

10. POGLAVLJE

ASP.NET

U ovom poglavlju:

- Kako pisati web-aplikacije
- Tipovi kontrola i njihovo korištenje
- Što su aplikacija i sesija
- Konfiguracija servera i aplikacija
- Cacheiranje izlaza, fragmenata i podataka
- Povezivanje s ADO.NET-om

Vjerujemo da o značaju i korisnosti web-aplikacija ne treba trošiti riječi. Štoviše, vjerujemo da će dio čitatelja knjige preskočiti neka (po njihovu sudu) manje zanimljiva poglavlja i odmah skočiti na igranje sa zlatnom kokom – na izradu web-aplikacija. Zato prije nego što zaronimo u detalje moramo zamoliti takve čitatelje da odustanu od te namjere i čitaju poglavlja redom. Naime, jedna od glavnih promjena koje je donio ASP.NET jest značajno približavanje programskom modelu koji smo upoznali u poglavlju o izradi prozorskih aplikacija, pa bi preskakanje istog moglo biti kobno.

Sukladno s općenitim stavom u knjizi, nećemo u detalje uspoređivati ASP.NET sa starijim inačicama ove tehnologije, no valja naglasiti da su se stvari značajno promijenile. Prvenstveno

III. DIO: DIJELOVI .NET-A

stoga što je ASP.NET objektno orijentiran i bolje strukturiran. Naravno, to ne znači da će vaše eventualno poznavanje ASP-a biti beskorisno, no morate imati na umu da će pristup pisanju web-aplikacija biti drastično izmijenjen.

Nostalgija za kôdom

Što učiniti s hrpom već postojećeg kôda i skriptama u ASP-u koje godinama uspješno rade? Prelazak na ASP.NET preporučuje se iz mnogo razloga, a na vrhu liste nalaze se performanse i stabilnost.

Iako je Microsoft pazio da “prevođenje” kôda bude što lakše, ipak se radi o prilično dugotrajnom i napornom poslu. Stoga je dobrodošla mogućnost suživota web-aplikacija pisanih u ASP-u i ASP.NET-u na istom računalu, pa čak i na istim web-stranicama. Doduše, oni neće moći dijeliti stvari poput aplikacijskih i sesijskih varijabli, niti na isti način pristupati bazama podataka, no mogućnost suživota olakšat će vam utoliko što ćete prebacivanje moći raditi po segmentima.

Evo nekoliko glavnih stvari na koje ćete morati paziti:

- Kôd i dalje može biti smješten između *tagova* `<% i %>`, ali u njemu ne mogu biti deklarirane varijable; također, taj će se kôd izvršavati prilikom renderiranja stranice, nakon ostalog kôda pisanog u datoteci s pozadinskim kôdom (o tome nešto kasnije).

- U ASP-u se moglo miješati jezike unutar iste stranice, dok kod ASP.NET-a cijela stranica mora biti u istom jeziku.

- Ukoliko ste u ASP-u koristili programski jezik Basic, koristili ste VBScript. U ASP.NET-u

koristi se Visual Basic .NET koji se razlikuje u sintaksi.

- U ASP-u su sve varijable bile tipa *variant*, što je značilo da mogu poprimiti vrijednost bilo kojeg tipa. U ASP.NET-u, kao i u cijelom .NET Frameworku, tip varijable mora biti strogo definiran (engl. *strongly typed*).

- ASP po *defaultu* proslijeđuje parametre kao reference, a ASP.NET kao vrijednosti.

- Parametri metoda moraju biti u zagradi, čak i ako nema povratne vrijednosti.

- Obavezno je deklariranje varijabli.

Naravno, ovo je tek vrh ledene sante. Detaljnije informacije možete naći na brojnim web-stranicama ili u MSDN-u, na putanji .NET Development > .NET Framework SDK > .NET Framework > Building Applications > Creating ASP.NET Web Applications > Migrating ASP Pages to ASP.NET ili .NET Development > ASP.NET > Technical Articles > Migrating to ASP.NET: Key Considerations.

Usput, postoji i alat za, uvjetno rečeno, automatsku konverziju stranica iz ASP-a u ASP.NET. Više informacija o njemu, kao i neke druge korisne savjete, možete potražiti na adresi <http://msdn.microsoft.com/asp.net/using/migrating>.

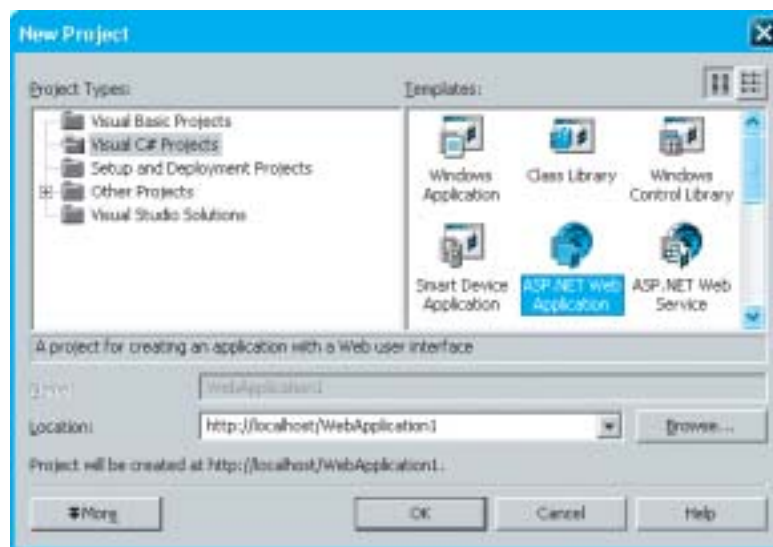
Kao i kod svih aplikacija pisanih u .NET-u, i stranice ASP.NET zahtijevaju postojanje .NET Frameworka na računalu na kojem se nalaze (posjetitelji tih stranica, naravno, ne moraju imati .NET Framework). Osim njega, zbog naravi samih web-stranica, u igri će morati biti i poslužitelj web-stranica, konkretno Internet Information Server (skraćeno IIS). On je sastavni dio gotovo svih Windowsa, a za posluživanje stranica tipa ASP.NET ne treba nikakvu posebnu konfiguraciju ili podešavanje – dovoljno je da nastavak datoteka bude .aspx i stranica će biti prepoznata kao stranica tipa ASP.NET.

Nastavak .aspx potječe od prvobitnog imena ASP.NET-a koji se tada zvao ASP Plus.



Forme i kontrole

Kao kod upoznavanja s prozorskim aplikacijama, i ovdje ćemo prolaziti kroz jednostavne primjere kako bismo upoznali mogućnosti i specifičnosti izrade web-aplikacija. Prvi korak u tome je stvaranje novog projekta.



Slika 10-1:
Stvaranje novog
projekta za
web-aplikaciju

U ovom poglavlju koristit ćemo predložak nazvan ASP.NET Web Application. Nakon što ga označite (vidi sliku 10-1), primijetit ćete da u polje lokacije ne upisujemo mapu na disku, nego mapu na lokalnom web-poslužitelju.

III. DIO: DIJELOVI .NET-A

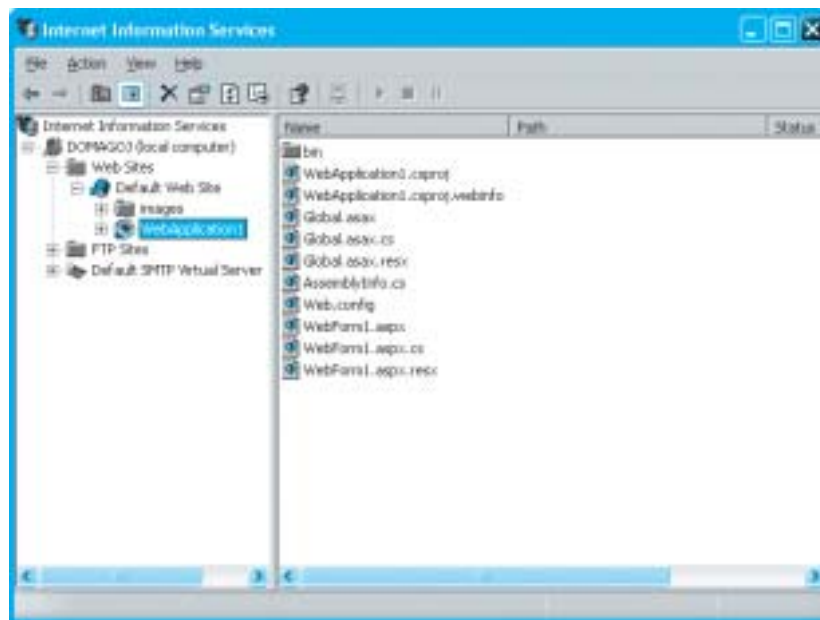
Naime, kako smo već spomenuli, svaka aplikacija zahtijeva postojanje web-poslužitelja koji je nužan za njezin razvoj i pokretanje. Logično bi bilo da svaka aplikacija bude na zasebnoj web-adresi (poput www.bug.hr), no kako većina klijentskih inačica Windowsa ima ograničenje na samo jedan web-site po poslužitelju, aplikacije se smještaju u zasebne mape i dobivaju adresu poput <http://localhost/ImeAplikacije/>. Osim toga, smještanje na posebnu osnovnu adresu uključivalo bi konfiguraciju IP-adresa i DNS-a, pa je ovaj pristup praktičniji i jednostavniji.



Adresa <http://localhost/> je adresa lokalnog računala. Ona upućuje na IP-adresu 127.0.0.1 koja predstavlja adresu lokalnog računala, koju nazivamo i *loopback*.

Nakon što kreirate novi projekt, Visual Studio će u pozadini napraviti nekoliko stvari. Prvo će kreirati mapu na disku. Ako niste dirali osnovnu konfiguraciju IIS-a, ta će se mapa nalaziti na putanji "C:\inetpub\wwwroot\ImeAplikacije". Na tu će mapu pokazivati već spomenuta virtualna mapa <http://localhost/ImeAplikacije/> koja će biti otvorena na web-poslužitelju i predstavljat će samostalnu web-aplikaciju sa svim karakteristikama, bez obzira na adresu.

Slika 10-2:
Novootvorena
aplikacija u mapi
označena je dru-
gačijom ikonom
od običnih mapa.



Bez obzira na to što za vrijeme razvoja web-aplikaciju smještamo u mapu, ona će kasnije moći normalno raditi na poslužitelju na osnovnoj adresi. Bit će je dovoljno kopirati.



Web-forma

Slično kao i kod izrade prozorskih aplikacija, i ovdje se koriste forme, no nešto izmijenjenih karakteristika. Dok smo kod prvih formu poistovjećivali s prozorom, web-forme možemo poistovjetiti s web-stranicom odnosno, kako se to često u sučelju Visual Studija naziva, web-dokumentom.

Web-aplikacije imaju jednu specifičnost – svaka je web-forma opisana s dvije datoteke. Prva datoteka sadrži sučelje opisano HTML-om i pojačano serverskim kontrolama (o tome nešto kasnije), dok u drugu datoteku dolazi kôd, funkcionalnost forme pisana u odabranom programskom jeziku.

To je princip koji nalaže Visual Studio, no nije jedini mogući. Naime, postoji mogućnost da oba dijela stranice stavite u istu datoteku, kao što to rade neki drugi razvojni alati, poput Web Matrixa. Međutim, odvajanje sučelja i kôda donosi puno prednosti, posebno na području preglednosti i čitljivosti. Takav način rada naziva se *code-behind*, dok se trpanje sučelja i kôda u istu datoteku zove *in-page coding*. Usporedbe radi, stari je ASP koristio ovaj drugi pristup što se često pokazivalo nepraktičnim.

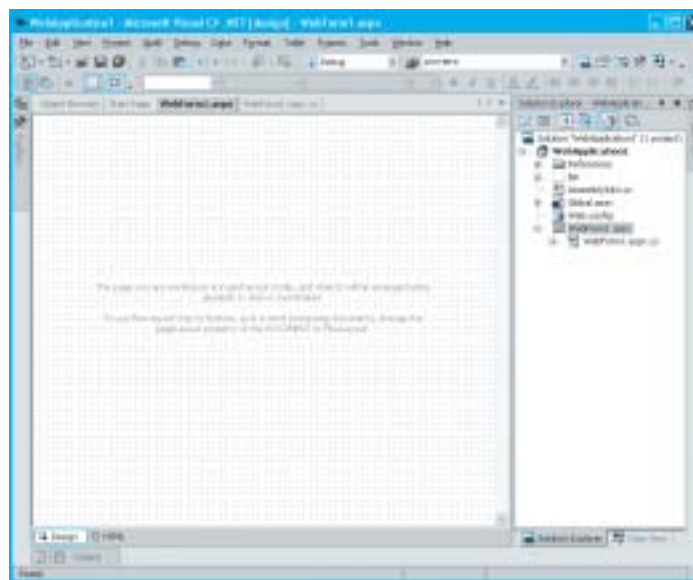
Kod pisanja aplikacija u načinu *code-behind* datoteka sučelja ima nastavak “.aspx”, dok datoteka s kôdom završava na “.aspx.cs”. Drugi nastavak ovisi o programskom jeziku u kojem radimo – da koristimo Visual Basic .NET, nastavak bi bio “.aspx.vb”.



Sukladno upravo rečenom, na stranicu u sučelju Visual Studija možemo gledati na tri načina. Prvi, dizajnerski način (slike 10-3 i 10-4), omogućava nam izradu sučelja dovlačenjem kontrola iz prozora Toolbox. On može raditi u dva načina – GridLayout i FlowLayout. Prvi nam omogućava slaganje kontrola po volji – svaka od njih bit će pozicionirana na stranici na apsolutnim koordinatama, baš kao što smo to radili s prozorskim aplikacijama. Međutim, webu je puno prirodniji način ovaj drugi, FlowLayout, koji ne dozvoljava apsolutno pozicioniranje kontrola, već ih na stranicu smješta po redu, jednu za drugom, dok se, želimo li razmake među njima, moramo snalaziti tablicama, razmacima i prelascima u novi red. Za to će vam pak biti potrebno poznavanje jezika HTML, u čije tajne nemamo namjeru ulaziti u ovoj knjizi. Za tu tematiku preporučujemo knjigu *Izrada weba – abeceda za webmastere*, izdanu u istoj biblioteci.

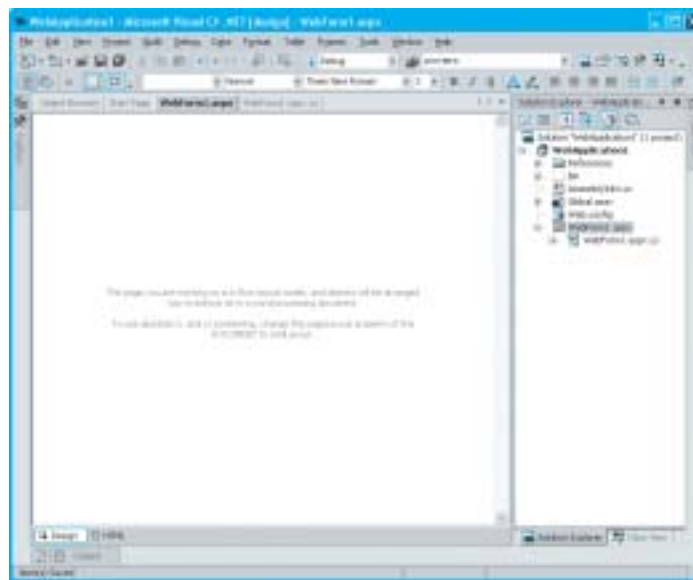
III. DIO: DIJELOVI .NET-A

Slika 10-3:
Pogled na
web-formu u
dizajnerskom
načinu –
GridLayout



Načini GridLayout i FlowLayout u dizajnerskom se načinu mogu razlikovati po točkicama – GridLayout u pozadini forme prikazuje mrežu točkica, dok je pozadina u načinu FlowLayout jednoboja.

