text`) till etiketten (`Label1.Text`).



Observera att textrutan ”kommer ihåg” vad som fyllts i den – detta är vad som kallas *view state*, något som går att stänga av (och ibland bör göras).

1.2.3 Lägga till webbsida med bakomliggande kod (*code-behind*)

I detta exempel ska vi göra samma sak som i exempel ovan (med inbäddad kod). Skillnaden ligger i att vi ska använda oss av VS.NET för att göra det mesta av jobbet åt oss. Vi kommer (förhoppningsvis ☺) se hur lätt det är att skapa ett webbformulär (*web forms*) och att det påminner mycket om sättet vi gör det i ett ”vanligt” Windows-formulär (*win forms*).

1. Högerklicka på projektet (raden med fet text) i *Solution Explorer* och välj *Add New Item...* från menyn som visa.
2. Markera *Web Form* i listrutan *Templates*, fyll i `bakomliggande.aspx` i textrutan *Name*, kontrollera att *Language* visar *Visual C#* samt bocka för *Place code in separate file*. Klicka på *Add* för att lägga till filen.
3. Öppna filen (om den inte öppnades automatiskt) och byt till HTML-läge (genom att klicka på fliken *Source* längst ner i editor) samt titta på första raden: Page-direktivet.

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

I Page-direktivet talas om vilket språk som skriptkoden har i webbsidan, men även:

- `AutoEventWireup` – (inte viktigt nu ☺ – sök på *AutoEventWireup attribute* i MSDN för förklaring – klicka på @Page, eller *ASP.NET Web Server Control Event Model*, om Express Edition).[16]
- `CodeFile` – fil som innehåller (bakomliggande) kod (med bl.a. händelsehanterare).
- `Inherits` – klass som aktuell ASPX-sida ärver från (finns oftast i filen ovan).

Det är nu dags att lägga till dynamisk kod. ☺

[16] Eftersom VS vill ha den där så lämnar vi lämpligen attributet som det är.

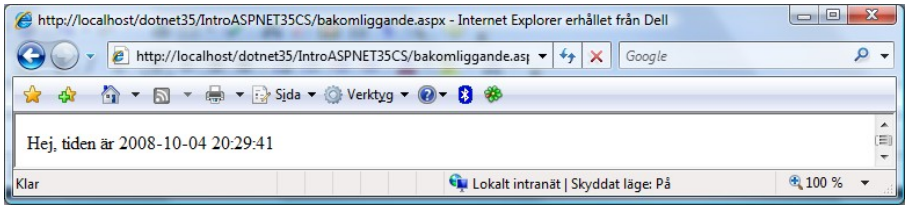
1.2.3.1 Lägga till dynamisk kod

Även med *code behind* kan vi använda `Response.Write` för att skriva ut till webbsidan. Men vi använder lämpligen etiketter till detta – i stort sett ett krav om vi använder GridLayout.

6. Dra och släpp en etikett (*Label*) från verktygsmenyn *Toolbox* (till vänster i VS) inuti den tomma div-taggen.
7. Växla till designläget (klicka på fliken *Design* längst ner i editor). Dubbelklicka på bakgrunden för att öppna kodfönstret och för att skapa metoden `Page_Load()`. Fyll i koden nedan i metoden.

```
Label1.Text = "Hej, tiden är " + DateTime.Now;
```

8. Bygg projektet (Shift+Ctrl+B) och visa webbsida genom att t.ex. högerklicka på den och välja View in Browser från meny som visas. Resultatet bör bli något i stil med skärmdump nedan.



(Det går att öppna en webbläsare och ”surfa” till webbsidan för att kunna växla mellan webbläsare och VS.NET. Glöm bara inte att bygga/kompilera applikationen innan du växlar till webbläsaren och laddar om webbsida.)

1.2.3.2 HTML-formulär

Även med *code behind* så använder vi webbkontroller.

1.2.3.3 Hantera händelser från HTML-formulär

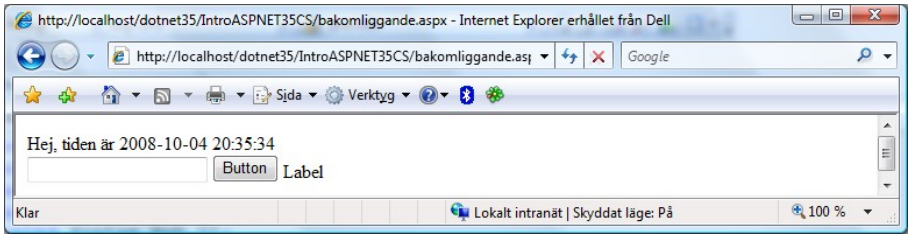
Återigen så hanteras formulär på samma sätt som med inbäddad kod, och åter igen använder vi VS för att göra mycket av jobbet… ☺

9. Dra och släpp en textruta, en knapp samt ytterligare en etikett på webbsidan (gärna i den ordningen som förra exempel).
10. Dubbelklicka på knappen för att skapa händelsehanteraren för knappen (antagligen med namnet `Button1_Click()`) och fyll i nedanstående kod[17] i metoden (anpassa eventuellt namn på etikett och textruta om dina kontroller fick andra namn).

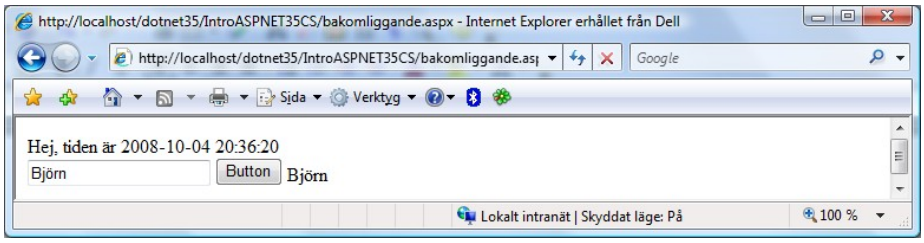
```
Label2.Text = TextBox1.Text;
```

11. Bygg projekt och ”kör” applikationen (eller växla till en webbläsare och ”surfa” till webbsidan). Resultatet bör se ut något i stil med nedanstående skärmdump.

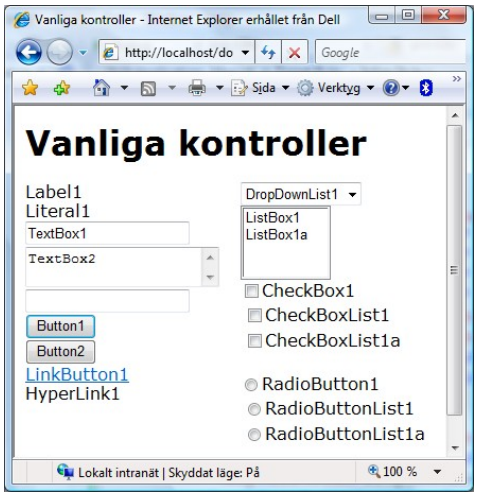
[17] Om webbkontrollerna har fått ett annat namn (om du ”lekt” lite) så får du anpassa namnen på kontrollerna. Har du följt beskrivningen ”till pricka” så bör det fungera. ☺



Prova att fylla i ditt namn i textrutan och klicka på knappen. Resultatet bör bli något i stil med nedanstående skärmdump.



1.2.4 Webb-(server-)kontroller i ASP.NET



Det finns ett antal olika sätt att skapa HTML-element i ASP.NET. I denna introduktion har jag valt att använda endast ett sätt – webbserverkontroller (eller bara webbkontroller, se bild till höger för exempel). Övriga typer är HTML-serverkontroller (*HTML server controls*), valideringskontroller (*validation controls*) och användarkontroller (*user controls* – kontroller som vi själva kan skapa), några som ni får läsa om någon annanstans ☺.

Nedan beskrivs och visas ASP.NET-koden för webbkontroller och sedan koden i *code behind* (kallat kodfilen) för kontrollerna som visas i bilden ovan. (En skillnad mot ASP.NET 1.x är att vi inte längre kan se variabler som deklareras för motsvarande kontroller i webbformuläret, d.v.s. ASPX-filen). Sist av allt visas den resulterande HTML-koden (som webbkontrollerna resulterar i) som skickas till webbläsaren, främst för ”oss” som kommer från äldre teknologier som ASP och PHP för att (förhoppningsvis) lättare förstå relationen mellan webbkontroller och HTML-element.

1.2.4.1 Koden för kontroller

I ASP-koden (eller snarare ASP.NET-koden) så har taggar för webbserverkontrollerna ett antal saker gemensamt:

- Kontrollerna börjar med `<asp:Kontrolltyp` (ersätt `Kontrolltyp` med typ av kontroll som önskas, t.ex. `TextBox`, `Button` eller `ListBox`).
- Attributet `id` används för att namnge kontrollen, d.v.s. unikt kunna identifiera dem. Detta namn används i kod som variabel namn för att manipulera kontrollerna.

- Attributet `runat`, med värdet `"server"`, som talar om för ASP.NET att koden ska tolkas (”exekveras”) av server innan webbsida skickas till klient.
- Attributet `text` som innehåller texten som kontrollen ska visa (om någon). Detta attribut kan ibland utelämnas samt dess värde placeras mellan öppnande och avslutande tagg.
- Kontrollerna måste ha en avslutande tagg, `</asp:KontrollTyp>`, eller den öppnande taggen avslutas i XHTML-stil med ett snedstreck (`<asp:Kontrolltyp ... />`).

1.2.4.2 Egenskaper för kontroller

Egenskaper (*properties*) för kontroller avser främst kontrollers utseende, däribland text, men även dess beteende, d.v.s. hur de ska reagera vid t.ex. tangentbordstryckningar (kortkommando[18], m.m.) och musrörelser. Dessa egenskaper kan ändras vid design i egenskapsfönstret (*Properties*) eller via kod, d.v.s. med objekten som motsvarar kontrollerna.

Egenskaperna för kontrollerna motsvarar till viss del attributen i ASP.NET-taggarna de flesta kontrollerna har ett antal gemensamma egenskaper, bl.a.

- `Text` – text som ska visas i kontrollen.
- `ID` – namn på kontrollen.[19]
- `Visible` – om kontroll ska visas i den resulterande webbsidan.

1.2.4.3 Partiella klasser

Fr.o.m. ASP.NET 2.0 används, vad som kallas, partiella klasser. Klasser för webbformulär delas upp i två delar: en del som innehåller kod som utvecklare skriver och en del som genereras av ASP.NET. Detta, anser jag, är en förbättring av hur ASP.NET fungerar då vi inte behöver ”stöka till” koden som vi måste redigera, d.v.s. endast kod som vi bör kunna redigera är tillgänglig. Det innebär också att vi ”tappar kontrollen”, något som några av oss kanske inte är så förtjusta i. ☺

1.2.4.4 Etiketter (*Label*)

Etiketter (*labels*) är egentligen bara text, men med webbserverkontroller så är det objekt som vi kan ändra egenskapen `Text` för att ändra vad som visas för besökaren. D.v.s. det är den kontroll som är lämpligast för att visa (icke-redigerbar) data för besökare (se även Literaler nedan).

ASP-koden för etiketter i sin enklaste form visas nedan (som den visas i bild med exempel på kontroller ovan).

```
<asp:Label ID="Label1" runat="server" Text="Label1"></asp:Label>
```

Resulterande HTML-kod för etiketter blir följande:

```
<span id="Label1">Label1</span>
```

[18] Kortkommando är tangentbordstryckningar som oftast involverar tangenterna Shift, Ctrl och/eller Alt i kombination med en annan tangent.

[19] Denna egenskap kan kanske kännas lite överflödig eftersom dess värde bör motsvara namnet på objektets variabel. Men egenskapen kan vara användbar om vi skulle loopa över vektor med kontroller t.ex..

1.2.4.5 Literaler (*Literal*)

Literaler (*literals*) är en ny kontroll fr.o.m. ASP.NET 2.0 och är **bara** text. I motsats till etiketter så genereras **inga** HTML-taggar (d.v.s. SPAN-tagg) i resulterande hemsida. Precis som med etiketter så är webbserverkontrollen ett objekt som vi kan ändra egenskapen `Text` för att ändra vad som visas för besökaren. D.v.s. det är den kontroll som är lämpligast för att visa (icke-redigerbar) data för besökare som **inte** ska skilja sig från omgivande text (se även Etiketter ovan).

ASP-koden för etiketter i sin enklaste form visas nedan (som den visas i bild med exempel på kontroller ovan).

```
<asp:Literal ID="Literal1" runat="server" Text="Literal1"></asp:Literal>
```

Resulterande HTML-kod för literal blir följande (d.v.s. enbart text och inga HTML-taggar):

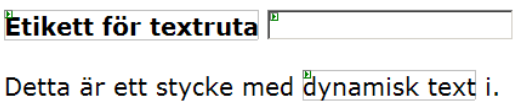
```
Literal1
```

1.2.4.5.1 När literaler respektive etiketter bör användas

Både literaler och etiketter är användbara för att infoga dynamisk text i webbsidor, d.v.s. text som kan ändras från programkod. Literaler är användbara när vi ska infoga dynamisk text i t.ex. löpande text eller rubriker där den dynamiska texten ska se likadan ut som den omgivande text. Etiketter däremot bör användas för just det – som etiketter – eller om vi vill att den dynamiska texten ska kunna formateras annorlunda än den omgivande text.

I exempel nedan finns två stycken som visar på hur etiketter respektive literaler kan användas. Det första stycket innehåller etikett (`Label1` – med fet stil) och textruta (`Textbox4`) samt det andra stycket statisk text och literal (`Literal2` – ASP.NET-kod för etikett respektive literal har markerats med fet stil). Resulterande hemsida och HTML-kod visas nedanför.

```
<p><asp:Label ID="Label2" runat="server" Font-Bold="True"
  Text="Etikett för textruta"></asp:Label>
  <asp:TextBox ID="TextBox4" runat="server"></asp:TextBox></p>
<p>Detta är ett stycke med
  <asp:Literal ID="Literal2" runat="server" Text="dynamisk
text"></asp:Literal>
  i.</p>
```




```
<p><span id="Label2" style="font-weight:bold;">Etikett för textruta</span>
  <input name="TextBox4" type="text" id="TextBox4" /></p>
<p>Detta är ett stycke med
  dynamisk text
  i.</p>
```

1.2.4.6 Textrutor (*TextBox*)

Textrutor är den huvudsakliga kontrollen för att hämta data från besökare av webbplatser. Det finns tre ”typer” av textrutor, motsvarande de i HTML-formulär, vars typ avgörs av attributet `TextMode` och dess värde:

- Avsaknaden av attribut (och värde) – enradig textruta.

- `"MultiLine"` – flerradig textruta.
- `"Password"` – lösenordsruta, d.v.s. textruta där tecken visas som asterisker (”*”).

ASP-koden för textrutor i sina enklaste former – enradig textruta, flerradig textruta och lösenordsruta – visas nedan.

```
<asp:TextBox id="TextBox1" runat="server">TextBox1</asp:TextBox>
<asp:textbox id="TextBox2" runat="server"
  TextMode="MultiLine">TextBox2</asp:textbox>
<asp:TextBox ID="TextBox3" runat="server"
  TextMode="Password">TextBox3</asp:TextBox>
```

Resulterande HTML-kod för de tre typerna av textrutor blir följande:

```
<input name="TextBox1" type="text" value="TextBox1" id="TextBox1" />
<textarea name="TextBox2" rows="2" cols="20"
id="TextBox2">TextBox2</textarea>
<input name="TextBox3" type="password" id="TextBox3" />
```

1.2.4.7 Knappar (*Button*)

Knappkontrollen motsvarar submit-knappar i HTML-formulär, en kontroll som bl.a. har följande egenskaper:

- `OnClick` – metod att utföra när knapp klickas på (se mer nedan).[20] (Lättaste sättet att sätta detta värde är att byta till designläge och sen dubbelklicka på knappen. Då sätts värdet på denna egenskap och signatur för metoden skapas i bakomliggande kod.)
- `CommandName` - ”namn” på kommando. Användbart för att skilja på vilken knapp som klickats på om man vill använda samma metod att utföra oavsett vilken knapp som klickas på, d.v.s. flera knappar (eller andra kontroller) kan använda samma händelsehanterare men kan skiljas åt via kommandonamnet (se mer nedan).

```
<asp:button id="Button1" runat="server" Text="Button1"
  onclick="Button1_Click" />
<asp:Button id="Button2" runat="server" Text="Button2"
  CommandName="cmdButton2" onclick="Button2_Click" />
```

Resulterande HTML-kod för knappar visas nedan (d.v.s. de två sista egenskaperna påverkar inte resulterande HTML-kod).

```
<input type="submit" name="Button1" value="Button1" id="Button1" />
<input type="submit" name="Button2" value="Button2" id="Button2" />
```

1.2.4.7.1 Hantera klickhändelse från knappar (och länkknappar)

Lättaste sättet att koppla en s.k. händelsehanterare till en knapp är att dubbelklicka på knappen i designläge. Detta lägger till egenskapen (attributet) `onclick` i kontrollens ASP.NET-kod samt skapar metodens signatur i bakomliggande kod. Fil med bakomliggande kod (ASPX.CS-fil) öppnas och markören placeras i metoden, klar för att börja lägga till kod i

[20] I VB.NET behövs inte denna egenskap – man kan använda det reserverade ordet Handles efter metodsignatur för att ange vilken, eller vilka, kontroller och händelser som metod ska exekveras för.

den. Nedan visas kod som skapas när man dubbelklickat på en knapp med namnet (id:t) `Button1`. Exempel på hur man kan använda sig av händelsehanteraren visades i exempel med bakomliggande kod ovan (d.v.s. hämtade värde från textruta och tilldelade till en etikett) och kommer att ske upprepade gånger i kommande exempel.

```
protected void Button1_Click(object sender, EventArgs e)
{

}
```

Metoden har alltid (eller i.a.f. oftast) två argument:

- `sender` av typen `Object` – kontroll (objekt) som genererade händelse, som man därmed kan fråga om mer information (t.ex. dess värden eller värden). (Man använder typen/klassen `Object` så att metoden kan användas för flera olika typer av kontroller. Vill man därför använda objektet i argumentet, för att t.ex. hämta värden, så måste man konvertera [*cast*] till rätt typ av kontroll – mer om detta nedan[?]).
- `e` av typen `EventArgs` (eller någon av dess subklasser) – innehåller eventuell övriga data som avser händelse (mer om detta nedan[?]).

Om en metod hanterar flera händelser, vilket är praktiskt om händelserna ska exekvera en större mängd gemensam kod, så kan vi t.ex. använda attributet `CommandName` i webbkontrollen. Vi använder då webbkontroll som genererade händelse, i argumentet `sender`, för att hämta kommandonamnet, med vilket vi kan avgöra vilken avvikande kod (relativt andra kontroller) som ska exekveras när kontroll genererar händelse.

I exempel nedan (se resultat i bild till höger) så finns två knappar och en nedrullningsbar listruta. Knapparna har givits texten `ButtonX` resp. kommandonamnet ”KnappX” (ersätt X med 1 el. 2) och listrutan har två alternativ så att det går att välja.[21] Knapparna har en klickhändelse (onclick) vilket kopplats till samma metod medan listrutan har en index-ändratshändelse (OnSelectedIndexChanged) som också kopplats till samma händelsehanterare (markerat med fet stil i ASP.NET-kod nedan). Etiketten, sist i koden, används för att visa resultat.

```
<asp:Button ID="Button1" runat="server" Text="Button1"
    onclick="Button1_Click" CommandName="Knapp1" />
<asp:Button ID="Button2" runat="server" Text="Button2"
    onclick="Button1_Click" CommandName="Knapp2" />
<asp:DropDownList ID="DropDownList1" runat="server"
    OnSelectedIndexChanged="Button1_Click" AutoPostBack="True">
  <asp:ListItem>Alternativ 1</asp:ListItem>
  <asp:ListItem>Alternativ 2</asp:ListItem>
</asp:DropDownList></p>
<p><asp:Label ID="Label1" runat="server" Text="Label"></asp:Label></p>
```

I koden för händelsehanterare testas först om det är en knapp som genererat händelsen (d.v.s. är ”utlösare”), och om så är fallet så tilldelas knappens kommandonamn till etikettens text. Här måste först konvertering (*cast*) ske, från `Object` till `Button`, innan vi kan fråga om värdet

[21] OK, så jag fuskade och tog bort det första alternativet som syns i bilden. Men det gjordes för att spara lite plats och göra exempel mindre komplext. :-)

på dess egenskap! Annars så testas om ”utlösare” är en nedrullningsbar listruta, och om så är fallet så tilldelas värdet i valt alternativ till etikettens text.[22] Även här måste konvertering ske!

```
protected void Button1_Click(object sender, EventArgs e)
{
  //Om utlösare är en knapp - anv. egenskap CommandName
  if (sender is Button)
    Label1.Text = ((Button)sender).CommandName;
  //... annars om utlösare listruta - anv. valt värde
  else if (sender is DropDownList)
    Label1.Text = ((DropDownList)sender).SelectedValue;
}
```

1.2.4.8 Länkknappar (*LinkButton*)

Länkknappar (*link buttons*) används istället för ”vanliga” knappar, d.v.s. inte för länkar till andra webbsidor (se Länkar nedan för detta). Funktionen är teoretiskt den samma som för knapparna ovan, och de har de flesta egenskaperna gemensamt, bl.a. `CommandName` och `onclick`. Vi ersätter, i stort sett, bara `Button` med `LinkButton` i kod för knapparna ovan. Nedan visas ASP.NET-koden (se koden för ”vanliga” knappar ovan för hur länkknappars händelser kan kopplas till metoder och hanteras).

```
<asp:LinkButton id="LinkButton1"
  runat="server">LinkButton1</asp:LinkButton>
```

Nedan visas HTML-koden som kontrollen genererar. I attributet `href` anropas en JavaScript-funktion som skickar webbformuläret (d.v.s. generera submit-händelsen). JavaScript-funktionen genereras av ASP.NET (liksom 2 dolda textfält[23]).

```
<a id="LinkButton1"
  href="javascript:__doPostBack('LinkButton1','')">LinkButton1</a>
```

Observera att en länkknapp resulterar i att JavaScript-kod krävs för dess funktion, och om någon nitisk besökare inte tillåter JavaScript i sin webbläsare så förhindras besökaren från att utnyttja webbplatsens funktion! Länkknappar bör alltså användas restriktivt!

1.2.4.9 Länkar (*HyperLink*)

En länk (*hyper link*) är en länk. ☺ Länkar genererar inga ASP-händelser, men skickar dock besökare till URL som länk refererar till. Nedan visas ASP.NET-kod och HTML-kod som genereras.

```
<asp:HyperLink ID="HyperLink1" runat="server"
  NavigateUrl="ny_sida.aspx">HyperLink1</asp:HyperLink>
```

```
<a id="HyperLink1" href="ny_sida.aspx">HyperLink1</a>
```

[22]Här använder jag inte bara en else-sats – man vet aldrig om någon utökar exempel med fler typer av kontroller, något som skulle generera fel om det inte var en listruta!

[23] De dolda fälten är `__EVENTTARGET` och `__EVENTARGS` som används av ASP.NET för att avgöra vilken kontroll i formuläret (d.v.s. dess namn) som genererade händelse samt eventuella parametrar som ska bifogas.

Man kan, som exempel ovan visar på, undra varför denna kontroll finns? Ett skäl är att man med en länkkontroll kan ändra adressen som länken ”pekar på” med programkod. Ytterligare en fördel är att man kan använda en relativ sökväg som anpassas efter var webbapplikationen (d.v.s. projektet som webbsida ingår) installeras. (Sökvägen har formen ~/mapp/fil.ext, där ~-tecknet syftar på webbapplikationens rot. Se mer intelligenta personers webbsidor för mer information. ☺) Ett tillfälle då man bör använda en länkkontroll istället för en ”vanlig” länk är när man försöker skydda webbsidor med inloggning och använder s.k. *URL rewriting*!

Intressanta egenskaper (utöver de för de flesta kontroller ☺) är:

- `ImageUrl` – användas för att ange bild för länk (om ej text används för länk)
- `NavigateUrl` – används för att sätta URL som länk refererar till (`ny_sida.aspx` i exempel ovan)
- `Target` – hur webbsida ska öppnas (t.ex. nytt fönster eller eventuell ram).

(I kod så är det främst intressant att t.ex. ändra dessa egenskaper beroende på val och händelser, som inte genereras av länkkontrollen, i ett webbformulär.)