Slika 10-4:
Pogled na
web-formu u
dizajnerskom
načinu –
FlowLayout



Iako vam se GridLayout vjerojatno čini privlačniji, aludiramo da ga ne koristite, posebno u projek-tima namijenjenima široj populaciji. Naime, on podrazumijeva neke stvari koje su na webu vrlo upitne, poput kompatibilnosti internetskih preglednika, fiksne dimenzije stranice i slično. Njih even-tualno možete koristiti za izradu internih web-stranica namijenjenih manjoj, kontroliranoj skupini ljudi.

I sami ćemo poslušati vlastiti savjet – u primjerima ćemo se baviti isključivo načinom FlowLayout, pa stoga odmah na početku svojstvo pageLayout objekta Document postavite na tu vrijednost.

ASP.NET pripada u serverske tehnologije, što znači da se izvršava na strani servera, dok klijent do-biva kôd pisan u HTML-u. Ipak, i na serverskoj strani glavnu ulogu ima HTML – on i dalje predstav-lja osnovu web-stranice i na njega se nadograđuje funkcionalnost u ASP.NET-u.

Svi u jednoga, jedan za sve

U knjizi pratimo rad Visual Studija i, su-kladno tome, koristimo dvije datoteke – HTML-stranicu i pozadinski kôd. Međutim, red je da pokažemo kako se i gdje pozadinski kôd treba staviti u slučajevima kada želimo da oba dijela budu u istoj datoteci:

```
<%@ Page Language="C#" %>
<%@ Import
    Namespace="System.Data" %>

<html>
    <script runat="server">

        // tu ide "pozadinski" kôd
```

```
void Page_Load(Object
    sender, EventArgs e)
{
    // ...
}
</script>

<body>
    <form runat="server">
        <!-- tu idu HTML i
            serverske kontrole -->
    </form>
</body>
</html>
```

Nekoć se to radilo na način da se kôd pisan u HTML-u kombinira sa skriptom, no ASP.NET, za-hvaljujući svojoj objektnoj orijentaciji, radi na drugi način. Trik je u tome da elemente HTML-a pretvaramo u kontrole te se njima pristupa kao i kontrolama prozorskih aplikacija.

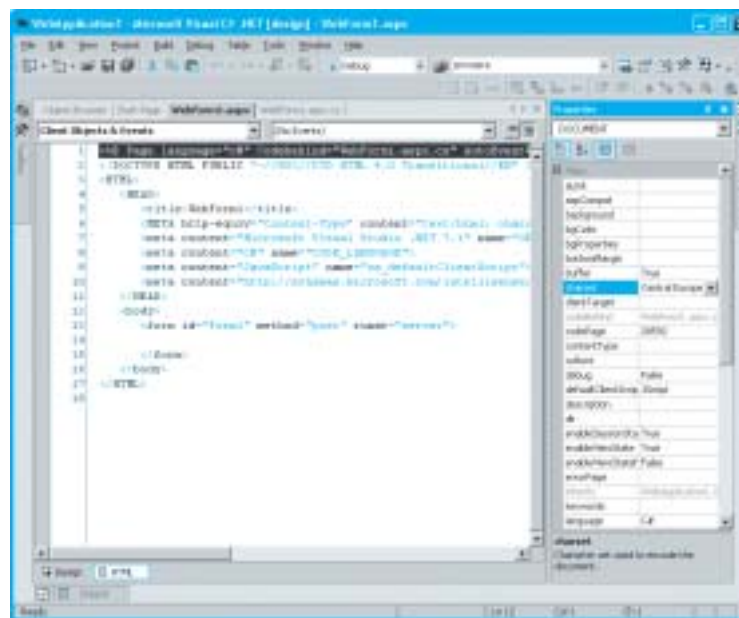
III. DIO: DIJELOVI .NET-A

To omogućava razdvajanje HTML-a i programskog kôda, kao i mogućnost vezanja uz događaje nad tim objektima, što u HTML-u inače ne postoji.

Osim što postojeće elemente HTML-a možemo pretvoriti u serverske kontrole, možemo koristiti i ASP.NET-ove web-kontrole koje nisu ništa drugo nego povezana skupina HTML-elemenata koji pružaju određenu funkcionalnost.

Naravno, ne moraju svi elementi HTML-a postati kontrole – oni koji se ne kreiraju dinamički i dalje će biti obični elementi HTML-a bez ikakvih dodira sa serverskim skriptiranjem.

Slika 10-5:
Pogled u HTML-kôd
prazne stranice pisane
u ASP.NET-u



Osim serverskih kontrola, web-stranica koja zasluhuje nastavak .aspx mora imati i tzv. *Page*-direktivu koju ćete u sučelju Visual Studija prepoznati po žutoj boji. U njoj se definiraju određeni parametri izvršavanja skripte. Njih, dakako, ne morate pisati ručno, već možete iskoristiti sučelje Visual Studija – spomenuti parametri prikazani su kao svojstva objekta Document. Osim tih parametara, preko svojstava tog objekta utječemo i na neke karakteristike samog dokumenta, njegova zaglavlja (*head*) i osnovnih postavki tijela (*body*). Na direktive ćemo se vratiti nešto kasnije.

Serverske kontrole HTML-a

Dovucimo na web-formu iz sekcije HTML prozora Toolbox kontrolu Label. Ako nakon toga pogledamo u HTML-kôd, uočićemo da je dodan otprilike sljedeći kôd:

10. POGLAVLJE: ASP.NET

```
<DIV style="DISPLAY: inline; WIDTH: 70px; HEIGHT: 15px"
ms_positioning="FlowLayout">Label</DIV>
```

Unatoč određenim stilskim atributima i nestandardnim atributima *taga* Div, radi se o običnom HTML-elementu. Da bismo od njega dobili serversku kontrolu, moramo u dizajnerskom načinu označiti element, kliknuti desnom tipkom miša i u padajućem izborniku izabrati "Run as Server Control". Tim postupkom element će dobiti dva nova parametra, a cijeli će redak izgledati ovako:

```
<DIV style="DISPLAY: inline; WIDTH: 70px; HEIGHT: 15px" ms_positioning="FlowLayout"
id="DIV1" runat="server">Label</DIV>
```

HTML-element nije moguće promovirati u kontrolu pukim dodavanjem spomenutih parametara. Opisana radnja promocije manifestira se na još jednom mjestu, što ćemo vidjeti nešto kasnije.



Bilo kako bilo, nakon promoviranja elementa u serversku kontrolu, njemu i njegovim svojstvima možete pristupiti iz pozadinskog kôda (koji se nalazi u drugoj datoteci). Pristupa mu se kao i svakom drugom objektu – navođenjem njegova imena. Primjerice ovako:

```
DIV1.InnerText = "Novi tekst za DIV1";
```

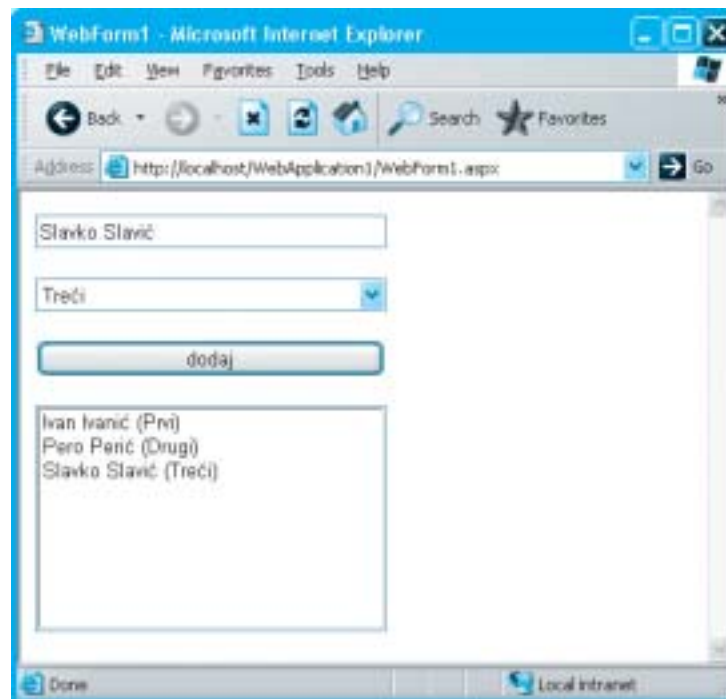
Slika 10-6:
Serverske kontrole u dizajnerskom načinu razlikuju se po sitnim trokutićima.

III. DIO: DIJELOVI .NET-A

Primjer: Beskorisna aplikacija

No kako ne bi sve ostalo na pustoj teoriji, krenimo s konkretnim primjerom. I ovoga ćemo se puta poslužiti idejom beskorisne aplikacije jer je to zgodan način za upoznavanje osnovnih funkcija i jer ćete uočiti paralele s razvojem prozorskih aplikacija.

Slika 10-7:
*Beskorisna aplikacija u
svom web-izdanju*

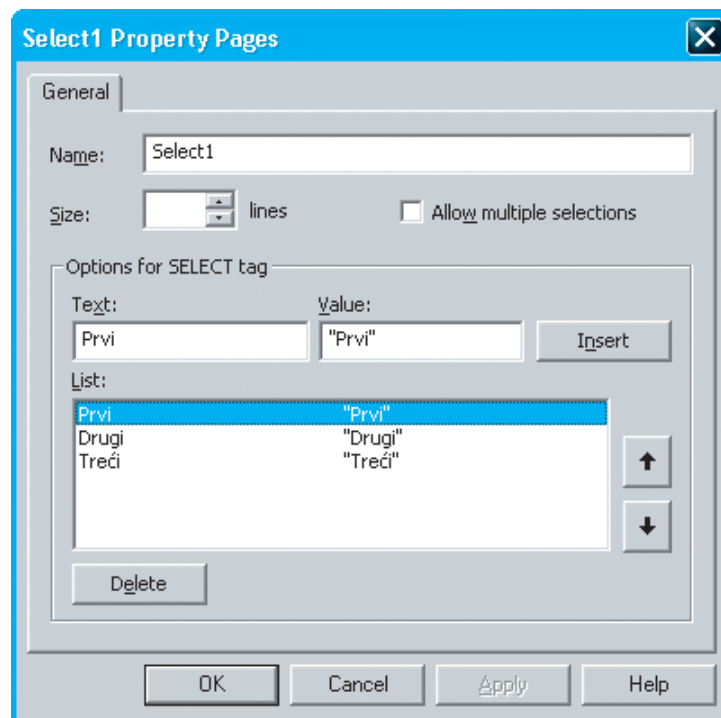


U tu ćemo svrhu na novootvorenu web-formu dovući potrebne elemente: po jedan Text Field, Drop-down, Button i Listbox. Sve ćemo promovirati u serverske kontrole i ostaviti im automatski dodeljena imena.

Zatim ćemo kliknuti desnom tipkom miša na kontrolu DrowDown kako bismo pomoću prozora koji se krije iza stavke Properties dodali stavke padajućeg izbornika.

Nakon promjena, u pogledu na HTML stvar će izgledati ovako:

```
<SELECT id="Select1" name="Select1" runat="server">
  <OPTION value="Prvi">Prvi</OPTION>
  <OPTION value="Drugi">Drugi</OPTION>
  <OPTION value="Treći">Treći</OPTION>
</SELECT>
```



Slika 10-8:
Pomoćni prozor za dodavanje stavaka padajućem izborniku

Uočite kako neke različite kontrole koriste isti HTML-element za prikaz na stranici.



Preostaje nam još isprogramirati funkcionalnost. Za one koji se ne sjećaju, cilj je da se pritiskom na gumb uzmu vrijednosti iz tekstualnog polja i padajuće liste i zapišu na listu koja se nalazi ispod gumba (vidi sliku 10-7).

Oni koji su se bavili razvojem web-aplikacija znaju koliko je to prije ASP.NET-a bilo složeno, no sada se to svodi na jednu liniju kôda!

Dvokliknite na gumb u dizajnerskom načinu i otvorit će se funkcija koja će biti pozvana kada gumb bude pritisnut. U nju ćemo ubaciti sljedeći kôd:

```
private void Button1_ServerClick(...)
{
    Select2.Items.Add(Text1.Value + " (" + Select1.Value + ")");
}
```

III. DIO: DIJELOVI .NET-A

Ukoliko vam se stvar čini poznata – u pravu ste! Postoje, doduše, određene razlike u imenima svojstava, no princip je u potpunosti jednak onome kod prozorskih aplikacija!

Proučimo što se događa u pozadini. Zavirimo li u HTML-kôd, možemo uočiti kako je cijelo tijelo stranice obuhvaćeno formularom – *tagovima* `<form>` i `</form>`. Taj formular nismo sami kreirali, već je on automatski ubačen prilikom kreiranja nove stranice. Kada kliknemo na gumb, sadržaj formulara šalje se serveru (kao i kod klasičnih web-stranica) i server vraća istu stranicu s izmijenjenim sadržajem. Cijela priča oko pamćenja stanja stranice obavlja se interno i programer na nju ne mora trošiti misli.

Ne treba posebno naglašavati da je stranica koja dolazi posjetitelju kao i svaka druga, s izuzetkom skrivenog parametra “__VIEWSTATE” koji pamti stanje stranice i njezine postavke. On je zaslužan za jednostavnost pisanja funkcionalnosti poput ove koju smo upravo pokazali. Pogledamo li iz drugog kuta, vidimo da Viewstate brine o vrijednostima serverskih kontrola odnosno da svaki element za koji želimo da Viewstate prati stanje moramo promovirati u serversku kontrolu.

Tablica 10-1:
*Lista serverskih kontrola
i popis elemenata koji se
u njih promoviraju*

HTML-element	Ime kontrole
HtmlAnchor	<code><a></code>
HtmlButton	<code><button></code>
HtmlForm	<code><form></code>
HtmlImage	<code></code>
HtmlInputButton	<code><input type="button"></code> , <code><input type="submit"></code> , <code><input type="reset"></code>
HtmlInputCheckBox	<code><input type="checkbox"></code>
HtmlInputFile	<code><input type="file"></code>
HtmlInputHidden	<code><input type="hidden"></code>
HtmlInputImage	<code><input type="image"></code>
HtmlInputRadioButton	<code><input type="radio"></code>
HtmlInputText	<code><input type="text"></code> , <code><input type="password"></code>
HtmlSelect	<code><select></code>
HtmlTable	<code><table></code>
HtmlTableCell	<code><td></code> , <code><th></code>
HtmlTableRow	<code><tr></code>
HtmlTextArea	<code><textarea></code>
HtmlGenericControl	Svi ostali elementi

Svaki HTML-element može biti promoviran u kontrolu, no pravo je pitanje – u koji tip kontrole će biti promoviran. Sve dostupne kontrole smještene su u klasi `System.Web.UI.HtmlControls` – za neke elemente postoje posebne kontrole, dok se ostali promoviraju u generičku kontrolu.