1.2.4.10 Nedrullningsbara listrutor (*DropDownList*)

Nedrullningsbara listrutor (*drop down lists*) är användbara för att låta besökare välja ett alternativ från en begränsad lista. Precis som motsvarande HTML-kontroll (*select*-taggen) så kan varje alternativ i en listruta ha ett värde som skickas (*value*) och ett värde som visas (*item*). Listan kan fyllas dynamiskt av bl.a. vektorer (*arrays*) eller data från datakällor, men även med statiska (fasta) värden m.h.a. ListItem-taggen. Intressant egenskaper är bl.a.

- `AutoPostBack` – om formulär ska skickas när valt alternativ ändras i listruta.
- `DataSource`[24] – datakälla för alternativ i listruta.
- `Items` – vektor för alternativ i listruta.

Nedan visas ASP.NET-kod och HTML-kod för listruta med ett (statiskt) alternativ som har samma värde som skickas och visas.

```
<asp:DropDownList id="DropDownList1" runat="server">
  <asp:ListItem Value="DropDownList1"></asp:ListItem>
</asp:DropDownList>
```

```
<select name="DropDownList1" id="DropDownList1">
  <option value="DropDownList1">DropDownList1</option>
</select>
```

1.2.4.10.1 Fylla en listruta med alternativ

Två sätt att fylla en listruta med kod är att lägga till alternativ via kontrollens egenskap `Items` eller tilldela en vektor till dess egenskap `DataSource`. Om vi använder egenskapen `DataSource` så måste vi även anropa listrutans metod `DataBind()` så att listrutan fylls. Exempel på bägge dessa sätt visas nedan, (observera att värde som skickas [*value*] och visas [*text*] är den samma. För exempel hur vi kan visa ett värde, t.ex. beskrivning, och skicka ett annat, t.ex. primärnyckel – se kapitel med mer avancerad ASP.NET nedan.)

[24] I kombination med `DataSource` så är även egenskaperna `DataMember`, `DataTextField`, `DataTextFormatString` och `DataValueField` intressanta. Dessa behandlas i kapitel med mer avancerad ASP.NET nedan.

ASP.NET-koden innehåller tre stycken listrutor.

```
<asp:DropDownList ID="ddlItems" runat="server"></asp:DropDownList>
<asp:DropDownList ID="ddlArray" runat="server"></asp:DropDownList>
<asp:DropDownList ID="ddlArrayList" runat="server"></asp:DropDownList>
```

I webbsidans metode Page_Load() läggs nedanstående kod till. Först deklareras variabler för vektorn (*array*), vilken tilldelas värde samtidigt, och samlingsobjekt (*collection*, `ArrayList`). Därefter fylls första listrutan m.h.a. av sin egenskap Items (d.v.s. en vektor) för att sen tilldela vektorn som datakälla samt binda data i andra listrutan. Innan vi kan tilldela, och binda, den tredje listrutan till samlingsobjektet måste vi först skapa objektet och fylla den med värden.

```
String[] arrVektor = { "Fyra", "Fem", "Sex" };
ArrayList arlSamling;

//Lägg till i listruta med egenskapen Items
ddlItems.Items.Add("Ett");
ddlItems.Items.Add("Två");
ddlItems.Items.Add("Tre");

//Lägg till i listruta m.h.a. en vektor
ddlArray.DataSource = arrVektor;
ddlArray.DataBind();      //Koppla data till kontrol

//Lägg till i listruta m.h.a. samlingsklass (ArrayList)
arlSamling = new ArrayList();
arlSamling.Add("Sju");
arlSamling.Add("Åtta");
arlSamling.Add("Nio");
ddlArrayList.DataSource = arlSamling;
ddlArrayList.DataBind();  //Koppla data till kontrol
```

Resultatet av ovanstående exempel visas till höger.

I programkod får vi, förutom `Items` och `DataSource` som i exempel ovan, tillgång till ett antal fler intressanta egenskaper, bl.a.

- `SelectedIndex` – index (ordningstal) på valt alternativ i listruta.
- `SelectedItem` – bakomliggande objekt för valt alternativ i listruta. Denna egenskap används främst när komplexa objekt (som med t.ex. `Hashtable`) läggs till i listruta.
- och `SelectedValue` – text som visas för valt värde.

Listrutor har en händelse *Selected Index Changed* som uppstår när besökare väljer ett nytt alternativ i listruta. Denna händelse uppstår även om knapphändelse sker (d.v.s. en knapp klickas på), d.v.s. vi skulle kunna hantera båda händelser (om det skulle vara nyttigt ☺).[25]

```
protected void ddlArrayList_SelectedIndexChanged(object sender,
  EventArgs e)
{
  //Kod att exekvera om valt alternativ i listruta ändras
}
```

[25] Intresseklubben informerar: När jag testade så utfördes listrutans händelsehanterare innan knappens. ☺

1.2.4.11 Listrutor (*ListBox*)

Listrutor (*list boxes*, d.v.s. listrutor som inte rullas ner) är användbara om besökare ska kunna välja ett eller flera alternativ från en begränsad lista – egenskapen `SelectionMode` används för att avgöra om en eller flera kan väljas (se exempel i nästa avsnitt). En listrutas funktion är till stor del den samma som för nedrullningsbara listrutor, d.v.s. se kod för nedrullningsbara listrutor ovan för hur de används. Den stora skillnaden är att besökare kan ges möjligheten att välja flera alternativ (se nedan för hur detta kan hanteras).

```
<asp:ListBox id="ListBox1" runat="server">
  <asp:ListItem Value="ListBox1"></asp:ListItem>
  <asp:ListItem Value="ListBox1a"></asp:ListItem>
</asp:ListBox>
```

```
<select size="4" name="ListBox1" id="ListBox1">
  <option value="ListBox1">ListBox1</option>
  <option value="ListBox1a">ListBox1a</option>
</select>
```

1.2.4.11.1 Hantera flera valda alternativ i listruta

```
<asp:ListBox ID="lstAlternativ" runat="server" SelectionMode="Multiple">
  <asp:ListItem>Alterantiv 1</asp:ListItem>
  <asp:ListItem>Alterantiv 2</asp:ListItem>
  <asp:ListItem>Alterantiv 3</asp:ListItem>
</asp:ListBox>
```

Om flera alternativ kan väljas i listruta så måste vi loopa över listrutans egenskap `Items` och testa om aktuellt alternativ är valt (i t.ex. händelsehanterare för en knapp). I exempel nedan används en listruta, med namnet `cblAlternativ`, och en etikett, med namnet `lblKlickadeAlt`, för att loopa över listrutans alternativ, kontrollera om alternativet är valt samt skriva ut värdet för valda alternativ.

```
//Loopa över alla alternativ i listruta
foreach (ListItem li in lstAlternativ.Items)
{
  //Om alternativ valt - lägg till dess värde i etikett
  if (li.Selected)
    lblMarkeradeAlt.Text += " " + li.Value;
}
```

1.2.4.12 Kryssrutor (*CheckBox*)

Kryssrutor används då besökare ska ha ett begränsat antal alternativ att välja från (med listrutor så är antalet i stort sett obegränsat) och där multipla alternativ ska kunna väljas. Det finns två webbkontroller för kryssrutor (*check boxes*): `CheckBox` för enstaka kryssrutor och `CheckBoxList` för en lista med kryssrutor. Fördelen med den senare är att vi kan skapa alternativen med kod när webbformulär laddas (precis som med listrutor, d.v.s. med egenskapen `Items` ☺) liksom vi kan loopa över vektorn `Items` för att avgöra vilka ”alternativ” i kryssrutelistorna som bockats för.

En intressant (eller kanske inte ☺) egenskap med de enkla kryssrutornas taggar är att vi **inte** får placera text mellan öppnande och avslutande tagg utan måste använda attributet `Text` – däremot kan vi göra det med kryssrutelistorna (se exempel nedan).

Intressanta egenskaper är `Selected` som kan användas för att avgöra om kryssruta blivit förbockad.

```
<asp:checkbox id="CheckBox1" runat="server"
  Text="CheckBox1"></asp:checkbox>

<asp:checkboxlist id="CheckBoxList1" runat="server">
  <asp:ListItem Value="cbl1a">CheckBoxList1a</asp:ListItem>
  <asp:ListItem Value="cbl1b">CheckBoxList1b</asp:ListItem>
</asp:checkboxlist>
```

```
<input id="CheckBox1" type="checkbox" name="CheckBox1" /><label
  for="CheckBox1">CheckBox1</label>

<table id="CheckBoxList1" border="0">
  <tr>
    <td><input id="CheckBoxList1_0" type="checkbox"
      name="CheckBoxList1$0" /><label
      for="CheckBoxList1_0">CheckBoxList1</label></td>
  </tr><tr>
    <td><input id="CheckBoxList1_1" type="checkbox"
      name="CheckBoxList1$1" /><label
      for="CheckBoxList1_1">CheckBoxList1a</label></td>
  </tr>
</table>
```

1.2.4.12.1 Enkla kryssrutor

Om vi använder enkla kryssrutor (`CheckBox`) så kan vi kontrollera om respektive kryssruta bockats för med dess egenskap `Selected` enligt nedan. En egenskap som vi kan använda för att bocka för (eller avbocka) en kryssruta med kod genom att tilldela värdet `True` (eller `False`).

```
if (CheckBox1.Checked)
{
  //Kod att exekvera om kryssruta bockats för
}
```

1.2.4.12.2 Kryssrutelistor

För att lägga till alternativ i en kryssrutelista (`cblAlternativ`) med kod kan vi skriva kod enligt nedan.

```
cblAlternativ.Items.Add("Fyra");
```

Om vi har en kryssrutelista med namnet `cblAlternativ` så skulle vi kunna loop över dess vektor `Items` och skriv ut värdet för förbockade alternativ i en etikett (`lblKlickadeAlt`).

```
//Loopa över alla positioner i kryssrutelista
foreach (ListItem li in cblAlternativ.Items)
{
  //Om förbockad - lägg till dess värde i etikett
  if (li.Selected)
    lblKlickadeAlt.Text += " " + li.Value;
}
```

Sammanfattningsvis så är kryssrutelistor praktiska för att skapa kryssrutor dynamiskt, men även om vi inte skapar dem dynamiskt så är de praktiska för att avgöra vilka kryssrutor som bockats för (genom att loopa över `Items` som ovan istället för en massa if-satser).

1.2.4.13 Radioknappar

Radioknappar påminner mycket om kryssrutor, dock med några skillnader. Även radioknappar används för att besökare ska kunna välja bland ett begränsat antal alternativ, men de placeras i grupper där endast ett alternativ ska kunna väljas (med nedrullningsbara listor är antalet alternativ i stort sett obegränsade). Det finns två webbkontroller även för radioknappar (*radio buttons*): `RadioButton` för att själv bygga upp grupper och `RadioButtonList` för att grupper ska behandlas som en kontroll. För att gruppera de enskilda radioknapparna använder vi egenskapen `GroupName`, d.v.s. vi ger deras egenskap samma värde (namn), så att endast en av radioknapparna med samma gruppnamn kan väljas.

```
<asp:RadioButton ID="RadioButton1" runat="server" Text="Radioknapp 1"
  GroupName="EgenGrupp" />
<asp:RadioButton ID="RadioButton2" runat="server" Text="Radioknapp 2"
  GroupName="EgenGrupp" />

<asp:RadioButtonList ID="rblAlternativ" runat="server">
  <asp:ListItem>Alternativ 1</asp:ListItem>
  <asp:ListItem>Alternativ 2</asp:ListItem>
  <asp:ListItem>Alternativ 3</asp:ListItem>
</asp:RadioButtonList>
```

```
<input id="RadioButton1" type="radio" name="EgenGrupp" value="RadioButton1"
  /><label for="RadioButton1">Radioknapp 1</label>
<input id="RadioButton2" type="radio" name="EgenGrupp" value="RadioButton2"
  /><label for="RadioButton2">Radioknapp 2</label>

<table id="rblAlternativ" border="0">
  <tr>
    <td><input id="rblAlternativ_0" type="radio" name="rblAlternativ"
      value="Alternativ 1" /><label for="rblAlternativ_0">Alternativ
      1</label></td>
  </tr><tr>
    <td><input id="rblAlternativ_1" type="radio" name="rblAlternativ"
      value="Alternativ 2" /><label for="rblAlternativ_1">Alternativ
      2</label></td>
  </tr><tr>
    <td><input id="rblAlternativ_2" type="radio" name="rblAlternativ"
      value="Alternativ 3" /><label for="rblAlternativ_2">Alternativ
      3</label></td>
  </tr>
</table>
```

1.2.4.13.1 Enkla radioknappar

För att avgöra om en enskild radioknapp är markerad så används egenskapen `Checked`.

```
if (RadioButton1.Checked)
{
  //Kod att exekvera om radioknapp markerats
}
```

1.2.4.13.2 Radioknappslistor

Även radioknappslistor kan skapas med kod, d.v.s. dynamiskt.

```
RadioButtonList1.Items.Add("Fyra");
```

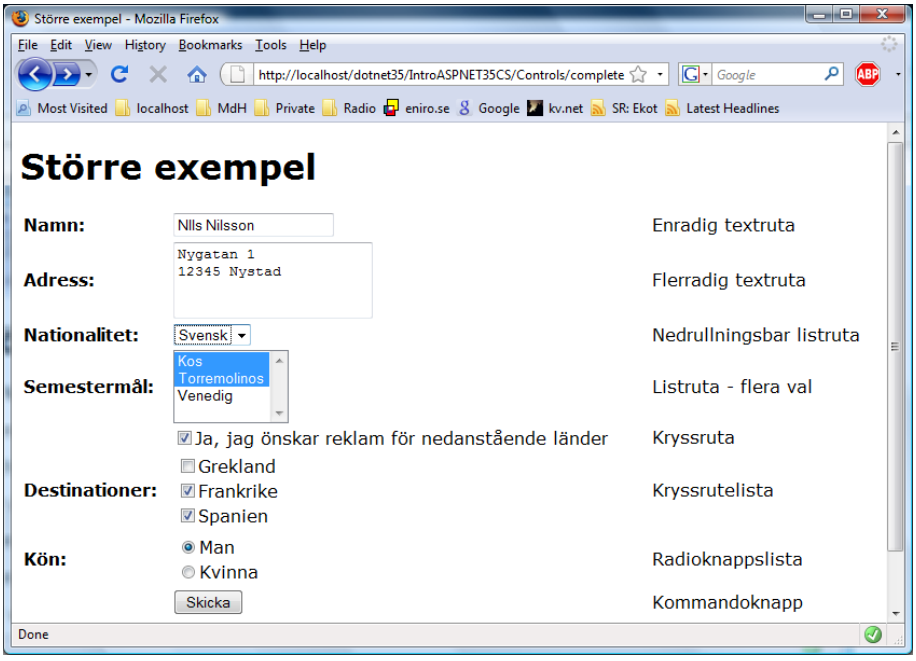
Om vi vill veta vilket alternativ som valts i en radioknappslista kan vi använda någon av egenskaperna `SelectedIndex`, `SelectedItem` eller `SelectedValue`.

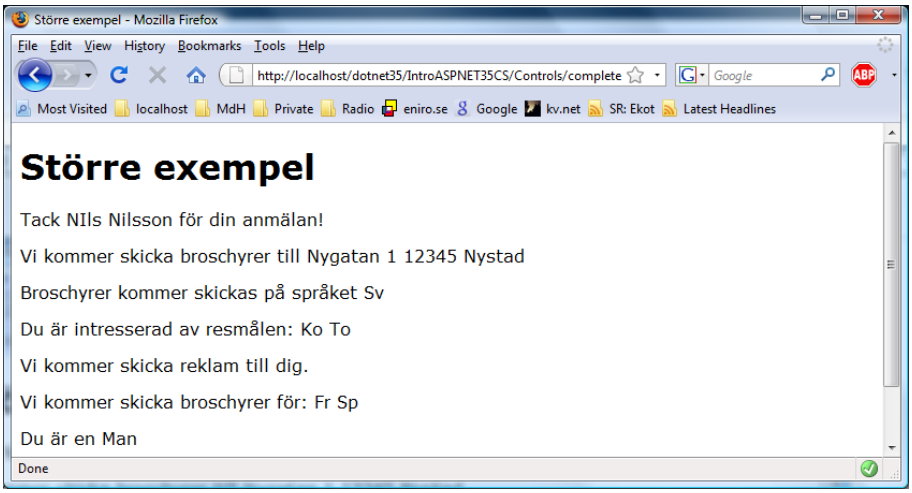
```
lblValtAlternativ2.Text = RadioButtonList1.SelectedValue;
```

Sammanfattningsvis så är radioknappslistor (liksom kryssrutelistor) mer praktiska en en enkel radioknapp, men här behöver man inte loopa över någon vektor för att ta reda på vilket alternativ som valts.

1.2.5 Exempel med kontroller

Exempel nedan visar hur en webbsida med webbkontroller kan se ut och hur vi kan hämta värden från webbkontrollerna. Nedan visas en bild på formulär ifyllt av en besökare och resultatet efter att besökare skickat formuläret.





ASP.NET-koden för formulär visas nedan.

```
<form id="form1" runat="server">
  <div>
    <h1>Större exempel</h1>
    <!-- Panel med formular -->
    <asp:Panel ID="pnlForm" runat="server">
      <!-- Tabell for design -->
      <table style="width: 800px;">
        <tr>
          <td><b>Namn:</b></td>
          <td><asp:TextBox ID="txtNamn" runat="server"></asp:TextBox></td>
          <td>Enradig textruta</td>
        </tr>
        <tr>
          <td><b>Adress:</b></td>
          <td><asp:TextBox ID="txtAdress" runat="server" Rows="3"
            TextMode="MultiLine"></asp:TextBox></td>
          <td>Flerradig textruta</td>
        </tr>
        <tr>
          <td><b>Nationalitet:</b></td>
          <td>
            <asp:DropDownList ID="ddlNationalitet" runat="server">
              <asp:ListItem Value="Da">Dansk</asp:ListItem>
              <asp:ListItem Value="No">Norsk</asp:ListItem>
              <asp:ListItem Value="Sv">Svensk</asp:ListItem>
            </asp:DropDownList>
          </td>
          <td>Nedrullningsbar listruta</td>
        </tr>
        <tr>
          <td><b>Semestermål:</b></td>
          <td>
            <asp:ListBox ID="lstSemestermal" runat="server"
              SelectionMode="Multiple">
              <asp:ListItem Value="Ko">Kos</asp:ListItem>
              <asp:ListItem Value="To">Torremolinos</asp:ListItem>
              <asp:ListItem Value="Ve">Venedig</asp:ListItem>
            </asp:ListBox>
          </td>
          <td>Listruta - flera val</td>
        </tr>
        <tr>
          <td><b></b></td>
          <td><asp:CheckBox ID="cbxReklam" runat="server"
              Text="Ja, jag önskar reklam för nedanstående länder" /></td>
          <td>Kryssruta</td>
        </tr>
        <tr>
          <td><b>Destinationer:</b></td>
```

```
            <td>
              <asp:CheckBoxList ID="cblDestinationer" runat="server">
                <asp:ListItem Value="Gr">Grekland</asp:ListItem>
                <asp:ListItem Value="Fr">Frankrike</asp:ListItem>
                <asp:ListItem Value="Sp">Spanien</asp:ListItem>
              </asp:CheckBoxList>
            </td>
            <td>Kryssrutelista</td>
          </tr>
          <tr>
            <td><b>Kön:</b></td>
            <td>
              <asp:RadioButtonList ID="rblKon" runat="server">
                <asp:ListItem>Man</asp:ListItem>
                <asp:ListItem>Kvinna</asp:ListItem>
              </asp:RadioButtonList>
            </td>
            <td>Radioknappslista</td>
          </tr>
          <tr>
            <td> </td>
            <td><asp:Button ID="btnSkicka" runat="server" Text="Skicka"
                onclick="btnSkicka_Click" /></td>
            <td>Kommandoknapp</td>
          </tr>
        </table>
      </asp:Panel>
      <!-- Panel med resultat -->
      <asp:Panel ID="pnlResultat" runat="server" Visible="false">
        <p>Tack <asp:Label id="lblNamn" runat="server"></asp:Label> för din
          anmälan!</p>
        <p>Vi kommer skicka broschyrer till <asp:Label id="lblAdress"
          runat="server"></asp:Label></p>
        <p>Broschyrer kommer skickas på språket <asp:Label
          id="lblNationalitet" runat="server"></asp:Label></p>
        <p>Du är intresserad av resmålen: <asp:Label id="lblSemestermal"
          runat="server"></asp:Label></p>
        <p><asp:Label id="lblReklam" runat="server"></asp:Label></p>
        <p>Vi kommer skicka broschyrer för: <asp:Label id="lblDestination"
          runat="server"></asp:Label></p>
        <p>Du är en <asp:Label id="lblKon" runat="server"></asp:Label></p>
      </asp:Panel>
    </div>
  </form>
```

Och koden för knappens klickhändelse visas nedan.

```
  pnlForm.Visible = false;     //Dölj formulär
  pnlResultat.Visible = true; //Visa resultat

  //Hämta namn, adress och nationalitet
  lblNamn.Text = txtNamn.Text;
  lblAdress.Text = txtAdress.Text;
  lblNationalitet.Text = ddlNationalitet.SelectedValue;

  //Hämta semestermål - kan vara multipla
  foreach (ListItem li in lstSemestermal.Items)
    if (li.Selected)
      lblSemestermal.Text += " " + li.Value;

  //Hämta om reklam
  if (cbxReklam.Checked)
    lblReklam.Text = "Vi kommer skicka reklam till dig.";

  //Hämta destinationer - kan vara multipla
  foreach (ListItem li in cblDestinationer.Items)
    if (li.Selected)
      lblDestination.Text += " " + li.Value;

  //Hämta kön
  lblKon.Text = rblKon.SelectedValue;
```

1.2.5.1 Om större exempel

I exempel ovan har jag gjort en del val, vilka bl.a. utvecklas i detta avsnitt.

Som exempel visar så är det ganska enkelt att hämta data från textrutor, men även nedrullningsbara listrutor (och listrutor som endast tillåter ett val) och andra listor (så som radioknappslistor). Men kontroller som tillåter flera val (listrutor och kryssrutelistor) så måste vi alltså loopa över alla alternativ i listorna och kontrollera om markerade.

För bl.a. listrutor (och andra typer av listor) har ”koder” använts som värde (i motsats till text som visas för besökare). Detta har gjorts då dessa brukar vara mindre (kortare) än värdet som de motsvarar, vilket bl.a. innebär att mindre data skickas från formulär. Dessa ”koder” brukar även vara primärnycklar i de eventuella tabeller som data till listor hämtas från, d.v.s. är unika. ”Koderna” är även lättare och mer effektiva att jämföra med, om strängar, då de är korta.

Med ASP.NET så innebär det ingen besparing av bandbredd om man aktiverat *ViewState*, d.v.s. att formulär ska komma ihåg vilka val som besökare har gjort, då all data i formulär ändå skickas tillbaka till server. (Data till en lista hämtas första gången som besökare laddar webbsida och sen skickas denna data fram och tillbaka tills besökare fyllt i formulär korrekt.)

1.3 Validering av kontroller i formulär

Innan man använder data från ett formulär som besökare fyllt i så bör (måste?) man validera data i formulärets kontroller (vilket inte görs i förra exemplet). Detta avsnitt behandlar detta ämne.

Validering bör alltid ske på serversida, men för att ge snabbare återkoppling (feedback) till besökare så kan det även ske i besökarens webbläsare. I ASP.NET så kan man lägga till valideringskontroller som genererar kod för klientsidan (d.v.s. vi slipper skriva egen JavaScript-kod).

1.3.1 Grundläggande

I detta exempel används en textruta, en knapp och en literal samt en valideringskontroll av typen RequiredFieldValidator, vilken kontrollerar att kontroll är ifylld. Utöver de gemensamma attributen för alla webbkontroller (t.ex. `ID` och `runat`) så krävs minst två attribut till för valideringskontroller:

- `ControlToValidate` – kontroll som ska valideras (`txtTal` i detta exempel).
- `ErrorMessage` – meddelande som ska visas om validering misslyckades (d.v.s. om värde som valideringskontroll ska validera inte uppfyller önskade krav).

```
<p>Fyll i ett tal:
  <asp:TextBox ID="txtTal" runat="server"></asp:TextBox>
  <asp:Button ID="btnSkicka" runat="server" Text="Skicka"
    onclick="btnSkicka_Click" />
  <asp:RequiredFieldValidator ID="rfvTal" runat="server"
    ControlToValidate="txtTal"
    ErrorMessage="FEL: Du måste fylla i ett
      tal."></asp:RequiredFieldValidator></p>

  <p><asp:Literal ID="litText"
    runat="server"></asp:Literal></p>
```

När besökare klickar på knappen så kontrolleras först eventuellt värde i textrutan, och om validering av värdet är OK så skickas formulär och text från textruta kopieras till literal.

```
protected void btnSkicka_Click(object sender, EventArgs e)
{
  litText.Text = txtTal.Text;
}
```

1.3.2 Valideringskontroller i ASP.NET

I ASP.NET 3.5 finns nedanstående valideringskontroller och en förklaring till vad de kan validera.