U koju se element kontrolu pretvorio možete vidjeti u pozadinskom kodu. Za svaku serversku kontrolu postoji jedan redak:

```
protected System.Web.UI.HtmlControls.HtmlInputText Text1;
protected System.Web.UI.HtmlControls.HtmlSelect Select1;
protected System.Web.UI.HtmlControls.HtmlInputButton Button1;
protected System.Web.UI.HtmlControls.HtmlSelect Select2;
```

Naime, kada promoviramo element u kontrolu, Visual Studio automatski dodaje i ovaj redak kako bismo mu mogli pristupiti iz pozadinskog kôda. Drugim riječima, želite li neki element pretvoriti u kontrolu ručno, osim dodavanja prije spomenutih atributa, morat ćete dodati i deklaraciju poput ovih iz primjera. U izboru adekvatnog tipa kontrole pomoći će vam tablica 10-1.

Serverske kontrole u pravilu trebaju biti unutar glavnog formulara (dakle, između tagova `<form>` i `</form>`). Međutim, u serversku kontrolu možete promovirati i elemente izvan njega (primjerice, naslov stranice sadržan između tagova `<title>` i `</title>`). Ipak, kako su te kontrole izvan obrasca, bit će zakinite za neke mogućnosti.



Svaki tip serverske kontrole ima svoje karakteristike. Mi se njima, međutim, ovdje nećemo detaljno baviti (za samostalno proučavanje preporučujemo MSDN Library na putanji `.NET Development > .NET Framework SDK > .NET Framework > Reference > Class Library > System.Web.UI.HtmlControls`), već ćemo proučiti tek neke osnovne karakteristike.

Generička serverska kontrola

Bavit ćemo se generičkom serverskom kontrolom nazvanom `HtmlGenericControl`. Nju ćemo koristiti za sve elemente koji nemaju “svoju” serversku kontrolu. Međutim, sve stvari koje ovdje spomenemo vrijedit će i za ostale tipove serverskih kontrola.

Za primjer ćemo koristiti jednu kontrolu tipa `Label`. Jezikom HTML-a, radi se o običnom `DIV` elementu koji je automatski dobio ime `DIV1`. Nakon dovlačenja Labela i promoviranja u serversku kontrolu, njegov kôd će izgledati ovako:

III. DIO: DIJELOVI .NET-A

```
<DIV style="DISPLAY: inline; WIDTH: 70px; HEIGHT: 15px" ms_positioning="FlowLayout"
id="DIV1" runat="server">Label</DIV>
```

Želimo li iz pozadinskog kôda izmijeniti tekst koji se u Labelu nalazi, napisat ćemo sljedeći izraz:

```
DIV1.InnerText = "Novi tekst";
```

No to smo već znali. Važno je napomenuti da spomenuti izraz moramo smjestiti u neku funkciju. Želimo li da se spomenuta promjena teksta dogodi prilikom učitavanja stranice, smjestit ćemo je u funkciju `Page_Load` koja je vezana upravo uz taj događaj. Naravno, postoji još hrpa događaja uz koju možemo vezati funkcije s određenom funkcionalnošću.

Ukoliko želimo u Label smjestiti neki izraz u HTML-u, koristit ćemo sljedeći izraz:

```
DIV1.InnerHtml = "<a href=\"http://www.bug.hr/\">click here</a>";
```

Uočite kako na mjesto navodnika stavljamo izraz `\` – radi se o već spominjanom *escape*-znaku koji omogućava korištenje navodnika koji bi inače značili kraj znakovnog niza.

Želimo li promijeniti stil (CSS) kojim je Label ispisan, koristimo sljedeću sintaksu:

```
DIV1.Style["color"] = "Red";
DIV1.Style["font-weight"] = "bold";
```

To će na stranici rezultirati sljedećim izrazom:

```
<DIV style="color: Red; font-weight: bold;" ... >
```

Ukoliko želimo pristupiti nekom drugom atributu različitom od `Style`, primjerice svojstvu `Align`, napisat ćemo sljedeći izraz:

```
DIV1.Attributes["align"] = "right";
```

Na taj način možemo pristupiti svim atributima serverskih kontrola, bez obzira na to o kojoj se kontroli radi. Naravno, neke kontrole imaju svoja, specifična svojstva koja nam olakšavaju rad s njihovim karakteristikama. Tako, primjerice, kontrola `HtmlAnchor` ima svojstvo `HRef` kojim direktno možemo pristupiti istoimenom atributu, a kontrola `Button`, primjerice, ima događaj koji nastupa kada se na nju klikne.



Sve vrijednosti svojstava serverskih kontrola HTML-a su tipa *string*.

Kao što ste mogli vidjeti iz prošlih primjera, serverske kontrole HTML-a prilično su nespretne za korištenje i brojne njihove funkcije nisu lako dostupne kroz sučelje Visual Studija. One su namijenjene naprednijim korisnicima – onima koji preferiraju pisanje vlastitih klijentskih skripti i žele imati potpunu kontrolu nad odgovorima web-poslužitelja. Za jednostavnije i objektivnije programiranje tu su ASP.NET-ove web-kontrole.

ASP.NET-ove web-kontrole

Ove se kontrole nalaze u skupini Web Forms prozora Toolbox i puno su bolje podržane od strane Visual Studija. Zapravo, serverske kontrole HTML-a postoje radi kompatibilnosti, lakše nadogradnje i zahtjevnijih korisnika koji žele potpunu kontrolu – ne samo da se sve može riješiti pomoću web-kontrola, već to autori ASP.NET-a i očekuju. Zato ćemo se tim kontrolama nešto više posvetiti na stranicama što slijede.

Dovućemo li web-kontrolu na web-formu i nakon toga zavirimo u prikaz HTML-a, primijetit ćemo da se ne koriste standardni *tagovi* HTML-a, već neki posebni koji izgledaju otprilike ovako:

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

Učit ćete prefiks “asp:” koji ćemo pronaći na svim ugrađenim web-kontrolama. Osim tog prefiksa, kao što ćemo vidjeti kasnije, mogu postojati i drugi prefiksi za kontrole koje smo sami razvili ili preuzeli s Interneta.

Treba odmah jasno reći da je ovo serverski prikaz kôda – kad stranica bude procesirana, klijent će dobiti klasičan kôd u HTML-u i to, što je najbolje, prilagođen pregledniku koji koristi.

Primjer: Obrazac za kontakt

Kako ne bi sve ostalo na pukoj teoriji, poslužiti ćemo se primjerom. Ovoga puta ćemo izrađivati obrazac za kontakt u kojem će posjetitelj morati upisati određen broj podataka, koji će se nakon toga poslati na određenu e-mail adresu.

Za početak ćemo na formu smjestiti web-kontrole koje nam za primjer trebaju. Prvo ćemo postaviti jednu običnu tablicu (iz skupine HTML), čisto iz estetskih razloga. U prvi stupac ručno ćemo utipkivati natpise, dok ćemo u drugi postavljati adekvatne kontrole.

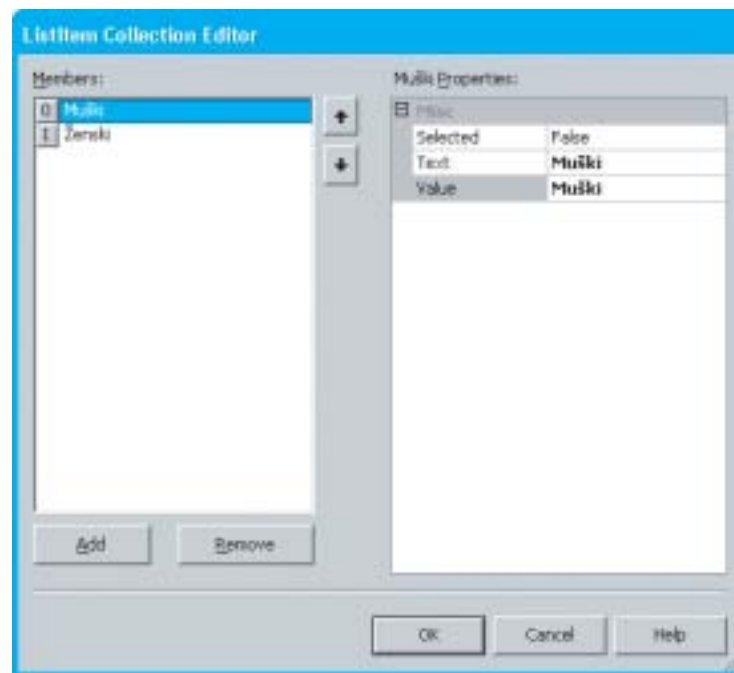
U prvom redu tako postavljamo kontrolu tipa TextBox – tekstualno polje za unos. Ono može biti jednolinijsko (svojstvo `TextMode` postavljeno na `SingleLine`), višelinijsko (`MultiLine`) ili sadržavati lozinku (`Password`). Osim tog svojstva pronaći ćemo i brojna druga svojstva. Od vizualnih treba izdvojiti svojstvo `CssClass` kojim kontroli pridružujemo klasu stilskeg predloška, a od ostalih spomenut ćemo svojstvo `Text` u kojem se nalazi znakovni niz upisan u kontrolu. Ostala svojstva prepoznat ćete i sami po njihovu imenu, a jednu skupinu svojstava ostavit ćemo za kasnije. Za potrebe primjera kontrolu smo preimenovali u `ImePrezimeText`.

III. DIO: DIJELOVI .NET-A

Slika 10-9:
Kako će primjer Obrazac za kontakt izgledati u konačnici

U drugom i trećem retku postavljamo dvije kontrole tipa DropDownList, koje ćemo nazvati DrzavaDDL i GradDDL. I tu nema novih i nepoznatih svojstava – možda samo treba napomenuti da se stavke tog padajućeg izbornika mogu definirati preko kolekcije Items. Mi to u ovom primjeru nećemo učiniti na taj način, no imajte na umu da je stvar vrlo slična prozorskom padajućem izborniku.

Slijedi još jedno tekstualno polje za upis (nazvano EMailText), a u petom ćemo redu susresti kontrolu tipa RadioButtonList (nazvanu SpolRadioButtonList). Ona nije ništa drugo nego skupina *radiobuttona* s kojom se radi na isti način kao i s kontrolom DropDownList. Izbore ćemo definirati preko svojstva Items, a odabranoj vrijednosti, kao što ćemo kasnije vidjeti, pristupat ćemo na potpuno isti način. Usto, ova kontrola krije neka zanimljiva svojstva. Naime, ASP.NET nam pruža mogućnost odluke kako će ponuđene opcije biti prikazane. Želimo li umjesto predefiniranog nabiranja izbora jedan ispod drugoga odrediti da budu smješteni u istom redu, mijenjat ćemo svojstvo RepeatDirection. Izbori mogu biti smješteni u posebnoj, nevidljivoj tablici (što izgleda ljepše) ili izvan nje. To odlučujemo svojstvom RepeatLayout. Konačno, ako izbora ima toliko da nam ne paše niti vodoravni niti okomiti raspored, možemo ih posložiti u tablicu s više stupaca i redova, što definiramo svojstvom RepeatColumns.



Slika 10-10:
Pomoćni prozor za
unos izbora u kontrolu
tipa RadioButtonList

U redu za pisanje poruke ubacili smo još jedno tekstualno polje, nazvali ga PorukaText, no ovaj put smo mu svojstvo TextMode postavili na vrijednost MultiLine.

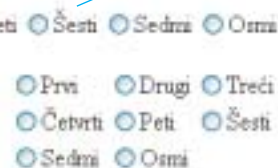
Slika 10-11:
Modaliteti svojstva RepeatDirection i RepeatColumns

```
RepeatDirection = RepeatDirection.Vertical;  
RepeatColumns = 0;
```



```
RepeatDirection = RepeatDirection.Vertical;  
RepeatColumns = 3;
```

```
RepeatDirection = RepeatDirection.Horizontal;  
RepeatColumns = 0;
```



```
RepeatDirection = RepeatDirection.Horizontal;  
RepeatColumns = 3;
```

III. DIO: DIJELOVI .NET-A

Na kraju dolazi obična kontrola tipa `CheckBox`, koju smo nazvali `MailingCheck`. U svojstvo `Text` valja staviti tekst koji će se iza nje pojaviti i na koji će ona biti vezana dok se uključenost odnosno isključenost može iščitavati svojstvom `Checked`, isto kao i kod prozorskih aplikacija.

Na kraju, na dnu obrasca, smještamo gumb – kontrolu tipa `Button` koja će služiti za slanje obrasca. Gumb smo nazvali `SlanjeButton`, a na tekst koji je na njemu upisali smo u svojstvo `Text`. Time smo zgotovili sučelje primjera.

Vraćanje stranice poslužitelju

Kao što smo već vidjeli kod serverskih kontrola HTML-a, cijela je stranica obrazac koji se vraća poslužitelju kada kliknemo na gumb. To vraćanje naziva se *postback* i ono, osim gumbom, može biti inicirano od strane bilo koje druge kontrole.

Objasnimo to primjerom. Nadogradit ćemo naš primjer tako da se, ovisno o izabranoj državi, popuni padajuća lista gradova. Za početak moramo napraviti neke pripremne radnje – napraviti dva polja s popisom država i gradova. Među deklaraciju varijabli (nakon deklaracije kontrola) dodajemo sljedeće linije:

```
...
protected System.Web.UI.WebControls.TextBox ImePrezimeText;

private string[] drzave = new string[4];
private string[] [] gradovi = new string[4] [];
```

Kreirana polja moramo popuniti podacima. U praksi bi se to vjerojatno čupalo iz neke baze podataka, no, jednostavnosti radi, mi ćemo koristiti direktnu populaciju na način koji slijedi. Kôd stavljamo u metodu vezanu uz događaj učitavanja stranice kako bi podaci bili od početka dostupni:

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    drzave[0] = "Hrvatska";
    drzave[1] = "Italija";
    drzave[2] = "Portugal";
    drzave[3] = "Brazil";

    gradovi[0] = new string[] { "Zagreb", "Split", "Rijeka", "Osijek" };
    gradovi[1] = new string[] { "Rim", "Siena", "Milano" };
    gradovi[2] = new string[] { "Lisabon", "Porto", "Fatima" };
    gradovi[3] = new string[] { "Sao Paolo", "Rio de Janeiro", "Minas Gerais" };
```

Nakon toga, trebamo te podatke smjestiti u pripadajuće padajuće izbornike. To ćemo učiniti ovako:

```

        DrzavaDDL.DataSource = drzave;
        DrzavaDDL.DataBind();
        GradDDL.DataSource = gradovi[DrzavaDDL.SelectedIndex];
        GradDDL.DataBind();
    }

```

Prva dva reda su jasna – prvom padajućem izborniku pridružujemo polje *drzave* kao DataSource, zatim te podatke “izvlačimo” u kontrolu metodom DataBind(). U drugom dijelu radimo istu stvar, no kako je polje *gradovi* dvodimenzionalno, radimo to samo s onim retkom koji pripada izabranoj državi. Kako je prva država na popisu Hrvatska, ovim komadom kôda će se kontrola GradDDL popuniti gradovima iz Hrvatske.

E sad, što kada posjetitelj izmijeni odabranu državu? U tom slučaju kontrolu GradDDL treba popuniti drugim podacima, gradovima koji se nalaze u toj državi. Stoga se treba vezati na događaj SelectionIndexChanged kontrole DrzavaDDL. U tu ćemo funkciju upisati sljedeće:

```

private void DrzavaDDL_SelectedIndexChanged(...)
{
    GradDDL.DataSource = gradovi[DrzavaDDL.SelectedIndex];
    GradDDL.DataBind();
}

```

Stvar bi besprijekorno radila u prozorskoj aplikaciji, no kod web-aplikacija imamo još malo posla. Naime, kako je ASP.NET serverska tehnologija, znamo da, ako želimo da se nešto na formi promijeni, moramo vratiti stranicu serveru odnosno napraviti *postback*. To bismo u našem primjeru napravili pritiskom na gumb, no želimo da se promjena izvrši odmah nakon promjene selekcije. Stoga među svojstvima kontrole DrzavaDDL treba pronaći AutoPostBack i postaviti ga na *true*. Na taj način smo stranici dali naredbu da promjena izbora u toj kontroli pošalje obrazac serveru na procesiranje.

Sjetimo se, zahvaljujući pamćenju stanja pomoću skrivenog parametra “__ViewState”, sustav će zapamtiti sve što smo dotada upisali pa, bez obzira na to što ponovo učitavamo stranicu, dotada upisane vrijednosti neće biti izgubljene.

Međutim, doći će do jedne druge neželjene nuspojave. Kada server vrati stranicu, ponovo će se izvršiti metoda Page_Load, čime će naše padajuće liste biti nanovo ispunjene. Tom radnjom izgubit će se učinjeni izbor pa cijela stvar uopće neće raditi kako smo očekivali.

Stoga spomenutoj metodi treba naložiti da punjenje padajućih izbornika napravi samo prilikom prvog učitavanja stranice. To ćemo napraviti pomoću svojstva Page.IsPostBack koji, ako ima vrijednost *true*, označava da je stranica vraćena (odnosno da je napravljen *postback*). Dakle, konačno verzija metode izgledat će ovako:

III. DIO: DIJELOVI .NET-A

```
private void Page_Load(object sender, System.EventArgs e)
{
    drzave[0] = "Hrvatska";
    drzave[1] = "Italija";
    drzave[2] = "Portugal";
    drzave[3] = "Brazil";

    gradovi[0] = new string[] { "Zagreb", "Split", "Rijeka", "Osijek" };
    gradovi[1] = new string[] { "Rim", "Siena", "Milano" };
    gradovi[2] = new string[] { "Lisabon", "Porto", "Fatima" };
    gradovi[3] = new string[] { "Sao Paolo", "Rio de Janeiro", "Minas Gerais" };

    if (!Page.IsPostBack)
    {
        DrzavaDDL.DataSource = drzave;
        DrzavaDDL.DataBind();
        GradDDL.DataSource = gradovi[DrzavaDDL.SelectedIndex];
        GradDDL.DataBind();
    }
}
```



Uočite da se punjenje podataka u polja nalazi izvan uvjetnog bloka i da će biti izvršeno pri likom svakog učitavanja stranice, bez obzira na to radi li se o vraćenoj stranici ili ne. Naime, pamćenje stanja kroz skriveni parametar vrijedi samo za kontrole, a ne i varijable koje definiramo u pozadinskom kodu.

Osim iniciranja slanja stranice serveru, kontrole možemo i isključiti iz pamćenja stanja. Drugim riječima, isključimo li kontroli svojstvo `EnableViewState` njezino stanje neće biti zapamćeno nakon vraćanja stranice serveru već će biti postavljeno na onu početnu, definiranu svojstvima u dizajnerskom načinu.



Želite li u potpunosti isključiti pamćenje stanja, jednostavno izbrišite obrazac u kojem se cijela stranica nalazi. Ipak, imajte na umu da ćete tim postupkom izgubiti velik dio funkcionalnosti.

Slanje elektroničke poruke

Pozabavimo se konačno slanjem *e-maila*. I sami pogađate – funkcionalnost ćemo vezati uz događaj klika na gumb. U pripadajuću metodu stavljamo sljedeće:

```
private void SlanjeButton_Click(object sender, System.EventArgs e)
{
    string Poruka = "";
    Poruka += "Ime i prezime: " + ImePrezimeText.Text + Environment.NewLine;
    Poruka += "Država: " + DrzavaDDL.SelectedValue + Environment.NewLine;
    Poruka += "Grad: " + GradDDL.SelectedValue + Environment.NewLine;
    Poruka += "Spol: " + SpolRadioList.SelectedValue + Environment.NewLine;
    Poruka += "Pretplata na mailing listu? " + (ListaCheck.Checked ? "Da" : "Ne")
        + Environment.NewLine;
    Poruka += Environment.NewLine;
    Poruka += PorukaText.Text + Environment.NewLine;

    SmtpMail.SmtpServer = "localhost";
    SmtpMail.Send(EMailText.Text, "primatelj@server.hr", "Poruka s Interneta",
        Poruka);
}
```

U prvih nekoliko redaka stvaramo poruku koja će biti poslana. Mogli smo sve smjestiti u isti redak, no ovako je preglednije. U tom dijelu možete vidjeti i kako se pristupa upisanim odnosno izabranim vrijednostima kontrola. Za prelazak u novi redak poruke koristimo konstantu `Environment.NewLine`.

Slanje poruke je izuzetno jednostavno i može se izvesti pomoću jedne linije. Ipak, prije korištenja moramo na početak datoteke dodati referencu na pripadajući *namespace*:

```
using System.Web.Mail;
```

Poruku šaljemo pomoću metode `Send`, kojoj kao parametre navodimo adresu pošiljatelja, adresu primatelja, temu poruke i samu poruku. Mi smo u primjer dopisali i definiranje svojstva `SmtpServer`, kako biste u svojim primjerima mogli koristiti i neki drugi od *defaultno* postavljenog.

Ovo je tek najjednostavniji oblik slanja poruke. Osim na ovaj način, poruke je moguće slati i na druge načine, koristeći druge metode i svojstva. Detalje potražite u dokumentaciji – ime *namespacea* znate.



III. DIO: DIJELOVI .NET-A

Provjera valjanosti upisanog

Iskusniji webaši primijetiti će da stvar još nije niti približno gotova. Što, primjerice, ako posjetitelj ne upiše svoju e-mail adresu i tako pokuša poslati poruku? Ili, što ako uopće ne ispunji obrazac, nego odmah klikne na gumb? Osim što od takvog praznog obrasca ne bi bilo nikakve koristi, u našem bi se slučaju javila i greška jer nije dozvoljeno poslati poruku bez adrese pošiljatelja, koja se vadi direktno iz kontrole `EMailText`. Drugim riječima, trebamo uvesti provjeru valjanosti upisanog ili, stručnim rječnikom, validaciju.



Validacija se, osim ukoliko drugačije ne postavimo, radi korištenjem klijentskog skriptiranja. Naravno, vi o tome ne morate brinuti – stvar je u potpunosti riješena unutar samih web-kontrola. Za posjetitelje s preglednicima koji ne podržavaju klijentske skripte, validacija će se vršiti na serverskoj strani.

Slika 10-12:
Validatori dodani na
kontrolu
ImePrezimeText i
EEmailText

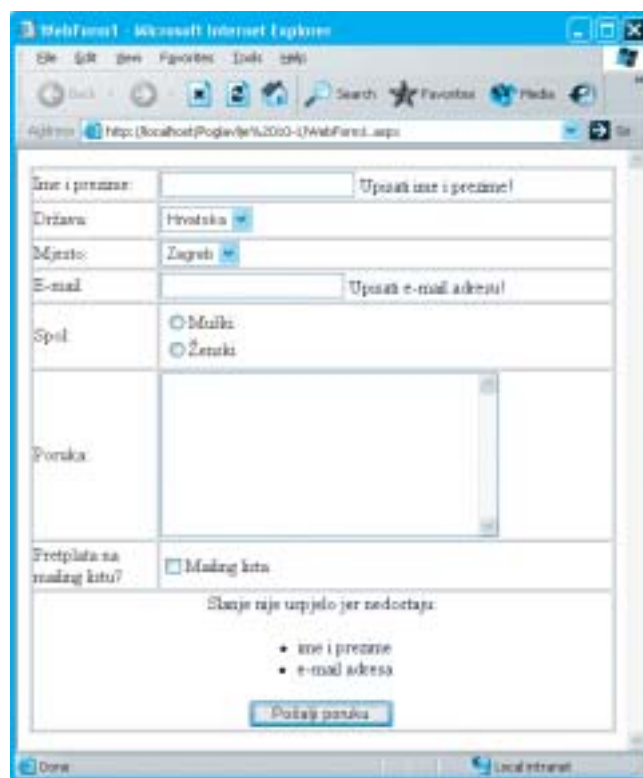
Dovucimo stoga dvije kontrole tipa `RequiredFieldValidator` i smjestimo ih kraj pripadajućih kontrola (vidi sliku 10-12). Imajte na umu da nije nužno da validacijske kontrole budu blizu onih na

10. POGLAVLJE: ASP.NET

koje su vezane, no tako je vizualno logičnije. Povezivanje validacijske kontrole s onom koju provjerava radimo preko svojstva `ControlToValidate`. U pripadajućem padajućem izborniku pojavit će se popis svih trenutno postojećih kontrola na web-formi, pa je dovoljno izabrati. Nakon toga, ta će se kontrola brinuti o tome da ne omogućava slanje obrasca a da nešto nije upisano u polje na koje je vezana.

Prema osnovnim postavkama validacijskih kontrola procedura će izgledati ovako: ako kliknemo na gumb a da nismo ništa unijeli u kontrole koje se provjeravaju, na stranici će se pojaviti dotada skrivene validacijske kontrole. Tekst koji želimo da prikazu u tom trenutku treba upisati u njihovo svojstvo `Text`.

Iako se ne vide, validacijske kontrole zauzimaju određeno mjesto na stranici. To je često neprihvatljivo ponašanje. Srećom, moguće ga je isključiti. To činimo tako da svojstvo `Display` iz `Static` promijenimo u `Dynamic`. Na taj način kontrola će zauzimati prostor na stranici samo kada bude vidljiva.



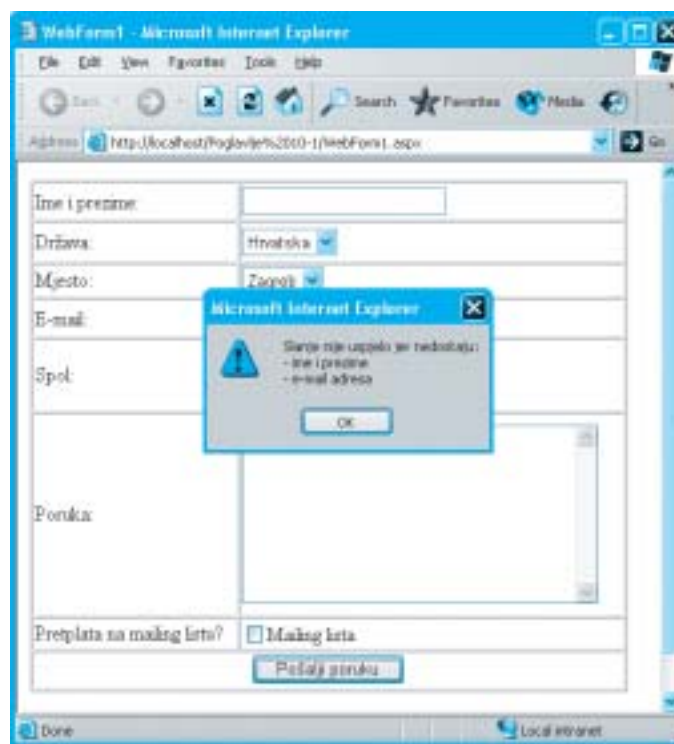
Slika 10-13:
Poruke validacijskih kontrola prikazane na oba mjesta

Ponekad je nezgodno da se upozorenje o neispunjenom polju javlja kraj njega. U tim slučajevima bismo radije da se poruke svih validacijskih polja prikazuju na istom, zajedničkom mjestu. Da bismo to postigli, dovući ćemo kontrolu tipa `ValidationSummary`. Samim time odredili smo zajedničko

III. DIO: DIJELOVI .NET-A

mjesto na kojem će se pojavljivati upozorenja svih validacijskih kontrola na stranici. Štoviše, svaka će se greška manifestirati na dva mjesta – na mjestu svoje validacijske kontrole i na zajedničkom prostoru. Te dvije poruke ne moraju biti identične – prvu, kao što smo već spomenuli, definiramo svojstvom Text, dok se na zajedničkom mjestu pokazuje ona smještena u svojstvo ErrorMessage. Želite li da se poruka pokazuje samo na zajedničkom dijelu, treba svim validacijskim kontrolama (osim ValidationSummary) svojstvo Display postaviti na vrijednost None.

Slika 10-14:
Iskakanje upozorenje o neispunjenosti polja

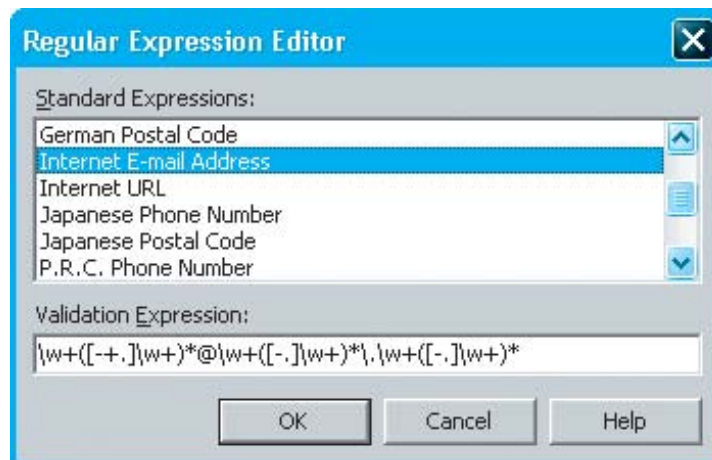


Po nama, najbolji način prikaza poruka o nepravilno ispunjenim poljima je upozorenje kakvo možete vidjeti na slici 10-14. S tim ciljem treba kontroli ValidationSummary uključiti svojstvo ShowMessageBox, a isključiti ShowSummary.

Naravno, moguće su i razne druge kombinacije spomenutih načina prikaza informacije o grešci.

U našem primjeru provjera postojanja bilo kakvog upisa za polje EMailText nije dovoljna. Naime, moramo posjetitelja upozoriti i u slučajevima kada upiše nevaljalu adresu elektroničke pošte. Za to će nam poslužiti kontrola RegularExpressionValidator koja provjerava zadovoljava li upisano određeni predložak.

Korištenje svih validacijskih kontrola je slično pa i kod ove treba proći istu proceduru vezanja uz kontrolu i definiranja načina ponašanja. Međutim, tu moramo definirati jedno svojstvo više – svojstvo `ValidationExpression` koje određuje predložak koji je dozvoljeno unijeti.



Slika 10-15:
Pomoćni prozor za izbor
validacijskog predloška

Sintaksa predložaka prilično je složena. Srećom, postoji predefinirani predložak za ono što nam u primjeru treba pa je dovoljno kliknuti na gumbić spomenutog svojstva te u prozoru koji će se otvoriti odabrati izraz “Internet E-mail Address” (slika 10-15).

Nakon toga, ta validacijska kontrola posjetitelja neće dalje puštati sve dok ne unese pravilnu adresu elektroničke pošte.

Osim spomenutih validacijskih kontrola postoje još neke koje ćemo tek telegrafski spomenuti. Prva među njima je `CompareValidator`, koja uspoređuje dvije kontrole (zgodno kod dvostrukog unosa lozinke) ili kontrolu uspoređuje s nekom predefiniranom vrijednošću. Isti se validator koristi i za insistiranje na određenom tipu unesenog podatka, primjerice datuma ili broja (svojstvo `Operator` postavlja se na `DataTypeCheck`, a `Type` na tip koji želimo tražiti).

Kontrola `RangeValidator` uspoređuje pripada li neka vrijednost određenom rasponu, dok nam kontrola `CustomValidator` omogućava samostalno programiranje validatora za koji ne postoji ugrađena kontrola.

I na kraju sage, otkrijmo što zapravo inicira proces provjere valjanosti. Radi se o svojstvu `CausesValidation` koji je u standardnoj postavi kontrola tipa `Button` (i još ponekih). Ukoliko ga isključimo, validacija neće nastupiti.

III. DIO: DIJELOVI .NET-A

Paneli

Kad smo već tako blizu, dotjerajmo primjer do savršenstva (barem što se programerskog dijela tiče, s vizualnošću bi se još štošta dalo učiniti). Naime, nedostaje nam potvrda da je poruka poslana. Umjesto nje, nakon slanja će se pojaviti isti obrazac, pa bi se moglo dogoditi da posjetitelj niti ne primijeti da je slanje obavljeno.

Slika 10-16:
Panele u dizajnerskom načinu prepoznamo po tankom sivom obrubu koji se u pokrenutoj aplikaciji ne vidi.

U tu ćemo svrhu na web-formu dovući dvije kontrole tipa Panel. Kao i kod prozorskih aplikacija, radi se o kontrolama koje nemaju vlastitu funkcionalnost, već sadrže druge kontrole. U prvu od njih ćemo smjestiti kompletno dosada napravljeno sučelje (jednostavno označite sve kontrole, desni klik na jednu od njih i iz padajućeg izbornika odaberite Cut, zatim desnom tipkom miša kliknite na Panel i odaberite Paste – imena i sva svojstva ostat će sačuvana). U drugi Panel ćemo smjestiti tekst zahvale. Panele nazovite ObrazacPanel i ZahvalaPanel, a potomjem svojstvo Visible postavite na *false*, kako se ne bi pokazivao na stranici.

Još nam preostaje napisati dvije linije kôda, na kraj funkcije vezane uz događaj klika na gumb:

```
ObrazacPanel.Visible = false;
ZahvalaPanel.Visible = true;
```

Nakon što poruka bude poslana, `ObrazacPanel` će se zajedno sa svim svojim kontrolama sakriti, a `ZahvalaPanel` bit će prikazan zajedno sa svojom rečenicom zahvale na ispunjenom obrascu.

Izrada korisničkih kontrola

Osim korištenja postojećih kontrola ili uključivanja tuđih, preuzetih s Interneta (vidi poglavlje o prozorskim aplikacijama), moguće je izrađivati i vlastite kontrole. Vaša kontrola može biti nadogradnja neke postojeće ili spajanje više različitih kontrola u neku logički povezanu funkcionalnost.

Vlastite kontrole koriste se kao i ugrađene, samo što, kao što ćemo kasnije vidjeti, imaju vlastiti prefiks. No krenimo redom...

Jednostavna web-kontrola

Prvo ćemo izraditi vrlo jednostavnu web-kontrolu koja će ispisivati IP-adresu posjetitelja. Postupak je sljedeći: u izborniku Project kliknite na stavku Add Web User Control. U prozoru upišite ime nove kontrole i potvrdite odabir klikom na OK.

Pokazat će se radna površina gotovo identična onoj koju smo vidjeli na početku izrade web-forme. U nju možete pisati tekst, stavljati HTML-elemente i koristiti serverske kontrole – sve što želite da vaša kontrola sadrži.

Mi ćemo za naš primjer otipkati tekst “Vaša IP-adresa:” te nakon njega staviti web-kontrolu Label. Zatim ćemo u pozadinskom kodu u metodi `Page_Load` upisati sljedeće:

```
private void Page_Load(object sender, System.EventArgs e)
{
    Label1.Text = Request.ServerVariables["REMOTE_ADDR"];
}
```

Tim kôdom ćemo u kontrolu Label1 upisati posjetiteljevu IP-adresu. Ona je zapisana u tzv. server-skoj varijabli `REMOTE_ADDR` kojoj pristupamo preko kolekcije `Request.ServerVariables`.

I to je to. Da bismo kontrolu pripremili za korištenje, trebamo napraviti *rebuild* cijelog projekta (jer naša je kontrola dio otvorenog projekta). Na taj način kontrola će se kompajlirati i moći ćemo je koristiti na svojoj web-formi. *Rebuild* radimo tako da desnom tipkom miša kliknemo na ime projekta u Solution Exploreru i iz izbornika izaberemo stavku Rebuild.

Korisničke kontrole imaju nastavak “.ascx”, što je skraćeno od *active server control*.



III. DIO: DIJELOVI .NET-A

Zatim se trebamo vratiti na dizajnerski način web-forme i jednostavno iz Solution Explorera dovući napravljenu kontrolu.



Slika 10-17:
Korisnička kontrola u dizajnerskom načinu izgleda ovako.

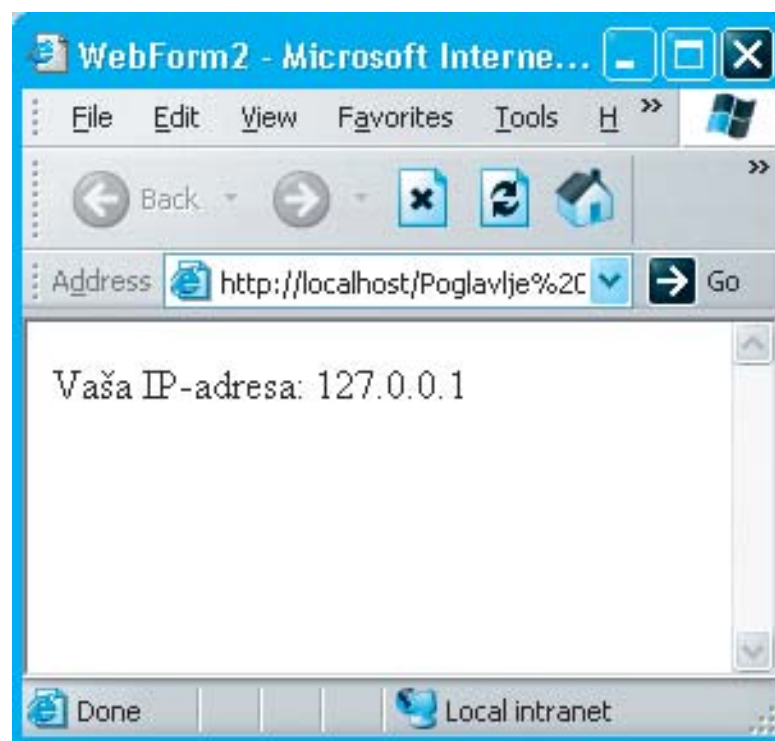


UserControl - WebUserControl11

U dizajnerskom načinu kontrola će izgledati kao na slici 10-17, no ako formu pokrenemo, ona će pokazati svoje pravo lice i funkcionalnost.



Slika 10-18:
Kad otvorimo formu u pregledniku, naša će korisnička kontrola izgledati ovako.



Pogledajmo što se dogodilo u pozadini. Otvorimo stoga web-formu tako da joj vidimo HTML-kôd. Odmah na početku otkrit ćemo sljedeću (žutu) liniju:

```
<%@ Register TagPrefix="uc1" TagName="WebUserControl1" Src="WebUserControl1.ascx" %>
```

Ta direktiva registrira našu komponentu za korištenje na web-formi. Parametrom `TagPrefix` određujemo njen prefiks, parametrom `TagName` ime *taga* koji ćemo koristiti, a `Src` označava datoteku u kojoj se kontrola nalazi.

Nešto kasnije u kodu pronaći ćemo sljedeću liniju:

```
<uc1:WebUserControl1 id="WebUserControl11" runat="server"></uc1:WebUserControl1>
```

Time pozivamo našu serversku kontrolu. Primijetite prefiks “uc1” i *tag* `WebUserControl1` koji smo definirali na vrhu stranice. Da tamo promijenimo prefiks i *tag*, morali bismo ih promijeniti i ovdje.

Jednom registriranu kontrolu na vrhu stranice u kodu možemo pozivati proizvoljan broj puta. Bitno je samo svaki put dodijeliti joj drugo identifikacijsko ime (parametar ID).

Složena web-kontrola

Priznajte, ovo je bilo prelagano. Zato ćemo se sada poigrati s nešto složenijom kontrolom kojoj ćemo prosljeđivati određeni parametar.

Da bismo to postigli, trebamo se podsjetiti da je svaka kontrola zapravo klasa. Dodavanje parametara web-kontroli zapravo je dodavanje parametara klasi, što smo već naučili u šestom poglavlju.

Napravit ćemo kontrolu koja će biti specijalizirana za prikaz linka na e-mail adresu. Tu funkcionalnost, dakako, možemo imati i sada, tako da kontroli `HyperLink` u svojstvo `NavigateUrl` ubacimo e-mail adresu s prefiksom “mailto:”. Međutim, našoj ćemo kontroli morati kao parametar dati samu e-mail adresu, a ona će se pobrinuti oko dodavanja prefiksa i postavljanja svojstva `Text`.

Otvorimo stoga novu korisničku kontrolu kao i u prošlom primjeru i dovucimo kontrolu tipa `HyperLink`. Zatim se prebacimo u pozadinski kôd i dodajmo sljedeće:

```
public string EMail
{
    set
    {
        HyperLink1.Text = value;
        HyperLink1.NavigateUrl = "mailto:" + value;
    }
    get
    {
        return HyperLink1.Text;
    }
}
```

III. DIO: DIJELOVI .NET-A

Deklarirali smo, dakle, javnu varijablu (parametar) EMail. U bloku *set* definiramo što će biti kada se njezina vrijednost postavlja – bit će pridružena svojstvu Text kontrole HyperLink1 kao i svojstvu NavigateUrl, ali s ubačenim prefiksom.

U drugom bloku (*get*) definiramo da ukoliko pristupamo varijabli EMail bude vraćena vrijednost svojstva HyperLink1.Text pošto se u njemu nalazi ta vrijednost. Nismo mogli vratiti vrijednost svojstva HyperLink1.NavigateUrl jer on osim adrese sadrži i prefiks.

U slučaju da se niti jedno svojstvo ne podudara s vrijednošću javne varijable, trebamo uvesti dodatnu, internu varijablu za pamćenje te vrijednosti. Ovako:

```
private string EMailAdresa;
public string EMail
{
    set
    {
        EMailAdresa = value;
        HyperLink1.Text = value;
        HyperLink1.NavigateUrl = "mailto:" + value;
    }
    get
    {
        return EMailAdresa;
    }
}
```

Nakon kompajliranja, ovu ćemo kontrolu moći koristiti na web-formi. Samu kontrolu možemo dovući iz Solution Explorera, dok ćemo parametar ubaciti ručno, na sljedeći način:

```
<uc1:SlozenaKontrola id="SlozenaKontrola1" runat="server"
    EMail="pero@pero.hr"></uc1:SlozenaKontrola>
```

Web-aplikacija

Sve što smo dosad spomenuli odnosilo se na jednu skriptu. ASP.NET, međutim, poznaje pojam aplikacije koja, pojednostavljeno rečeno, čini skup više skripti u istoj mapi i pripadajućim podmapama.

Inicijalno, svaki web-*site* definiran u IIS-u je zasebna web-aplikacija. Međutim, zasebnim aplikacijama možete proglasiti i mape unutar *sitea*. Sjetimo se da je Visual Studio kod kreiranja novog projekta stvorio virtualnu mapu koja je ujedno i samostalna web-aplikacija. Da biste to učinili sami, potrebno je ući u konzolu za konfiguraciju IIS-a (nalazi se u Control Panel > Administrative

ASP.NET-ove mobilne kontrole

Osim izrade web-stranica za "normalne" preglednike, .NET Framework omogućava izradu tzv. mobilnih web-stranica korištenjem ASP.NET-ovih mobilnih kontrola. Radi se o kontrolama koje su svojim karakteristikama i mogućnostima prilagođene prikazu na mobilnim uređajima.

Da biste napravili mobilnu web-stranicu, potrebno je prilikom kreiranja projekta ili dodavanja nove forme izabrati predložak Mobile Web Form. Nakon toga će se u prozoru Toolbox pojaviti nova skupina kontrola pod nazivom Mobile Web Forms.

Mobilne web-forme mogu se prikazivati na cijeloj paleti mobilnih uređaja, od najjednostavnijih mobitela s mogućnošću prikazivanja stranica WAP-a do ručnih računala koja podržavaju gotovo sve funkcionalnosti "pravih" preglednika. Dakako, te se stranice mogu pregledavati i pomoću preglednika na osobnom računalu. Naravno, vi se oko kompatibilnosti ne morate brinuti – kôd koji se šalje klijentskom pregledniku automatski će se prilagođavati njegovu tipu.

Tools), označiti mapu koju želimo promovirati u aplikaciju, ući u prozor Properties i kliknuti na Create, kao što vidimo na slici 10-19. Na taj način možete promovirati i virtualne i fizičke mape.



Slika 10-19:
Promoviranje mape u web-aplikaciju

III. DIO: DIJELOVI .NET-A

Komunikacija servera i klijenta

Skripte okupljene pod kapom web-aplikacije imaju određene zajedničke karakteristike, no na kraju se sve uvijek svodi na to da su skripte vrlo samostalne. Web-stranice od svojih početaka funkcioniraju na isti način – korisnik zatraži određenu stranicu, server mu je pošalje i više ne komuniciraju. Među klijentom i serverom ne postoji stalna veza, svaki zahtjev za stranicom zasebna je komunikacija, koja prestaje nakon što se stranica isporuči.

Naravno, poslužitelj i klijent razmjenjivat će i druge podatke osim web-stranice. Većina te komunikacije obavlja se “skriveno”, preko zaglavlja web-dokumenta kojeg posjetitelj najčešće nije niti svjestan. Klijent prilikom traženja stranice šalje zaglavlje koje sadrži brojne informacije o tipu klijenta, stranici koju traži, kolačićima i sličnom. Server prilikom odgovaranja na upit također vraća zaglavlje u kojem daje informacije o stranici koju šalje. Zahvaljujući toj međusobnoj komunikaciji moguće je prenošenje vrijednosti među stranicama koje bi inače bile dva potpuno odvojena svijeta.

Prenošenje vrijednosti među web-stranicama je općenito moguće na dva načina – preko spomenutog zaglavlja ili preko parametara u adresi stranice. Vraćanje stranice poslužitelju (*postback*) riješeno je na ovaj prvi način i u potpunosti automatizirano, no kad spajamo različite skripte moramo se pouzdati u vlastite sposobnosti iskorištavanja tih načina komunikacije.



Slanje parametara preko adrese naziva se *get*, a slanje preko zaglavlja stranice naziva se *post*. Adekvatnim parametrom u HTML-u, u formularu je moguće koristiti oba načina.

Treba odmah na početku naglasiti da na ovaj način nije uobičajeno prenositi velike količine podataka. Primjerice, nikad nećete prenositi cijeli zapis baze, nego samo njegovo identifikacijsko polje, a ostale podatke ćete pomoću posebnog upita nanovo izvući iz baze.



Prenošenje parametara ne mora vršiti isključivo između skripti unutar iste web-aplikacije. Parametre možete prebacivati i skriptama druge aplikacije, čak i ako se nalaze na drugom serveru.

Bilo kakvo povezivanje skripata zapravo je obično linkanje. Primjerice, želimo li sa stranice Skripta1.aspx omogućiti odlazak na stranicu Skripta2.aspx, dovoljno je postaviti link (bilo u HTML-u, bilo pomoću web-kontrole HyperLink) na tu stranicu. Primjerice:

10. POGLAVLJE: ASP.NET