- RequiredFieldValidator – att en kontroll (fält) måste vara ifylld (se exempel ovan).
- RangeValidator – att värde är inom ett visst intervall (t.ex. tal, sträng eller datum).
- RegularExpressionValidator – att värde uppfyller ett s.k. *regular expression*.[26]
- CompareValidator – att värden i t.ex. två kontroller är de samma (användbart för att t.ex. verifiera att lösenord fyllts i korrekt).
- CustomValidator – ger möjlighet att validera med egna funktioner på både klient (med JavaScript) och/eller server.
- ValidationSummary – utför inge validering utan används för att summera eventuella felmeddelande från valideringskontroller ovan.
- DynamicValidator – fångar undantag (eng. *exceptions*) vid datamanipulatoin

[26] *Regular expressions* är ett komplext ämne och behandlas inte i denna sammafattning. Men lite sökande på Internet bör ge färdiga ”uttryck” som kan användas direkt, eventuellt med smärre justeringar.

2 Använda databaser i ASP.NET

Här finns bl.a. ”enkla” exempel för att visa kontroller så som GridView[27] och ListBox kan användas för att presentera data i databser (för fler exempel, se sammanfattningen *Webbsidor och databaser*). Beskrivningar i detta kapitel bygger på användande av *Visual Studio (.NET) 2008* (*Visual Web Developer*), men här presenteras även kod som genereras av VS.NET.[28]

2.1 Kontroller och databaser

I detta avsnitt beskrivs först hur enkelt det är att presentera data från en tabell i en webbsida med kontrollen GridView för att sen titta på hur man kan fylla en listruta med data från en tabell. I.o.m. version 2 av ASP.NET så blev det mycket lättare att hämta och presentera data från databaser.

2.1.1 Kontrollen GridView (och SqlDataSource)

Kontrollen GridView är en dynamisk kontroll vars storlek kan anpassas efter hur mycket data som en databastabell (eller frågeresultat, vektor, XML-fil/-sträng, m.m.) innehåller. Resultatet, efter att webbserver exekverat ASP.NET-koden, är en HTML-tabell.

Det finns ett antal olika sätt att använda en GridView, d.v.s. det jag beskriver nedan är bara **ett sätt**. Här beskrivs hur man lägger till kontrollerna en i taget samt konfigurerar dem i VS.NET. Kontrollernas funktion och utseende anpassas sen steg för steg.

Exempel nedan består av två filer:

- ASPX-filen `EnkelTabell.aspx`, kallat formuläret nedan, som innehåller HTML- och ASP.NET-koden.
- *code behind*-filen `EnkelTabell.aspx.vb`, kallat kodfilen nedan, som innehåller koden för logik.

2.1.1.1 Tabell i exempel

Detta exempel använder en tabell med namnet `Studenter` som innehåller två kolumner (se bilder nedan för exempel på innehåll):

- `personnr` (`NCHAR(10)`) – t.ex. 7123456789.[29]
- `namn` (`NVARCHAR(50)`) – t.ex. Nils Nilsson.

2.1.1.2 Lägga till kontrollerna till webbsida

För att visa data från databaser krävs minst två kontroller:

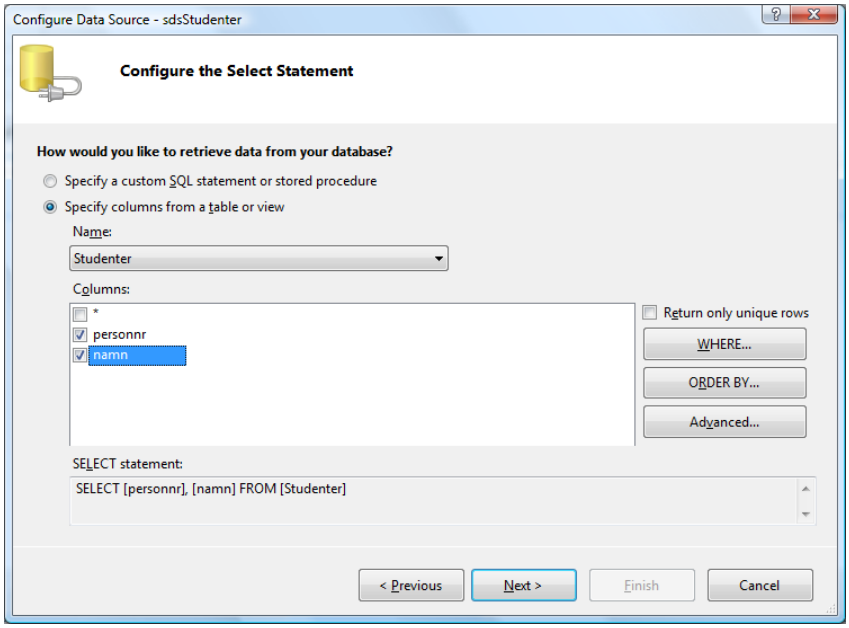
- En datakontroll – GridView i detta exempel.
- En datakälla – SqlDataSource i detta exempel då databasen är SQL Server Express som kan installeras med VS.NET.

När jag lägger till kontrollerna så brukar jag först lägga till datakontrollen (t.ex. GridView) och sist datakällan. Därmed påverkar datakällor webbsidans design så lite som möjligt under utveckling (samt, antagligen, av gammal vana från ASP.NET 1.x, där kontroller låg längst ner

[27]Kontrollen GridView ersätter kontrollen DataGrid som fanns i ASP.NET 1.x.
[28]Valet att bygga beskrivningar på VS.NET har gjorts då det är lättast ☺, men även då det numera går att ladda ner gratisversioner av programmet från Microsofts hemsidor.
[29] Ett (svenskt) personnummer är alltid 10 siffror, därav valet att använda datatypen `NCHAR(10)`.

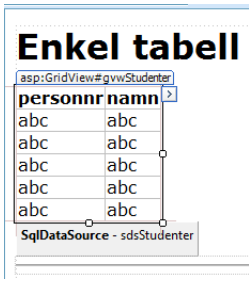
i designfönster). Beskrivning nedan kan, initialt, göras antingen i kod- eller designläge, men när vi ska börja konfigurera kontrollerna måste vi byta till designläge.

1. Dra och släpp en GridView på webbsidan. Döp om kontrollen (m.h.a. dess ID) till `gvwStudenter`.
2. Dra och släpp en SqlDataSource (eller en som passar din databasserver) på webbsidan. Döp om kontrollen till `sdsStudenter`.
3. Byt till designläge (om inte redan där) genom att klicka på fliken Design längst ner.
4. Markera datakällan och klicka på pilen som visas längst upp till höger på kontrollen. Välj *Configure Data Source...* från menyn som visas (”*Tasks*-menyn”). Detta startar en guide (eng. *wizard*).
5. Välj eventuell existerande datakälla, eller följ steg nedan för att skapa en ny datakälla.
   - Klicka på knappen *New Connection...*.
   - Välj typ av datakälla, t.ex. *Microsoft SQL Server*, och klicka på *Continue*.
   - Välj server, fyll i inloggningsinformation och välj databas samt klicka på OK.
6. Med datakälla vald, klicka på Next>.
7. Välj alternativet Specify columns from table och bocka för önskade kolumner (se bild till höger). Klicka på Next>.

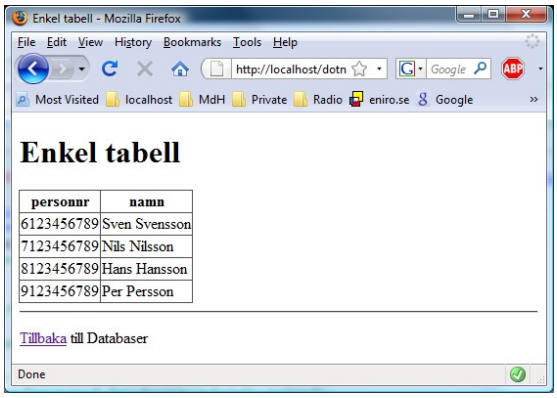


När vi väljer kolumner bör vi alltid ange respektive kolumn, d.v.s. undvik att använda *-teknet.

8. Testa eventuellt ”frågan” och klicka sen på Finish.
9. Markera GridView och välj nyss konfigurerade datakällan (sdsStudenter) i listrutan *Choose Data Source*. Detta anpassar GridView som visar namnet på valda kolumner i datakälla ovan som rubriker (se bild till höger). (I detta exempel så ändrar GridView från de tre kolumner, som är standard, till två.)
10. Kompilera och visa webbsida i webbläsare (t.ex. genom att högerklicka i webbsida och välja *View in Browser* från menyn som visas).



Resultatet bör se ut något i stil med det i bild nedan.



2.1.1.3 Genererad ASP.NET-kod

Stegen ovan genererar kod i stil med den nedan. Om kontrollerna lades till i kodläge så syns en del skillnader, bl.a. följande extra attribut i GridView:

- AutoGenerateColumns – om kolumner ska skapas automatiskt (eller om kolumner i taggen Columns ska användas – se mer nedan).
- DataKeyNames – namnet på kolumn(-er) med primärnyckel.
- DataSourceID – namnet på datakällans kontroll.

I GridView så finns två nya taggar:

- Columns – vilka kolumner som ska visas (se nästa punkt).
- BoundField – kolumn som är bunden till datakälla. Dess attribut är relativt självförklarande (primärnyckelns kolumn har dock ett extra attribut ReadOnly – mer om detta när vi tittar på att redigera poster nedan).

Även datakällans har fått extra attribut:

- ConnectionString – talar om vilken anslutningssträng som ska användas (d.v.s. hur anslutning till datakälla ska ske) – här betyder det att den finns i filen web.config och har namnet BjornPConnectionString1 (ofta databas – BjornP i exempel – och suffixet ConnectionString).
- SelectCommand – SQL-fråga för vad som ska hämtas. :-)

```
<asp:GridView ID="gvwStudenter" runat="server" AutoGenerateColumns="False"
  DataKeyNames="personnr" DataSourceID="sdsStudenter">
  <Columns>
    <asp:BoundField DataField="personnr" HeaderText="personnr"
      ReadOnly="True" SortExpression="personnr" />
    <asp:BoundField DataField="namn" HeaderText="namn"
      SortExpression="namn" />
  </Columns>
</asp:GridView>

<asp:SqlDataSource ID="sdsStudenter" runat="server"
  ConnectionString="<%$ ConnectionStrings:BjornPConnectionString1 %>"
  SelectCommand="SELECT [personnr], [namn] FROM [Studenter]">
</asp:SqlDataSource>
```

(En trevlig sak, jämfört med version 1.x av ASP.NET, är att vi inte längre behöver skriva någon kod för att binda data till webbkontrollen,något som var ganska förvirrande för nybörjare... d.v.s. även mig när jag började med ASP.NET och databaser.)

2.1.1.4 Anpassa tabells utseende

Tabellen i bild ovan är inte den mest tilltalande... Dags att anpassa den en aning.[30]

Först skapar vi en CSS-fil (t.ex. StyleSheet.css, eller öppnar existerande) och lägger till följande egenskaper/stilar. (Det som ändras är bakgrundsfärg, förgrunds-/textfärg, eventuell fet stil, fontstorlek resp. font.)

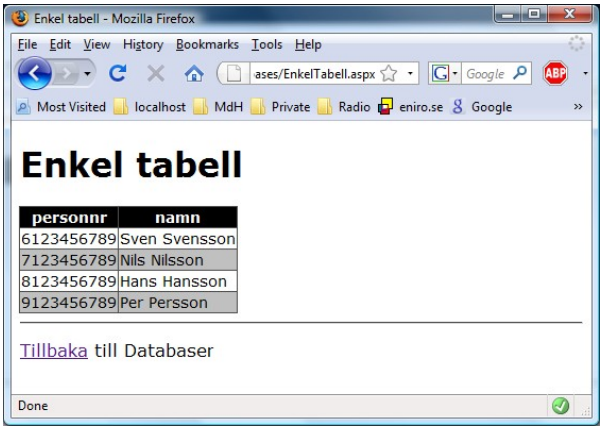
```
.listajamn
{
  background-color: #c0c0c0; color: black;
  font-size: 14px; font-family: Verdana
}
.listarubrik
{
  background-color: black; color: white;
  font-weight: bold; font-size: 14px; font-family: Verdana
}
.listaudda
{
  background-color: white; color: black;
  font-size: 14px; font-family: Verdana;
}
```

CSS-filen länkas sen in i vår webbsida genom att dra den från *Solution Explorer* och släppa på webbsidan i designläge (eller med nedanstående rad i sidans HEAD-tagg – anpassa eventuell sökväg och/eller filnamn i attributet `href`) .

```
<link href="StyleSheet.css" rel="stylesheet" type="text/css" />
```

Sen anger vi ett värde på CssClass för egenskaperna HeaderStyle, RowStyle och AlternatingRowStyle i egenskapsfönstret (*Properties*) för GridView: listarubrik, listaudda resp. listajamn.

Resultatet av applicerade stilmallar visas i bild till höger.



Om vi tittar på koden som genererats för GridView så ser den ut något i stil med den nedan, d.v.s. ett antal taggar har lagts till för stilarna.

```
<asp:GridView ID="gvwStudenter" runat="server" AutoGenerateColumns="False"
    DataKeyNames="personnr" DataSourceID="sdsStudenter">
  <RowStyle CssClass="listaudda" />
  <HeaderStyle CssClass="listarubrik" />
  <AlternatingRowStyle CssClass="listajamn" />
  <Columns>
    <asp:BoundField DataField="personnr" HeaderText="personnr"
      ReadOnly="True" SortExpression="personnr" />
    <asp:BoundField DataField="namn" HeaderText="namn"
      SortExpression="namn" />
  </Columns>
</asp:GridView>
```

[30] Med snygga till menar jag göra den mer lättläst/tilltalande – design är inte min starka sida. :-)

2.1.1.5 Redigera poster

Om vi vill kunna redigera poster finns ett antal olika alternativ, ett är t.ex. att kunna redigera posterna direkt i GridView.[31] (Och med risk för att vara tjatig: detta är mycket enklar fr.o.m. version 2 av ASP.NET då vi slipper skriva kod för att göra det numera. :-)) För detta ska fungera måste vi göra två saker:

- anpassa datakälla och lägga till UPDATE-sats (samt INSERT- och DELETE-satser när vi ändå är i farten :-)).
- lägga till en knappkolumn (*button column*), som antingen kan vara knappar eller länkar.

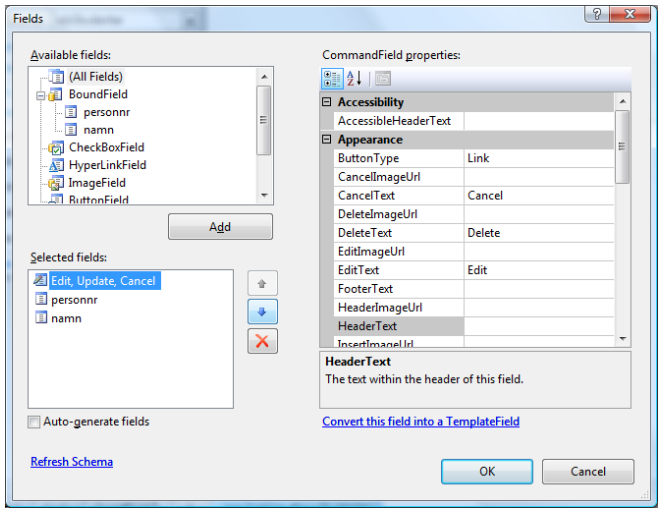
Vi kan även välja vilka kolumner i tabell som går att redigera och inte – de senare markeras som endast läsbara (*read only*), lämpligen primärnycklar (som skedde automatiskt i exempel ovan).

Nedan anpassas först datakälla och seden GridView.

1. Öppna webbsida och markera datakällan (sdsStudenter om exempel följs).
2. Klicka på pilen uppe till höger på datakälla för att visa *Tasks*-menyn.
3. Välj *Configure Data Source...* från menyn som visas.
4. Klicka på *Next>*. (Vi behöver inte byta datakälla. :-))
5. I steget *Configure the Select Statement*, klicka på knappen *Advanced...*.
6. Bocka för alternativet *Generate INSERT, ...* och klicka på OK.
7. Klicka på *Next>*.
8. Klicka på *Finish.*

Vi ser ingen skillnad i webbsidan (om vi inte tittar på koden – se mer nedan).

9. Markera GridView och visa dess *Tasks*-meny (pilen uppe till höger).
10. Bocka för alternativet *Enable Editing*. En kolumn med texten *Edit* visas längst till vänster i GridView.
11. Välj *Edit Columns...* från Tasks-menyn. Detta visar dialogrutan *Fields* (se bild till höger).
12. Markera listalternativet *Edit, Update, Cancel* i listrutan *Selected fields* och klicka på pilknappen som pekar neråt (till höger) två gånger för att flytta ner alternativet sist. Därmed kommer ”*Edit*-kolumnen” att placeras längst till höger i GridView.

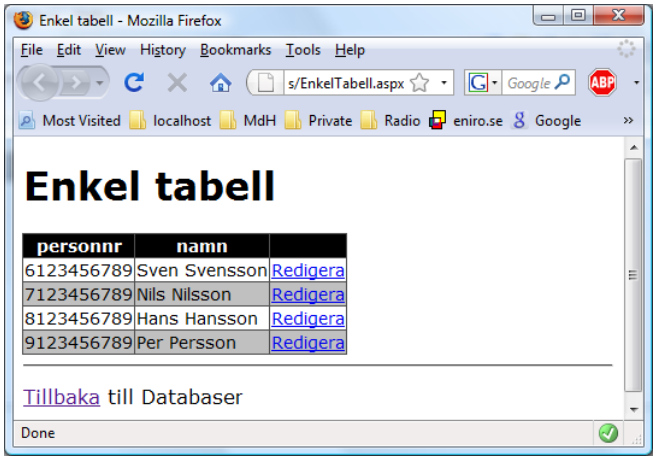


(För att ändra t.ex. rubriker i GridView så kan man markera kolumnen i listrutan *Selected fields* samt ändra dess egenskaper i listrutan *CommandField properties*. Det är också här som man kan ange om ett fält ska vara skrivskyddat, d.v.s. *read only*.)

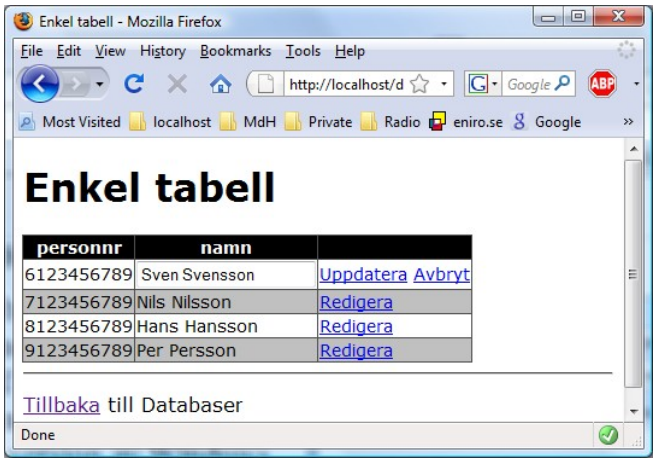
[31] Observera att detta inte alltid är det mest praktiska ur användarsynvinkel, även om det är det för dig som utvecklare!

13. Klicka på OK för att stänga dialogrutan.

Om vi tittar på webbsidan i en webbläsare så har den ett utseende i stil med den i bild till höger. I VS.NET så stod det Edit i länkar i Edit-kolumnen men om vi tittar på webbsidan i svensk version av Windows så står det Redigera.



När användare klickar på Redigera-länken (*edit*) så visas studentens namn i en textruta samt Ändra-länken ändras till Uppdatera- och Avbryt-länkar (*update* resp. *cancel*) (se bild nedan till höger; det syns främst då vald students första namn visas med annat typsnitt än övriga).



2.1.1.6 Genererad kod

Om vi byter till kodläge (genom att klicka på fliken *Source* längst ner i kodfönstret) så ser vi att vår datakälla har fått ett antal nya attribut:

- DeleteCommand – innehåller SQL-sats för att radera en post.
- InsertCommand – innehåller SQL-sats för att lägga till en post.
- UpdateCommand – innehåller SQL-sats för att uppdatera en post.

Datakällan har även fått ett antal taggar för parametrar till de nya SQL-satsern, som i sin har taggar med själva parametrarna.

```
<asp:SqlDataSource ID="sdsStudenter" runat="server"
  ConnectionString="<%$ ConnectionStrings:BjornPConnectionString1 %>"
  SelectCommand="SELECT [personnr], [namn] FROM [Studenter]"
  DeleteCommand="DELETE FROM [Studenter] WHERE [personnr] = @personnr"
  InsertCommand=
   "INSERT INTO [Studenter] ([personnr], [namn]) VALUES (@personnr, @namn)"
  UpdateCommand=
   "UPDATE [Studenter] SET [namn] = @namn WHERE [personnr] = @personnr">
  <DeleteParameters>
    <asp:Parameter Name="personnr" Type="String" />
  </DeleteParameters>
  <UpdateParameters>
    <asp:Parameter Name="namn" Type="String" />
    <asp:Parameter Name="personnr" Type="String" />
  </UpdateParameters>
  <InsertParameters>
    <asp:Parameter Name="personnr" Type="String" />
    <asp:Parameter Name="namn" Type="String" />
  </InsertParameters>
</asp:SqlDataSource>
```

Tittar vi på koden för GridView så är det främst en tagg som lagts till (om inte fler saker ändrades ovan :-)):

- CommandField – anger vilka kommandokolumner som ska visas, endast *Edit*-kolumn i detta exempel.

```
<asp:GridView ID="gvwStudenter" runat="server" AutoGenerateColumns="False"
  DataKeyNames="personnr" DataSourceID="sdsStudenter">
  <RowStyle CssClass="listaudda" />
  <HeaderStyle CssClass="listarubrik" />
  <AlternatingRowStyle CssClass="listajamn" />
  <Columns>
    <asp:BoundField DataField="personnr" HeaderText="personnr"
      ReadOnly="True" SortExpression="personnr" />
    <asp:BoundField DataField="namn" HeaderText="namn"
      SortExpression="namn" />
    <asp:CommandField ShowEditButton="True" />
  </Columns>
</asp:GridView>
```

2.1.1.7 Till sist...

Ovanstående exempel fungerar alldeles strålande för de flesta syften, men när vi behöver presentera data från flera t.ex. tabeller så börjar det bli mer komplext. (Se exempel/tutorials på Microsofts webbsidor för ASP.NET – adress finns sist i detta dokument.)

2.1.2 Kontrollen ListBox

Listrutor[32] (vanliga eller nedrullningsbara) består av ett värde, som ska skickas (t.ex. en primärnyckel), och en text, som ska visas för besökare (båda vilka kan vara samma). Vad vi vill hämta från en databas är alltså ett frågeresultat (SQL-sats) som består av endast (eller max) två kolumner: det som ska skickas och det som ska visas.[33] Vi måste, om värde och text ska vara olika, även tala om vilken kolumn som ska vara värde respektive text.

I detta exempel kommer vi använda en tabell innehållande kurser med fyra kolumner och när vi hämtar kurserna kommer vi sammanfoga tre av kolumnerna (de som inte är primärnyckel) för att visa för besökare. Primärnyckel kommer användas som värde att skicka.

Exempel nedan innehåller två filer:

- ASPX-filen `EnkelListruta.aspx`, kallat formuläret nedan, som innehåller HTML- och ASP.NET-koden.
- *code behind*-filen `EnkelListruta.aspx.cs`, kallat kodfilen nedan, som innehåller koden för logik.

2.1.2.1 Tabell i exempel

Detta exempel använder en tabell med namnet `kurser` som innehåller fyra kolumner:

- `kurskod` (`NCHAR(6)`) – t.ex. EI0700 eller EIK013.
- `kursnamn` (`NVARCHAR(50)`) – t.ex. Webbutveckling med ASP.NET.
- `niva` (`CHAR(1)`) – A, B, C eller D.
- `poang` (`SMALLINT`) – t.ex. 5 eller 10.