```
<asp:HyperLink id="HyperLink1" runat="server" NavigateUrl=" Skripta2.aspx">Skok na
drugu stranicu</asp:HyperLink>
// ili jednostavnije:
<a href="Skripta2.aspx">Skok na drugu stranicu</a>
```

Želimo li u adresi prenijeti neki parametar, koristit ćemo sljedeći izraz:

```
<a href="Skripta2.aspx?parametar1=vrijednost&parametar2=vrijednost">Klikni me</a>
```

U odredišnoj ćemo skripti proslijeđeni parametar pročitati na sljedeći način:

```
string parametar1 = Request.QueryString["parametar1"];
string parametar2 = Request.QueryString["parametar2"];
```

Naravno, u svojoj skripti parametar ne morate pridružiti varijabli – možete ga smjestiti u svojstvo Text neke kontrole te tako ispisati na stranici ili iskoristiti na neki drugi način.

Parametrima proslijeđenima kroz formular pristupamo ovako:

```
string parametar1 = Request.Form["parametar1"];
string parametar2 = Request.Form["parametar2"];
```

Na taj način, iako to u praksi najčešće nije potrebno, možete pristupati i vrijednostima polja generiranih od strane web-kontrola.

Želite li za startnu stranicu koja se otvara u pregledniku nakon kompajliranja postaviti neku drugu umjesto prve, u Solution Exploreru kliknite desnom tipkom miša i iz izbornika izaberite "Set As Start Page".



Osim linkanja koje zahtijeva da posjetitelj klikne na vezu, korisnika možete prebaciti na drugu skriptu naredbom u kodu. Primjerice, da smo željeli u našem primjeru Obrasca za kontakt nakon uspješnog slanja poruke posjetitelja preusmjeriti na posebnu stranicu za zahvalom, koristili bismo sljedeću liniju:

```
Response.Redirect("zahvala.aspx");
```

I u ovom slučaju možemo prosljeđivati parametre u adresi – dovoljno ih je samo dopisati.

III. DIO: DIJELOVI .NET-A



Metodu `Response.Redirect()` radi tako da javi pregledniku (klijentu) neka zatraži drugu stranicu, što ovaj posluša i na taj način preusmjeri korisnika. Želite li da se redirekcija obavi bez znanja klijenta, koristite metodu `Server.Transfer()`.

Sesijske varijable

I sami ste vjerojatno zaključili kako je prenošenje parametara na ovaj način prilično nepraktično. Naime, mogu se prenositi samo parametri tipa *string* (doduše, oni se mogu konvertirati u ostale tipove u kodu, no to može biti nespretno i nesigurno), a i postoji mogućnost da korisnik “upadne” u parametre i promijeni ih.

Stoga u ASP.NET-u (kao i njegovim starijim verzijama) poslužitelj za svakog posjetitelja održava tzv. sesiju. Naime, prvi put kada neki posjetitelj zatraži neku skriptu naše aplikacije, otvara se nova sesija vezana uz njega. Ta sesija će posjetitelja pratiti do kraja njegova posjeta našoj web-aplikaciji.



Određivanje kraja sesije vrlo je neprecizno. Naime, kako ne postoji stalna veza između klijenta i servera, server ne zna kada je posjetitelj otišao s njegovih stranica, a kada samo nešto čeka i planira nastaviti surfanje po njima. Stoga se za kraj sesije najčešće uzima 20 minuta od zadnje aktivnosti na stranicama. Znači, ako korisnik 20 minuta ništa ne dira po vašim stranicama, njegova će sesija biti ugašena. On se možda vrati nakon 25 minuta, no u tom će slučaju za njega biti kreirana nova sesija.

Sesija može pamtiiti varijable. Primjerice, ako u kodu neke stranice pridružite vrijednost varijable na sljedeći način:

```
Session["Ime varijable"] = "vrijednost";
```

...ona će biti dostupna i na svim ostalim stranicama koje će nakon ove taj posjetitelj otvoriti. Pazite – vrijednost varijable bit će dostupna samo za tog istog posjetitelja, a ne i za sve ostale posjetitelje. Svaki od njih će imati vlastite sesijske varijable s vlastitim vrijednostima.



Sesijske varijable kojima nije pridružena vrijednost imaju vrijednost *null*.

10. POGLAVLJE: ASP.NET

Sesijske varijable nije potrebno deklarirati niti im je moguće odrediti tip (uvijek su tipa *object*). To u praksi znači da ćete im moći pridruživati vrijednosti bilo kojeg tipa. Primjerice:

```
Session["Varijabla"] = "1";
Session["Varijabla"] = 3;
```

Iako ova praksa izgleda primamljivo, preporučujemo vam da to ne koristite radi preglednosti i dosljednosti – radije napravite dvije različite sesijske varijable, za svaki tip vrijednosti po jednu. Ukoliko vam neka od njih ne treba, napišite sljedeće:

```
Session.Remove("Ime varijable");
```

Međutim, najvažniji razlog je nužnost konverzije prilikom korištenja s ostalim varijablama. Primjerice, prilikom čitanja vrijednosti sesijske varijable i njezina spremanja u varijablu tipa *string*, morat ćemo napraviti konverziju:

```
string varijabla = Session["Ime varijable"].ToString();
```

Sesijske se varijable čuvaju na poslužitelju i posjetitelj im ne može pristupiti niti znati da su kreirane. Može, međutim, na razne načine inicirati prekid sesije, pa stoga ona nije potpuno pouzdan način čuvanja vrijednosti.



Pomoću ovih varijabli najčešće se rješavaju stvari poput postavki posjetitelja koje treba pamtit i tijekom cijelog posjeta ili brojanje stranica koje je posjetio tijekom jednog posjeta.

Vjerojatno ste iz sintakse primijetili da se zapravo radi o kolekciji. Dakle, postoji mogućnost da nad njom radimo sve što i s klasičnom kolekcijom, kao što je šetanje kroz sve njene vrijednosti. Uzmimo za primjer da smo postavili sljedeće serijske varijable:

```
Session["Jedan"] = "1";
Session["Dva"] = 3;
Session["Tri"] = true;
```

Kroz cijelu kolekciju tih varijabli možemo proći na jedan od sljedeća dva načina:

```
for (int n = 0; n < Session.Count; n++)
{
```

III. DIO: DIJELOVI .NET-A

```

        Response.Write(Session[n] + "<br>");
    }

    // ili...

    foreach (string x in Session)
    {
        Response.Write(Session[x] + "<br>");
    }

```

Dakle, varijablama je moguće pristupiti preko ključa (prvi dio primjera) i preko indeksa (drugi dio).



Naredba `Response.Write` jednostavno ispisuje parametar koji joj damo na web-stranicu. Ona se u principu ne koristi u pozadinskom kodu (posebno ne metodi `Page_Load` jer ispisuje stvari na sam početak dokumenta, prije zaglavlja HTML-a, što nije pravilno), no može poslužiti za demonstraciju.

Kolačići

Za kolačiće (engl. *cookies*) ste sigurno čuli, ako ni zbog čega drugoga, onda zbog poslovične fobije od gubitka privatnosti. Tehnički, radi se o malenim tekstualnim datotekama na posjetiteljevu računalu u koje web-skripte mogu zapisivati neke njima potrebne podatke.



Sustav za održavanje sesija također koristi kolačić u koji zapisuje identifikacijski kôd sesije.

I kolačići su vezani isključivo uz jednog korisnika, no oni se ne brišu završetkom sesije, već ostaju i za sljedeći posjet, koji može biti za nekoliko sati, nekoliko dana ili čak nekoliko mjeseci.

Drugim riječima, želite li neke vrijednosti vezane uz određenog posjetitelja zapamtiti i nakon kraja sesije, koristit ćete kolačiće.

Kolačiće koristimo nešto drugačije. Prvi red predstavlja pisanje, a drugi čitanje (uočite razlike u naredbama!):

```
Response.Cookies["Ime varijable"].Value = "vrijednost";
string varijabla = Request.Cookies["Ime varijable"].Value;
```

Budući da je vrijednost kolačića tipa string nije bila potrebna konverzija. Međutim, u njega ne možete spremiti, primjerice, broj – ako ga prije ne konvertirate:

```
Response.Cookies["Nekakav broj"].Value = 345.ToString();
```



Međutim, s kolačićima možemo raditi mnoge druge stvari. Osim svojstva Value, svaki kolačić ima neka dodatna svojstva, od kojih ćemo kao najzanimljiviji izdvojiti Expires. Njime definiramo rok trajanja kolačića. Primjerice, sljedeći kolačić će trajati do 27. svibnja 2005. u 10 sati i 50 minuta:

```
Response.Cookies["Ime varijable"].Expires = new DateTime(2005, 5, 27, 10, 50, 00);
```

Rok trajanja možemo odrediti i relativno u odnosu na trenutni datum i vrijeme. Evo kolačića koji će trajati deset dana (od trenutka izvršenja ove naredbe):

```
Response.Cookies["Ime varijable"].Expires = DateTime.Now.AddDays(10);
```

Aplikacijske varijable

I sesijske varijable i kolačići odnose se na samo jednog posjetitelja. Što kada želimo spremiti neku vrijednost koja treba biti dostupna svim korisnicima web-aplikacije, svim posjetiteljima naših web-stranica? U tom ćemo slučaju koristiti aplikacijske varijable.

Sintaksa njihova korištenja ista je kao i kod sesijskih varijabli. Pogledajmo:

```
Application["Ime varijable"] = "vrijednost";
varijabla = Application["Ime varijable"].ToString();
```

Sve napisano za sesijske varijable vrijedi i ovdje, osim činjenice da su aplikacijske dostupne svim korisnicima web-aplikacije. Međutim, iz tog razloga mogu uzrokovati probleme.

Zamislimo situaciju da u nekoj skripti na stranici imamo sljedeći izraz (prvi redak je za slučaj da aplikacijska varijabla nije definirana):

```
if (Application["Broj"] == null) Application["Broj"] = 0;
Application["Broj"] = (int)(Application["Broj"]) + 1;
```

III. DIO: DIJELOVI .NET-A

Zadatak drugog retka je da poveća vrijednost navedene aplikacijske varijable za jedan. No što ako u isto vrijeme dva korisnika aplikacije pokrenu istu naredbu? Nastat će problem. Zato kod pridruživanja vrijednosti aplikacijskim varijablama treba koristiti zaključavanje. U sljedećem primjeru ćemo zaključati aplikacijske varijable, napraviti promjenu i zatim otključati zaključano:

```
Application.Lock();
if (Application["Broj"] == null) Application["Broj"] = 0;
Application["Broj"] = (int)(Application["Broj"]) + 1;
Application.Unlock();
```



Zaključavanje odnosno otključavanje uvijek koristite samo i isključivo neposredno prije odnosno poslije pridruživanja vrijednosti kako bi aplikacijske varijable bile zaključane što kraće. Naime, dok su aplikacijske varijable zaključane svi će ostali posjetitelji čekati na njihovo otključavanje, što može drastično utjecati na performanse.



Osim spomenutih načina, ASP.NET omogućava još jedno mjesto pamćenja varijabli na aplikacijskom nivou. Radi se o zajedničkim (statičnim) varijablama klase koje, kao što smo spomenuli u šestom poglavlju, poprimaju samo jednu vrijednost bez obzira na broj instanciranih objekata i zajedničke su svim instancama.

Događaji aplikacije

Kao svaki pravi objekt, i aplikacija ima svoje događaje. Događaje aplikacije pronaći ćete u datoteci Global.asax koja je automatski kreirana prilikom stvaranja projekta.

U njoj su već predefinirane metode povezane na osnovne aplikacijske događaje. Mi ćemo ovdje spomenuti četiri osnovne, one vezane uz događaje početka i završetka aplikacije te početka i završetka sesije.

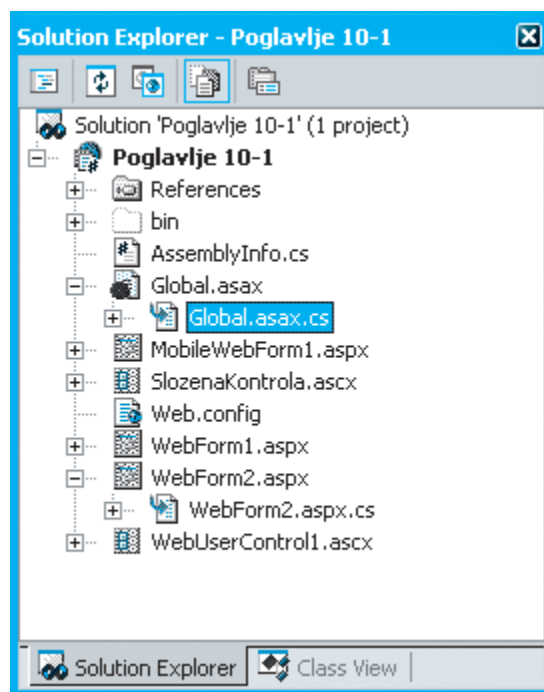
Najčešća namjena tih metoda je postavljanje inicijalnih vrijednosti aplikacijskih odnosno sesijskih varijabli (tako da izbjegnemo onu provjeru `Application["Broj"] == null` iz nedavnog primjera). Evo jednog vrlo čestog primjera – brojanje trenutno aktivnih posjetitelja na stranicama:

10. POGLAVLJE: ASP.NET

```
protected void Application_Start(Object sender, EventArgs e)
{
    Application["Broj posjetitelja"] = 0;
}

protected void Session_Start(Object sender, EventArgs e)
{
    Application.Lock();
    Application["Broj posjetitelja"] = (int)(Application["Broj posjetitelja"]) + 1;
    Application.Unlock();
}

protected void Session_End(Object sender, EventArgs e)
{
    Application.Lock();
    Application["Broj posjetitelja"] = (int)(Application["Broj posjetitelja"]) - 1;
    Application.Unlock();
}
```



Slika 10-20:
*Datoteka Global.asax
automatski je kreirana pri-
likom stvaranja projekta.*

III. DIO: DIJELOVI .NET-A

Što ovaj primjer radi? Najprije u metodi vezanoj uz događaj starta aplikacije postavljamo inicijalnu vrijednost aplikacijske varijable Broj posjetitelja. U metodu događaja koji nastupa prilikom otvaranja nove sesije (kako se sesija otvara za svakog novog posjetitelja, događaj nastupa prilikom dolaska posjetitelja na stranice) povećavamo varijablu Broj posjetitelja za jedan, dok prilikom zatvaranja sesije (znači, kad posjetitelj ode sa stranica), tu varijablu smanjujemo za jedan. Na taj način u varijabli Broj posjetitelja uvijek imamo aktualan broj posjetitelja na stranici.

Događaji vezani uz kraj sesije i aplikacije (ovaj potonji ne koristimo u primjeru) nisu pouzdani. Uzimimo banalan primjer – grešku na serveru koja dovede do naglog prekida izvršavanja web-aplikacije. U tom slučaju događaji kraja sesije i aplikacije uopće neće nastupiti. Drugim riječima, u njih ne biste smjeli stavljati stvari koje su ključne za aplikaciju. Primjerice, ne bi bilo pametno napraviti da se nešto broji u aplikacijskog varijabli s namjerom da se prilikom gašenja aplikacije to zapiše u bazu podataka.

Kod zatvaranja sesije vrijedi ista priča, uz nekoliko iznimki. Primjerice, nećete moći pomoću tog događaja posjetitelju ispisati poruku u stilu “Hvala što ste posjetili naše stranice” jer ih je on u tom trenutku već davno napustio, no moći ćete, primjerice, mijenjati aplikacijske varijable kao u primjeru – ako se dogodi havarija spomenuta u prošlom odlomku, ionako će sve vrijednosti aplikacijskih varijabli biti izgubljene pa njihova točnost pada u drugi plan.

Web-aplikacija ima i brojne druge događaje. Primjerice, pomoću događaja BeginRequest i EndRequest možemo se ubaciti na početak odnosno kraj svakog odgovora poslužitelja na upit klijenta dok se aplikacijski događaj Error javlja svaki put kada se dogodi neuhvaćena greška pa ga možete iskoristiti za nekakvo izvještavanje o greškama.

Postavke stranice

Sigurno ste na vrhu stranice u HTML-u primijetili da je prvih par redova žute boje. Te su naredbe drugačije od ostatka kôda i nazivaju se direktive. Njima definiramo određene parametre vezane uz stranicu.

Najkorištenija direktiva je ona nazvana Page, a izgleda otprilike ovako:

```
<%@ Page language="c#" Codebehind="WebForm2.aspx.cs" AutoEventWireup="false"
Inherits="Poglavlje_10_1.WebForm2" %>
```

Kad kreirate novu stranicu u Visual Studiju bit će generirana otprilike takva direktiva. Iz nje trenutno možemo iščitati sljedeće: bit će pisana u jeziku C#, pozadinski kôd nalazi se u datoteci WebForm2.aspx.cs i nasljeđuje klasu Poglavlje_10_1.WebForm2.

Parametar AutoEventWireup nešto je složeniji – sjetimo se da uz svaki događaj trebamo vezati neku metodu koju želimo da se izvrši prilikom tog događaja. Međutim, ako postavimo ovu vrijednost na *true* (ili je ne navedemo – *true* je njezina *defaultna* vrijednost), to povezivanje ne moramo radi-

ti – sustav će sam pozivati metode `Page_Init` i `Page_Load` bez obzira na to što nisu povezane sa “svojim” događajima. Visual Studio stavlja ovu vrijednost na *false* stoga što on u automatski generiranom pozadinskom kodu povezuje događaj `Load` s metodom `Page_Load`. Stoga ako uključimo još i ovu opciju ta će se metoda pozivati dva puta!

Direktiva `Page` se može koristiti samo jednom na svakoj stranici. Kontrole ne mogu imati ovu direktivu, kod njih se koristi slična direktiva `Control`, s nešto manje dostupnih parametara.



```
<%@ Page ... Buffer="false" ... %>
```

Ako ne navedemo parametar `Buffer` ili ga postavimo na *true*, prvo će poslužitelj generirati kompletnu web-stranicu namijenjenu klijentu, a tek onda je zaista i poslati. Ukoliko stavimo *false*, stranica će se klijentu slati dio po dio, kako koji bude generiran. U praksi uključeni *buffer* omogućava da u zadnji čas otkažete slanje stranice posjetitelju i preusmjerite ga na neku drugu stranicu.

```
<%@ Page ... EnableSessionState="false" ... %>
```

Isključivanjem opcije `EnableSessionState` isključit ćete stranici mogućnost da koristi sesijske varijable. Ukoliko pak tom parametru pridružite vrijednost `ReadOnly`, stranica će sesijske varijable moći samo čitati.

```
<%@ Page ... EnableViewState="false" ... %>
```

Parametar `EnableViewState` uključuje odnosno isključuje pamćenje stanja kontrola na stranici. Imajte na umu da na ovaj način nećete u potpunosti isključiti postojanje skrivenog parametra “`__VIEWSTATE`” – on će i dalje sadržavati mali izraz pamteći neka osnovna stanja stranice.

```
<%@ Page ... EnableViewStateMac="true" ... %>
```

Uključite li opciju `EnableViewStateMac`, skriveni parametar stanja bit će kodiran kako bi se spriječila mogućnost korisnikovog mijenjanja tog parametra. Imajte na umu da uključivanje ove opcije utječe na performanse servera pa ga koristite samo na stranicama na kojima je integritet podataka ključan.

```
<%@ Page ... ErrorPage="greske.aspx" ... %>
```

III. DIO: DIJELOVI .NET-A

Parametrom `ErrorPage` određujete stranicu na koju će posjetitelj biti preusmjeren u slučaju nastupa neuhvaćene greške.

```
<%@ Page ... RequestEncoding="iso-8859-2" ResponseEncoding="iso-8859-2" ... %>
```

Parametri `RequestEncoding` i `ResponseEncoding` određuju način prikazivanja “neengleskih” znakova na stranicama. Kod .NET-a *defaultna* postavka je UTF-8, univerzalni set znakova koji pokriva većinu svjetskih jezika, što uključuje i naš. Međutim, možete poželjeti da se posjetitelju web-stranice poslužuju na nekoj drugoj kodnoj stranici.

Parametar `RequestEncoding` govori u kojem setu znakova stranica očekuje zahtjeve od klijenta, dok `ResponseEncoding` određuje kako će poslužitelj odgovoriti na zahtjev odnosno u kojem setu znakova će stranica biti poslana klijentu.

U navedenom primjeru stranica će očekivati upit i na njega odgovarati koristeći centralnoeuropski ISO-standard.

Tablica 10-2:
***Neki od dostupnih encodinga
za ispravan prikaz “naših
znakova”***

Kratica	Ime
utf-8	Unicode (UTF-8)
iso-8859-2	Central European (ISO)
windows-1250	Central European (Windows)
ibm852	Central European (DOS)

U tablici 10-2 možete vidjeti neke od mogućnosti za prikaz “naših znakova”. Radi kompatibilnosti preporučujemo vam da koristite Central European (ISO) – njega podržava najveći broj preglednika na svim sustavima, iako će kroz koju godinu, kada postane još bolje prihvaćen, Unicode biti najbolji izbor.

```
<%@ Page ... Trace="true" ... %>
```

Praćenje parametara stranice koje uključujemo parametrom `Trace` jedno je od najkorisnijih svojstava ASP.NET-a, pogotovo u slučajevima kad nešto krene po zlu, a vi nikako ne možete otkriti o čemu se radi.

Ono, naime, na kraj stranice dodaje nekoliko tablica popunjenih svim relevantnim podacima o stranici (vidi sliku 10-21). Ti podaci uključuju osnovne informacije (identifikacijsku oznaku sesije, vrijeme, korištene setove znakova, tip zahtjeva, status), informacije o procesiranju stranice, stablo kontrola, popis svih sesijskih i aplikacijskih varijabli zajedno s njihovim tipovima i vrijednostima,

10. POGLAVLJE: ASP.NET

Slika 10-21:
Uključeno praćenje para-
metara stranice donos
pregršt informacija.

U praćenje parametara stranice možete dodavati i svoje parametre iz pozadinskog kôda. To može poslužiti za ispisivanje vrijednosti varijabli za koje niste sigurni kako se ponašaju. Sintaksa je sljedeća:

```
Trace.Write("Varijabla Var1 je jednaka " + Var1.ToString());
```



sve dostupne kolačiće, detaljne informacije o zaglavlju poslane od strane posjetiteljeva preglednika (iz kojih, primjerice, možemo vidjeti koji preglednik posjetitelj koristi) kao i kompletan popis serverskih varijabli koje sadrže apsolutno sve informacije o poslužitelju, klijentu i njihovoj vezi.

I parametara i direktiva ima još. Jednu od direktiva, Register, već smo upoznali prilikom korištenja korisničkih kontrola na stranicama, a još jednu među njima upoznat ćemo kasnije u poglavlju, u priči o *cacheiranju*. Ostale koje se rjeđe koriste ostavljamo vam na samostalno proučavanje – putanja u dokumentaciji je .NET Development > .NET Framework SDK > .NET Framework > Reference > ASP.NET Syntax > Web Forms Syntax > Directive Syntax.



III. DIO: DIJELOVI .NET-A

Konfiguracijske datoteke

Određene postavke možete postaviti i na nivou aplikacije smještajući ih u konfiguracijske datoteke. Te će se postavke poštovati na svakoj stranici aplikacije. Na taj način može se definirati neopisivo velik broj stvari, a mi ćemo, kao i uvijek, obraditi korištenije i zanimljivije.

Ipak, prije nego što krenemo na seciranje konfiguracijskih datoteka, red je da objasnimo njihovu hijerarhiju. Postoji, naime, jedna glavna konfiguracijska datoteka koja se zove “machine.config” i nalazi se u mapi “C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\CONFIG”. Ona predstavlja konfiguracijsku datoteku za cijelo računalo – sve što je u njoj definirano vrijedit će u svim web-aplikacijama na tom računalu. Osim ako neke druge konfiguracijske datoteke ne prepisu određenu opciju.

Naime, svaki web-site također može imati svoju konfiguracijsku datoteku. Ona se međutim naziva “web.config” i nalazi se u njegovoj osnovnoj mapi, tzv. *rootu*. Postavke u toj datoteci prepisat će one u glavnoj konfiguracijskoj datoteci.

Nadalje, svaka web-aplikacija na određenom *siteu* može imati svoju konfiguracijsku datoteku. Primjerice, aplikacija s adresom “<http://localhost/WebApplication1/>” u svojoj će mapi (WebApplication1) također vjerojatno imati svoju konfiguracijsku datoteku nazvanu “web.config”.

Kada naša web-aplikacija traži koje postavke treba koristiti, prvo zavrjuje u glavnu konfiguracijsku datoteku i uzima tamo postavljene opcije. Zatim gleda datoteku koja se nalazi na adresi “<http://localhost/web.config>” (uvjetno rečeno, jer je tim datotekama nemoguće pristupiti na taj način) i opcijama koje tamo pronađe prepíše one koje je pokupila iz glavne datoteke. Konačno, ulazi u konfiguracijsku datoteku same aplikacije (dakle, uvjetno rečeno, “<http://localhost/WebApplication1/web.config>”) i tamo pronađenim vrijednostima postavlja konačnu konfiguraciju.

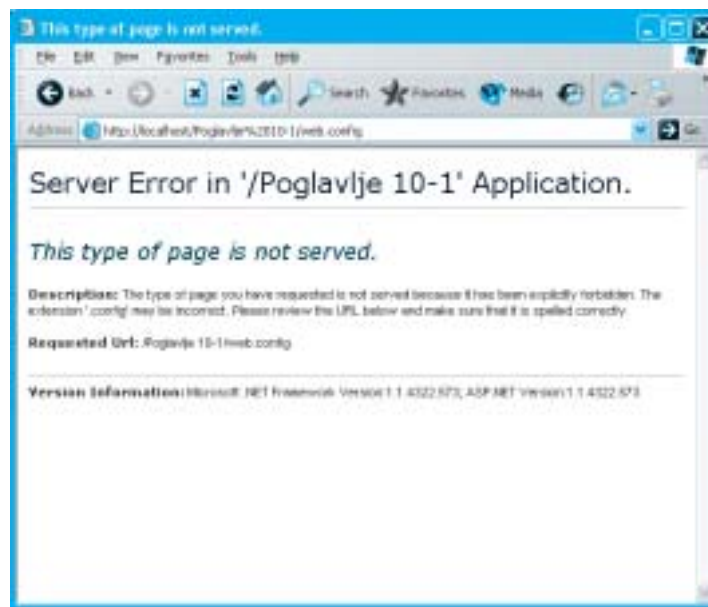
Zvuči jednostavno? Nismo još gotovi. Naime, konfiguracijska datoteka višeg stupnja može zabraniti prepisivanje parametara pomoću konfiguracijske datoteke nižeg stupnja. Drugim riječima, ako u datoteci “machine.config” odredite da se ne može prepisivati opcije pomoću datoteka “web.config”, onda će svaki pokušaj prepisivanja rezultirati greškom.



Preporučujemo vam da konfiguracijske datoteke, kao uostalom i sve datoteke koje koristite u programiranju, otvarate i mijenjate pomoću Visual Studija. Naime, sve se datoteke prema inicijalnim postavkama spremaju u formatu Unicode, što mnogi editori ne podržavaju. Nakon snimanja datoteka s takvim programima one bi mogle ispasti neispravne i neupotreblijive.

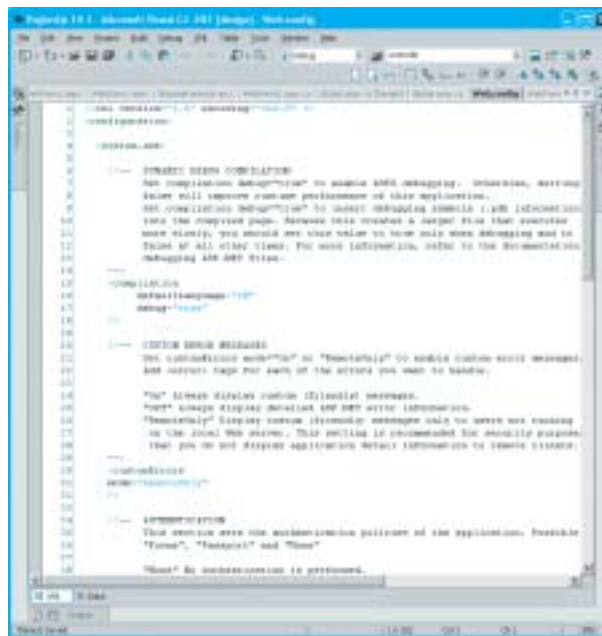
Cijela ova priča nije od krucijalne važnosti – u pravilu ćete se baviti isključivo opcijama na nivou konfiguracijske datoteke za aplikaciju. Što postavite tamo, vrijedit će za cijelu vašu aplikaciju.

10. POGLAVLJE: ASP.NET



Slika 10-22:
Poruka koja se javi kad pokušate preko weba pristupiti datoteci web.config

Kako prema inicijalnim postavkama prepisivanje nije zabranjeno, moći ćete definirati sve što želite. Ipak, naletite li na neki strogo konfiguriran poslužitelj, ove će vam informacije uštedjeti mnogo truda.



Slika 10-23:
Datoteka web.config otvorena u Visual Studiju

III. DIO: DIJELOVI .NET-A



Prilikom svake promjene konfiguracijske datoteke aplikacija će se automatski restartati i nove će vrijednosti početi vrijediti.

Igranje s opcijama konfiguracijske datoteke “web.config” vrlo je jednostavno jer se te datoteke automatski kreiraju prilikom stvaranja novog projekta, a neke su mogućnosti i objašnjene u samoj datoteci u obliku komentara.



Konfiguracijske datoteke osjetljive su na razliku između malih i velikih slova.

Konfiguracijske datoteke zapravo su XML-dokumenti. Iako njih još zvanično nismo obradili (to nas čeka u sljedećem poglavlju), vjerujemo da nećete imati problema sa savladavanjem njihove sintakse.

Statične varijable

U konfiguracijsku datoteku možemo zapisati neke statične varijable koje će vrijediti za cijelu aplikaciju. To se najčešće koristi za spremanje *connection stringa* za pristup bazi podataka, kao što ćemo vidjeti u primjeru nešto kasnije u poglavlju. Evo primjera:

```
<configuration>
  <appSettings>
    <add key="ConnectionString" value="neki connection string"/>
  </appSettings>
</configuration>
```

Sve se postavke u konfiguracijskoj datoteci nalaze između *tagova* <configuration> i </configuration>. Između njih stavljamo *tagove* <appSettings> i </appSettings> unutar kojih, pomoću vidljive sintakse, dodajemo ime varijable (parametar *key*) i vrijednost (parametar *value*).

Toj vrijednosti iz pozadinskog kôda možemo pristupiti na sljedeći način (uočite nužno navođenje *namespacea* na vrhu):

```
using System.Configuration;
...
string cs = ConfigurationSettings.AppSettings["ConnectionString"];
```

Postavke stranice

Neke postavke koje smo definirali na stranici možemo, umjesto da ih postavljamo na svakoj stranici posebno, definirati i na nivou cijele aplikacije. Naravno, ukoliko stranica ima drugačije postavljenu vrijednost, ova u konfiguracijskoj datoteci bit će prepisana (ignorirana).

Kodne stranice ćete gotovo sigurno poželjeti definirati na nivou cijele aplikacije jer nema smisla da svaka stranica biva enkodirana na drugi način.