2.1.2.2 Lägga till kontrollerna till webbsida

För att visa data från databaser krävs minst två kontroller:

- En datakontroll – ListBox i detta exempel.

[32] Faktiskt gäller mycket beskrivet i detta avsnitt även för kryssrute- och radioknappslistor.
[33] Om vi sammanfogar flera kolumner i en tabell så bör vi namnge den med ett alias, d.v.s. att vi använder det reserverade ordet `AS` i SELECT-klausulen av SQL-sats.

- En datakälla – SqlDataSource i detta exempel då databasen är SQL Server Express som kan installeras med VS.NET.

När jag lägger till kontrollerna så brukar jag först lägga till datakontrollen (t.ex. ListBox) och sist datakällan (som med GridView ovan). Beskrivning nedan kan, initialt, göras antingen i kod- eller designläge, men när vi ska börja konfigurera kontrollerna måste vi byta till designläge.

1. Dra och släpp en ListBox på webbsidan. Döp om kontrollen (m.h.a. dess ID) till `lbxKurser`.
2. Dra och släpp en SqlDataSource (eller en som passar din databasserver) på webbsidan. Döp om kontrollen till `sdsKurser`.
3. Byt till designläge (om inte redan där) genom att klicka på fliken *Design* längst ner.
4. Markera datakällan och klicka på pilen som visas längst upp till höger på kontrollen. Välj *Configure Data Source...* från menyn som visas (”*Tasks*-menyn”). Detta startar en guide (eng. *wizard*).
5. Välj eventuell existerande datakälla, eller följ steg i förra avsnittet för att skapa en ny datakälla, och klicka på *Next>*.
6. Välj alternativet *Specify a custom SQL statement or stored procedure* och klicka på *Next>*.
7. Fyll i SQL-sats (se kodruta nedan), eller klicka på knappen Query Builder... för att bygga SQL-satsen ”visuellt”.
   - Markera tabellen (Kurser i exempel) i dialogrutan *Add Table* (nummer 2!), klicka på *Add* och sen *Close* för att återvända till dialogrutan *Query Builder* (nummer 1).
   - Bocka för alla kolumner som ska vara med (alla fyra i exempel).
   - Anpassa eventuellt SQL-sats i textrutan nedanför tabellen med kolumnerna. I detta exempel vill vi att det ska vara följande:[34]

```
SELECT kurskod, kursnamn + ', ' + niva + ', ' + CAST(poang AS NVARCHAR)
    + ' poäng' AS Kurs
FROM Kurser
```

   - Klicka på OK för att stänga dialogrutan.
8. Klicka på *Next>*.
9. Klicka eventuellt på *Test Query* för att testa frågan, d.v.s. om den fungerar. :-) Klicka sen på *Finish*.

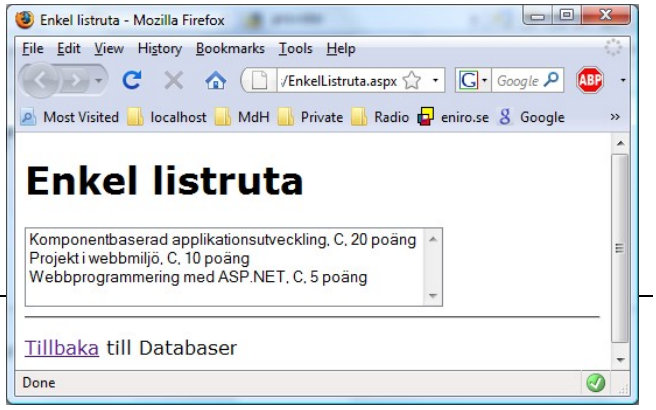
Återigen kan vi inte se någon skillnad i webbsidan (annat än i koden).

10. Markera listrutan och öppna dess Tasks-meny.
11. Välj Choose Data Source... från menyn som visas.

[34] Koden `CAST(poang AS NVARCHAR)` används för att konvertera värdet i `poang`, som är ett heltal, i T-SQL (d.v.s. SQL Servers SQL-dialekt).

12. Välj datakälla i första listrutan (sdsKurser i exempel), värde som ska visas (Kurs i exempel) i andra och värde som ska skickas (kurskod i exempel) i sista. Klicka på OK för att stänga dialogrutan.

Öppna webbsidan i en webbläsare och en sida i stil med den i bild till höger bör visas (givetvis beroende på vilka kurser som finns i tabellen Kurser :-)). (Exempel visar på att antagligen skulle kurskod även varit med i frågans andra kolumn...)



2.2 Använda egna klasser från ASP.NET [ATT GÖRA]

Ibland är det praktiskt att skapa klasser som vi kan återanvända i andra projekt. En sådan klass är SQLProxy som används för databasåtkomst.[35]

2.2.1 Klassen SQLProxy

Klassen SQLProxy innehåller nedanstående metoder.

- `GetReader(String)` – returnerar en DataReader för SQL-satsen i argumentet.
- `GetDataSet(String)` – returnerar ett DataSet för SQL-satsen i argumentet.

Det finns lite olika sätt att skapa klassen. Här kommer jag använda ett projekt – ett projekt som vi kan lägga till i lösningar där vi vill använda den.

[35] Klassen SQLProxy är en ”standardklass”, d.v.s. inget jag kommit på själv. ☺

3 Användarkontroller (*user controls*) [ATT GÖRA]

3.1.1 Skapa en enkel användarkontroll [ATT GÖRA]

3.1.2 Skapa en mer avancerad användarkontroll [ATT GÖRA]

3.1.3 Infoga en användarkontroll med egenskaper [ATT GÖRA]

När vi infogar en användarkontroll i ett webbformulär så skapas inte en motsvarande instansvariabel i code behind-filen. Vi måste alltså själva deklarera en variabel i code behind-filen för att kunna manipulera kontrollen med kod.

4 Inloggning och privata sidor

I version 2.0 av ASP.NET så introducerades ramverken *membership* och *roles* för att hantera identiteter resp. roller för inloggning, d.v.s. vi behöver inte längre skriva logik för att hantera inloggning. (Vad det egentligen är är ett antal gränssnitt (API[36]), d.v.s. i teorin så skulle vi kunna använda vilken databas som helst som det finns implementationer av dessa gränssnitt för.)

I detta kapitel beskrivs först hur man konfigurerar för ramverken *membership* och *roles* samt sen hur man använder sig av dessa funktioner.

4.1 Inledning

I sin enklaste form så är det bara att aktivera ramverket *membership*, m.h.a. *ASP.NET Configuration* i VS.NET, och börja köra (om man utvecklar på sin egen dator med SQL Server Express på). Detta skapar en SQL Server-databasfil med namnet `ASPNETDB.MDF` som automatiskt kopplas till SQL Server Express på lokal dator. (Om vi vill använda denna fil på en ”produktionsserver” så måste vi själva koppla denna databasfil till en SQL Server-instans!)

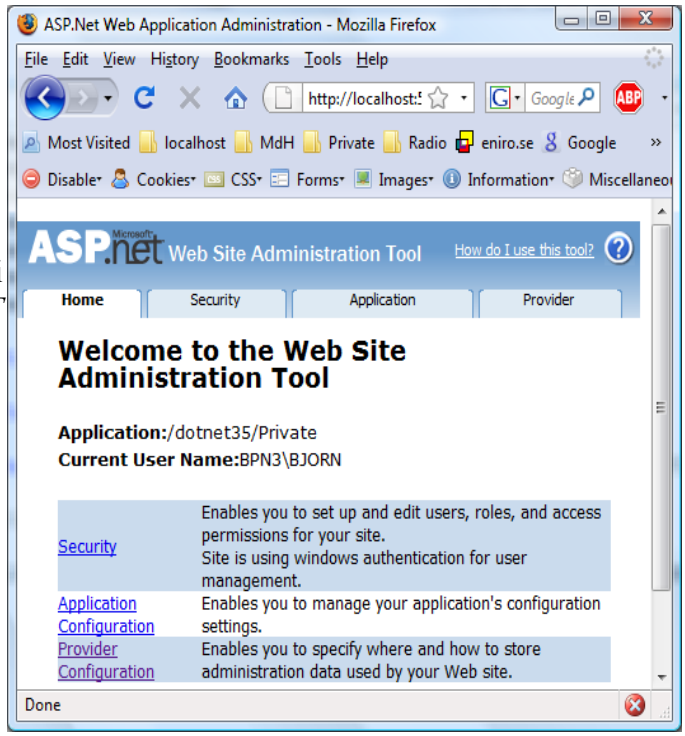
4.2 Konfigurera ramverken *membership* och *roles*

Innan vi kan börja använda ramverken *membership* eller *roles* så måste, som sagt, en del tabeller skapas, vilket lättast görs med en guide enligt instruktioner nedan. Först beskrivs hur vi använder standardinställningar genom att skapa filen `ASPNETDB.MDF` och sedan hur vi använder SQL Server, Express eller fullständig version. Man gör alltså **antingen eller**!

4.2.1 Skapa och använda ASP.NET-tabeller i ASPNETDB.MDF

Observera att detta främst ett alternativ om vi utvecklar lokalt! D.v.s. om vi har IIS och SQL Server Express på utvecklingsdator.

1. Starta VS.NET och skapa ett nytt projekt av typen *Web Site* (*File→New→Web Site...*, eller anslut till existerande webbplats).
2. Markera projekt i *Solution Explorer* (eller på någon gren/fil i projektet) och klicka på *ASP.NET Configuration*-knappen (hammare och jordglob, eller välj från menyn *Website*). Detta startar en webbläsare med gränssnittet för konfiguration av webbplats (se bild till höger).[37]
3. Klicka på länken *Provider Configuration*.
4. Klicka på länken *Select a single provider...*.



[36] *Application Programming Interface*.
[37] Observera att gränssnittet endast är aktiv under en viss tid, d.v.s. om du får en ”timeout” så stäng webbläsare och starta gränssnittet igen.

5. Klicka på länken *Test* och vänta på ett positivt resultat.
6. Klicka på OK om test gick bra.
7. Klicka på fliken *Security*.
8. Under rubriken *Users*, klicka på länken *Select authentication type*.
9. Markera alternativet *From Internet*. Detta gör att vi själva kan hantera användare och roller (istället för via Windows). Klicka på *Done* längst ner på sidan.
   (Om det uppstår fel här, lägg till en ny ASPX-sida i projekt, lägg till en SqlDataSource och konfigurera den – under konfiguration, spara ConnectionString. Prova sen att utföra dessa steg igen... Fråga mig inte varför – jag hänvisar till Bill Gates...:-))
10. På sidan (fliken) *Security* så visas nu att det finns 0 existerande användare och du kan börja lägga till dem du vill ha.

Om vi även vill använda roller så gör man följande.

1. Starta *ASP.NET Configuration*.
2. Klicka på fliken *Security*.
3. Under *Roles*, klicka på länken *Enable roles*.
4. Lägg till de roller du vill ha (samt koppla roller till användare).

4.2.2 Skapa och använda ASP.NET-tabeller i SQL Server

Detta alternativ är mer passande om vi vill utveckla för en dedicerad webbserver.

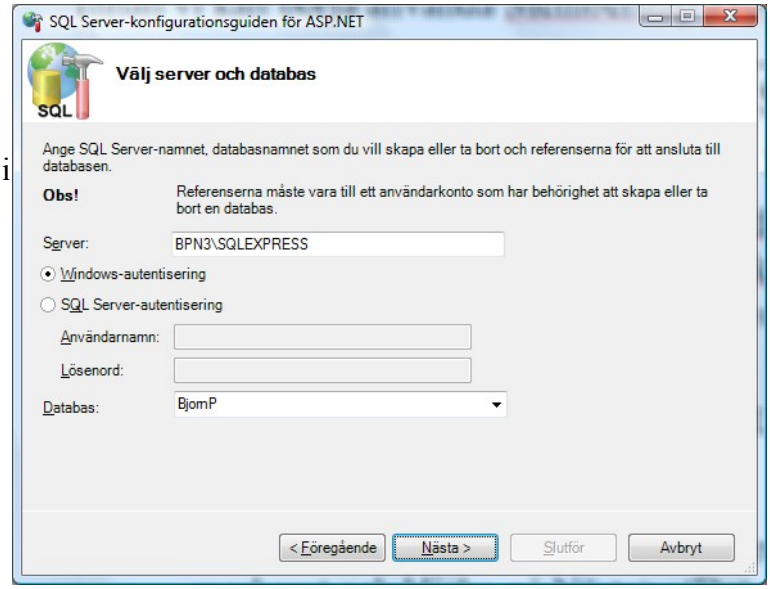
4.2.2.1 Skapa tabeller i databas

Nedan beskrivs hur man skapar tabeller i en SQL Server, Express eller fullständig version. I dessa exempel kommer jag att använda SQL Server Express, men det bör fungera på ett liknande sätt med fullständig version (med undantag för namn/adress för server, m.m.). Jag föredrar detta sätt att jobba, d.v.s. att jag skapar en databas i ”Express”, då det påminner mest om hur man gör med en fullständig version av SQL Server. Innan anvisningar nedan följs bör alltså en ny databas ha skapats i SQL Server (Express eller fullständig version).

1. Starta Visual Studio 200X Command Prompt (om den finns under Tools i din version av VS.NET) eller starta en ”vanlig” kommandotolk och byt till `C:\Windows\Microsoft.NET\Framework\v2.0.xxxxx` (ersätt `xxxxx` med din version).[38]
2. Skriv `aspnet_regsql` för att starta SQL Server-konfigurationsguiden för ASP.NET.
3. Klicka på Nästa> för att fortsätta.
4. Kontrollera att alternativet ”Konfigurera SQL Server för tillämpningstjänster” är markerat och klicka på Nästa>. (Det andra alternativet är om vi vill radera...)

[38] Ja, det ska vara version 2.0 även om vi använder version 3.5! Så många förändringarna har inte skett i den grundläggande strukturen av ASP.NET mellan version 2.0 och 3.5 – min förståelse (idag) är att 3.5 främst introducerar nya klasser och funktioner. (Min version av 2.0 är i skrivande stund v2.0.50727.)

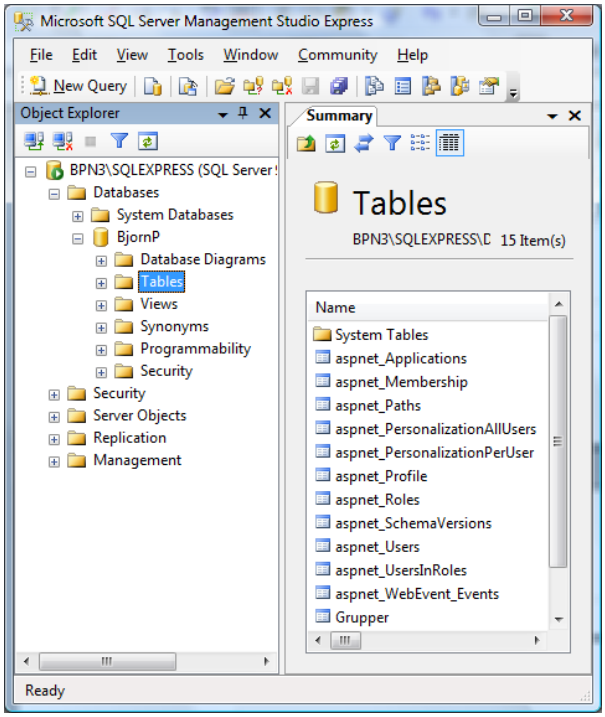
5. Fyll i server och eventuell instans. I mitt fall så använder jag SQL Server Express på min egen dator (BPN3), d.v.s. jag har fyllt i BPN3\SQLEXPRESS.
   Välj hur validering av användare i databas ska ske, d.v.s. via Windows eller ett konto i databasen. Eftersom SQL Express finns på min dator så har jag valt Windows.
   Välj sen databas som tabeller ska skapas i. I mitt fall har jag skapat en databas BjornP som jag valt.[39]
   Klicka sen på Nästa> för att fortsätta.



6. Klicka på Nästa> för att skapa tabellerna.

7. Klicka på Slutför för att avsluta guiden.

Om vi startar *MS SQL Server Management Studio Express* (SSMSE) så bör vi hitta ett antal tabeller som börjar med ”aspnet_” i databasen (se bild till höger).



4.2.2.2 Konfigurera webbapplikation att använda SQL Server-databas [ATT GÖRA]

Detta kan göras på olika sätt (och ser lite annorlunda ut beroende på version av .NET...), så jag brukar tycka om att ha inställningarna tillgängliga elektroniskt så att jag bara behöver markera, kopiera och klistra in (samt ev. anpassa namn på saker och ting, t.ex. namn på ConnectionString).

4.3 Ramverket membership och webbkontroller

I verktygsmenyn (*Tools*) i VS.NET finns en ”mapp” Login med kontroller som använder ramverket *membership*. Dessa är:

- Login – loginformulär som innehåller textrutor, knapp och annat för inloggning.
- LoginView – kontroll som kan visa olika saker, från mallar (templates), beroende på vem som är inloggad eller grupp som inloggad tillhör.
- PasswordRecovery -
- LoginStatus -
- LoginName – visar inloggad besökares inloggningsnamn.

[39] Om du inte kan välja en databas så har du antagligen fyllt i ett felaktigt användarnamn eller lösenord!

- CreateUserWizard – guide för att låta besökare skapa sin egen användaridentitet. Denna kontroll kan anpassas för att inkludera även egen data (i egna tabeller).
- ChangePassword – byt-lösenord-formulär.

4.3.1 Kontrollen Login

Denna kontroll behöver i stort sett var den enda kontrollen på inloggningssidan. :-)

5 AJAX (Asynchronous JavaScript and XML) [ATT GÖRA]

AJAX är en av modeorden inom webbutveckling, och jag får väl erkänna att den har sin nytta. :-) Microsoft släppte sin ”version 1” av AJAX som ett tillägg till ASP.NET 2.0, men i.o.m. version 3.5 (3.0?) av ASP.NET så är den integrerad i ASP.NET.

AJAX i .NET består av två delar:

- ett serverramverk – ett antal ASP.NET-kontroller
- ett klientramverk - ???

Utöver detta finns det även ett bibliotek med utökningar av Microsofts ”inbyggda” ASP.NET-kontroller som utvecklas m.h.a. communities: AJAX Toolkit.

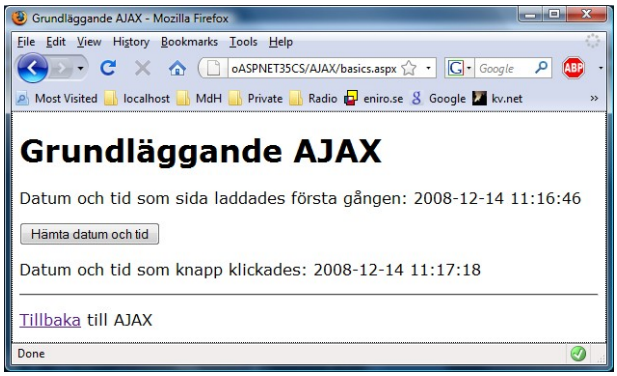
5.1 Grundläggande

För att AJAX ska fungera i ASP.NET behöver man lägga till två kontroller i webbsidan:

- ScriptManager - ????
- UpdatePanel – innehåller alla kontroller som ska kunna uppdateras asynkront.

5.1.1 Ett första exempel

Principen för webbsidan är att datum och tid som sida laddas visas samt en knapp som hämtar nytt datum och tid.



6 Tips, frågor och svar (FAQ) och annat som blev över...

I detta kapitel beskrivs lösningar på problem jag ”råkat ut för” och mina (eller av mig stulna ☺) lösningar.

6.1 Datum som fil senast ändrades (skrevs till)

I FrontPage finns en funktion för att infoga/uppdatera datum som fil senast ändrades. Jag har ännu inte hittat en motsvarande funktion i VS.NET (om den finns...). Men nedanstående lösning fungerar. ☺ Lösningen bygger på att en etikett med namnet `lblUppdaterad` används för att presentera datumet.

Här hämtas först URL till aktuell fil med funktionen `CurrentExecutionFilePath()`, som används i anropet av metoden `MapPath()` för att erhålla fysisk sökväg till fil i serverns filsystem. Sen testas (lite onödigt?) att fil existerar (och för att visa hur vi kan kontrollera en fils existens ☺) innan anrop av `GetLastWriteTime()` som returnerar datum som fil senast uppdaterades. Denna kod kan t.ex. placeras i metoden `Page_Load()`.

```
string strPath = Server.MapPath(Request.CurrentExecutionFilePath); //Fysisk
sökväg

if(File.Exists(strPath))
  lblUppdaterad.Text = File.GetLastWriteTime(strPath).ToString();
//Hämta datum
```

6.2 Använda Visual Studio .NET

Beskrivningar i denna sammanfattning har gjorts utifrån kod skriven (och genererad ☺) i Visual Studio 2008 Professional (VS.NET 2K8, eller bara VS.NET). Här har jag samlat lite tips om hur vi använder VS.NET och hur vi kan låta VS.NET hjälpa oss göra utveckling lättare. Använd möjligheten att kunna placera flera projekt i samma lösning, vilket gör det lättare att dela på filer, eller kopiera filer, mellan projekt samt felsöka webbapplikationer som utvecklas i flera projekt (t.ex. komponentbaserade applikationer).

6.2.1 Öppna ett existerande projekt

När vi skapar ett nytt webbprojekt så skapas en webbapplikation på webbserver. En webbapplikation är en mapp, alla filer i mappen samt alla undermappar (som inte tillhör en annan webbapplikation). Vi kan därför inte skapa ett nytt projekt för att redigera ett existerande webbprojekt (och därmed webbapplikation).

Istället, om vi inte redan har ett existerande lösning, så skapas en tom lösning som vi lägger till ett existerande projekt från webb i, vilket beskrivs nedan.

1. Högerklicka på lösningen i lösningsfönstret (*Solution Explorer*) och välj Add… från menyn som visas samt Existing Project From Web… från undermenyn som visas.
2. Fyll i URL till webbserver (**inte** projekt, d.v.s. utelämna sökväg) i dialogrutan Add Existing Project From Web som visas och klicka på OK.
3. Bläddra till mapp med projekt, markera XXPROJ-filen (ersätt XX med språk, t.ex. CS för C# och VB för VB.NET) och klicka på knappen Open.

6.2.2 Webbprojekt med flera utvecklare [ATT GÖRA]

När vi är flera som jobbar på samma webbprojekt så uppstår problemet med samtidig användning av filer då alla kodfiler finns i delad mapp (t.ex. på en webbserver), och med användning avses redigering. Vi måste m.a.o. skydda (eller försöka skydda) utvecklare från att öppna och redigera samma fil samtidigt. En lösning är att sitta i samma rum… ☺ Men en bättre är att använda källkodskontroll (*source control*).

7 Adresser och litteratur

Nedan finns adresser till diverse verktyg samt litteratur och webbsidor som använts som referensmaterial.

7.1 Verktyg för nerladdning

- Visual Studio 2008 Express Edition-produkter
  http://www.microsoft.com/express/product/

7.2 Böcker

Idag (2008) finns nyare versioner av böckerna. :-) Men detta är böckerna jag hämtat mycket av min grundläggande kunskap om ASP.NET ifrån och sen ”lärt om mig” för senare versioner.

- Anderson, R. et al, *Professional ASP.NET*, Wrox Press, 2001.
- Barwell, F. et al, *Professional VB.NET*, Wrox Press, 2001.
- Goode, C. et al, *Beginning ASP.NET 1.0 with C#*, Wrox Press, 2002.
- Homer, A. & D. Sussman, *ASP.NET Distributed Data Applications*, Wrox Press, 2002.
- Robinson, Ed, et al, *Upgrading MS Visual Basic 6.0 to MS Visual Basic.NET*, Microsoft Press, 2002.
- Ullman, C. et al, *Beginning ASP.NET 1.1 with Visual C# .NET 2003*, Wiley Publishing, 2004.

samt webbsidor hos Microsoft (bl.a. MSDN) liksom webbplatserna asp.net och GotDotNet.com.

7.3 Webbsidor

Webbsidorna är i.a.f. en aning mer uppdaterade än några av böckerna ovan. :-)

7.3.1 Konfiguerar *membership* och *roles providers*

- *Configuring ASP.NET 2.0 Application Services to use SQL Server 2000 or SQL Server 2005*
  http://weblogs.asp.net/scottgu/archive/2005/08/25/423703.aspx
- *Creating the Membership Schema in SQL Server*
  http://www.asp.net/learn/security/tutorial-04-vb.aspx?wwwaspnetrdirset=1