```
<configuration>
  <globalization
    requestEncoding="iso-8859-2"
    responseEncoding="iso-8859-2"
  />
</configuration>
```

Neki se drugi parametri koje smo upoznali kod direktive Page u konfiguracijskog datoteci definiraju na ovaj način:

```
<configuration>
  <system.web>
    <pages
      buffer="true"
      enableSessionState="true"
      enableViewState="true"
      autoEventWireup="true"
    />
  </system.web>
</configuration>
```

Pažljivo uočavajte gdje su pojedine opcije smještene. Primjerice, opcija <pages> obavezno mora biti u sekciji <system.web>.



Praćenje parametra

Praćenje parametara na nekoj stranici možemo uključiti pomoću direktive Page. Međutim, želimo li uključiti to praćenje za sve stranice unutar aplikacije, uključit ćemo opcije enabled i pageOutput. Prva će općenito uključiti praćenje parametara, dok će druga naložiti da se informacija ispisuje na svakoj stranici.

III. DIO: DIJELOVI .NET-A

```
<configuration>
  <system.web>
    <trace
      enabled="true"
      localOnly="true"
      pageOutput="true"
      requestLimit="15"
    />
  </system.web>
</configuration>
```

Ukoliko parametar pageOutput isključimo, do informacija o praćenju ćemo moći doći preko posebne, virtualne adrese – <http://localhost/trace.axd>. Datoteka na kraju adrese, dakako, ne postoji, no web-poslužitelj će nam vratiti stranicu s popisom svih zahtjeva nad aplikacijom (vidi sliku 10-24). Klikom na link “View Details” možemo vidjeti detalje o svakom zahtjevu kao što je bilo prikazivano na kraju stranice.

Slika 10-24:
Stranica s popisom
svih zahtjeva nad
aplikacijom

No.	Time of Request	File	Status Code	Verb	Remaining
1	21.3.2004 9:44:48	/WebForm2.aspx	200	GET	View Details
2	21.3.2004 9:49:24	/WebForm2.aspx	200	GET	View Details
3	21.3.2004 9:49:26	/WebForm2.aspx	200	GET	View Details
4	21.3.2004 9:49:26	/WebForm2.aspx	200	GET	View Details
5	21.3.2004 9:49:26	/WebForm2.aspx	200	GET	View Details
6	21.3.2004 9:49:27	/WebForm2.aspx	200	GET	View Details
7	21.3.2004 9:49:28	/WebForm1.aspx	200	GET	View Details
8	21.3.2004 9:49:30	/WebForm2.aspx	200	GET	View Details
9	21.3.2004 9:49:30	/WebForm2.aspx	200	GET	View Details
10	21.3.2004 9:49:32	/WebForm1.aspx	200	GET	View Details

Kako zahtjeva u aplikaciji može biti izuzetno puno, parametrom requestLimit određujemo koliko će zadnjih zahtjeva biti pamćeno i prikazano na stranici.

Informacije o praćenju parametara inicijalno su dostupne samo s lokalnog računala. O tome se brine parametar localOnly. Kada bismo ga isključili, te bi informacije mogli vidjeti svi posjetitelji naših stranica.

Upravljanje sesijom

Iako je sesija kod ASP.NET-a u praksi ostala gotovo ista u odnosu na starije inačice, u samoj biti promijenjeno je mnogo toga. Tako, primjerice, za cijeli posao oko održavanja sesija možemo zadužiti neki vanjski server, što posebno dolazi do izražaja u slučajevima kada se više servera brine o istoj web-aplikaciji (što se naziva web-farma). Mi u detalje toga nećemo ulaziti, no zato ćemo spomenuti mogućnost održavanja sesije bez kolačića.

Naime, kao što smo već spomenuli, sustav za održavanje sesije koristi kolačić kako bi zapisao identifikacijski kôd sesije. Ukoliko posjetitelj preglednik ima isključene kolačiće (da, ima i takvih...), rješenje može biti uključivanje parametra *cookieless* i korištenje tzv. sesije bez kolačića.

```
<configuration>
  <system.web>
    <sessionState
      cookieless="false"
      timeout="20"
    />
  </system.web>
</configuration>
```

U tom će se slučaju identifikacijski kôd prenositi u adresi stranice koja će izgledati, primjerice, ovako:

```
http://localhost/ImeAplikacije/(so5os0550bxammv5a5skvivz)/WebForm1.aspx
```

Korisnik identifikacijski kôd na ovaj način može vrlo jednostavno promijeniti i tako "iskočiti" iz postojeće sesije i napraviti novu. Doduše, ista se stvar može napraviti i s kôdom zapisanim u kolačić, no to je znatno složenije.



Konačno, u konfiguracijskoj datoteci možemo i izmijeniti rok trajanja sesije nakon zadnjeg zahtjeva za stranicom. Parametru *timeout* navodi se vrijeme u minutama.

Poruke o greškama

Iz poruka o greškama može se puno saznati o strukturi i načinu rada web-aplikacije. Stoga je poželjno da takve poruke ne dođu u ruke osobama zlih namjera, a kako su web-stranice dostupne svima, znači da bi svima trebalo prikazivati tek informaciju da je greška nastupila, a detalje zadržati u tajnosti.

III. DIO: DIJELOVI .NET-A

```
<configuration>
  <system.web>
    <customErrors
      mode="RemoteOnly"
    />
  </system.web>
</configuration>
```

Ovako postavljena vrijednost parametra određuje da se općenita informacija o grešci (tzv. “prijateljska” poruka odnosno engl. *friendly error message*) pokazuje svim vanjskim posjetiteljima, a korisniku na lokalnom računalu prikazat će se detaljna poruka o grešci.

Parametar mode može poprimiti vrijednosti i “On” (svima će biti prikazana općenita poruka) ili “Off” (svima će biti prikazana detaljna poruka).

Osim toga, možemo definirati da se umjesto generičke osnovne poruke prikaže neka naša. Primjerice, želimo li da svim vanjskim posjetiteljima umjesto osnovne poruke bude prikazana stranica “error.aspx”, u konfiguracijsku datoteku ćemo smjestiti sljedeći izraz:

```
<configuration>
  <system.web>
    <customErrors
      mode="RemoteOnly"
      defaultRedirect="error.aspx"
    />
  </system.web>
</configuration>
```

Autentikacija i autorizacija korisnika

Zabrana pristupa nepoznatim (anonimnim) korisnicima često se koristi kod stranica koje nisu namijenjene javnosti. U ASP.NET-u je napravljena izuzetna podloga za implementaciju autentikacije i autorizacije korisnika.

Prema inicijalnim postavkama svatko može pristupiti web-stranicama. Takvi se posjetitelji nazivaju anonimni korisnici.



Autentikacija je proces utvrđivanja tko je došao na stranice, a autorizacija je utvrđivanje ima li on pravo pristupa određenim stranicama.

10. POGLAVLJE: ASP.NET

```
<configuration>
  <system.web>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
</configuration>
```

Ovakva postavka omogućava svim korisnicima da pristupaju našim stranicama.

Kada bismo napisali sljedeće, svima bismo onemogućili pristup:

```
<deny users="*" />
```

Međutim, mi želimo postići da se pristup zabrani samo anonimnim korisnicima, pa ćemo umjesto gornjih izraza napisati ovo:

```
<deny users="?" />
```

Ako posjetitelj dođe na web-aplikaciju s ovakvom postavkom, trebat će se autentificirati. Nakon uspješne autentifikacije on neće više biti anonimni korisnik, pa će moći pristupiti aplikaciji.

Autentifikaciju možemo napraviti na četiri načina, a to definiramo sljedećim izrazom:

```
<authentication mode="Windows" />
```



Slika 10-25:
Prozor za upis korisničkog imena i lozinke prilikom korištenja autentifikacijskog načina Windows

III. DIO: DIJELOVI .NET-A

Ako je autentikacijski način postavljen na "Windows", znači da će posjetitelji morati upisati korisničko ime i lozinku koji postoje u bazi korisnika poslužitelja (ili domene u kojoj se on nalazi).

Osim autentikacijskog načina Windows (koji je najjednostavniji za implementaciju), dostupni su načini Forms (autentikacija se obavlja putem posebno napravljene stranice s obrascem za unos imena i lozinke), Passport (autentikacija putem Microsoftova servisa .NET Passport) i None (za *siteove* kojima autentikacija nije potrebna ili imaju vlastiti sustav autentikacije).



Isprobavanje autentikacije nećete jednostavno moći obaviti na lokalnom računalu jer vas poslužitelj neće tretirati kao anonimnog korisnika, već kao korisnika s korisničkim računom kojim ste prijavljeni na računalu.

Cacheiranje

Web-stranice često istovremeno posjećuje velik broj posjetitelja. Generiranje stranica za svakoga posebno zahtjevan je posao i postoji mogućnost da vaš server to ne može podnijeti. Stoga u ASP.NET-u postoje tri vrste *cacheiranja* – funkcionalnosti koja omogućava spremanje stranice, njezinih dijelova ili podataka u memoriju. Ako neki posjetitelj zatraži stranicu koja je spremljena u *cache*, ona neće biti ponovo generirana, već će se posjetitelju poslati direktno iz *cachea*.

Imajte na umu da *cacheiranje* nije idealno rješenje u svim prilikama. Općenito možemo reći da ukoliko web-stranica ima puno posjetitelja kojima se prikazuje na isti način *cacheiranje* može biti vrlo korisno i može značajno ubrzati posluživanje stranica. Međutim, u slučaju, recimo, pretraživača u koji svaki posjetitelj upisuje drugačiji izraz, *cacheiranje* nema nikakvog smisla – stranica će se ionako svaki put generirati iznova zbog drugačijih parametara, a stranice spremljene u *cache* nitko neće koristiti pa će nepotrebno zauzimati memoriju i tako usporavati rad poslužitelja.

Cacheiranje stranica

Želimo li neku stranicu uključiti u proces *cacheiranja* moramo joj na početak dodati sljedeću direktivu:

```
<%@ OutputCache Duration="100" VaryByParam="none" %>
```

Direktivi OutputCache u primjeru navodimo dva parametra. Prvi određuje trajanje *cache* u sekundama – u našem slučaju znači da će stranica biti spremljena u memoriji sljedećih sto sekundi.

Ako neki posjetitelj pozove stranicu nakon isteka tog vremena, ona će biti iznova generirana i opet spremljena u *cache* na sljedećih sto sekundi.

Ukoliko će se stranica pozivati s određenim parametrima (bilo u adresi – GET, bilo kroz zaglavlje – POST), poželjet ćete se poigrati parametrom *VaryByParam*. Naime, pomoću njega možete odrediti hoće li se stranica s drugačijim parametrima tretirati kao sasvim nova stranica ili će, unatoč drugačijim parametrima, posjetitelju biti poslužena ona iz memorije. Ako stavimo vrijednost “none”, parametri će biti ignorirani, a ako stavimo zvjezdicu (“*”), parametri će se uvažavati. Moguće je i rješenje između – možete nabrojati parametre koji će biti uvažavani, dok će ostali biti ignorirani. Primjerice:

```
<%@ OutputCache Duration="100" VaryByParam="location;count" %>
```

Osim po parametru, stranicu je moguće diferencirati i prema zaglavlju (*VaryByHeader*) i nekom vlastitom uvjetu (*VaryByCustom*).

Ako koristite *cacheiranje* na stranicama koje se generiraju iz baze podataka, imajte na umu da se promjene na bazi neće manifestirati na stranicama sve dok *cache* ne istekne i stranica se ponovo generira. Zato je praksa da se kao trajanje *cacheiranja* stavljaju manje vrijednosti.



Cacheiranje dijelova stranice

Na gotovo isti način možete *cacheirati* i dijelove stranice. Doduše, dijelove koje želite *cacheirati* najprije morate pretvoriti u korisničke kontrole te onda kontrolama dodati direktivu *OutputCache* na isti način kao što smo to u prošlim primjerima dodavali stranicama.

```
<%@ OutputCache Duration="100" VaryByParam="none" %>
```

Kada koristimo direktivu *OutputCache* na kontrolama, osim već spomenutih parametara, otvara nam se još jedan koji bi trebalo spomenuti – *Shared*.

Naime, kako kontrole možemo koristiti na više stranica, treba odlučiti smije li poslužitelj istu kontrole na različitim stranicama tretirati kao jednake. Inicijalno će se u memoriju spremati zasebna kopija kontrole za svaku stranicu, a ukoliko to želimo izbjeći, napisat ćemo sljedeće:

```
<%@ OutputCache Duration="100" VaryByParam="*" Shared="True" %>
```

III. DIO: DIJELOVI .NET-A

Cacheiranje podataka

Varijable *cachea* su svojevrsna alternativa aplikacijskim varijablama. Naime, jednom postavljena aplikacijska varijabla zauzima prostor u memoriji do kraja izvršavanja aplikacije koja na stabilnim produkcijskim web-poslužiteljima može trajati godinama. Varijable u kodu se čiste iz memorije nakon što je stranica poslužena, sesijske se brišu po isteku sesije, dok aplikacijske traju, traju i traju...

Varijable *cachea* po svemu su identične aplikacijskima uz, naravno, dvije sitnice. Prvo, prilikom mijenjanja njihovih vrijednosti one se automatski zaključavaju i, drugo, poslužitelj se brine da one koje se duže vremena ne koriste automatski izbrišu iz memorije.

Njih se najčešće koristi za, primjerice, spremanje stvari kao što su lista opcija za padajući izbornik (da se ne vuče svaki put iz baze podataka) ili pak neke druge vrijednosti koje su iste za sve korisnike. Naravno, svaki put treba provjeriti je li tako spremljen podatak istekao, no na taj način možemo značajno uštedjeti na resursima poslužitelja.

Varijable *cachea* se koriste prema već viđenom modelu i sve rečeno za sesijske i aplikacijske vrijedi i ovdje:

```
Cache["Ime varijable"] = "vrijednost";
string varijabla = Cache["Ime varijable"].ToString();
```

Međutim, vjerojatnije ćete u ove varijable spremati neke veće količine podataka. Recimo da želimo polje iz primjera Obrazac za kontakt spremati u *cache* kako se ne bi trebalo iznova puniti prilikom svakog učitavanja stranice. Evo kako ćemo to napraviti:

```
if (Cache["drzaveArray"] == null)
{
    drzave = new string[] { "Hrvatska", "Italija", "Portugal", "Brazil" };
    Cache.Insert("drzaveArray", drzave, null, DateTime.Now.AddMinutes(10),
        TimeSpan.Zero);
}
else
{
    drzave = (string[])(Cache["drzaveArray"]);
}
```

Najprije uvjetom provjeravamo postoji li zapisana vrijednost u varijablu *drzaveArray*. Ako je jednaka vrijednosti *null* znači da ne postoji, bilo da nikada nije niti postojala, bilo da je istekla. U tom slučaju punimo polje podacima (u našem slučaju na ovaj loš način, kod vas vjerojatno iz neke baze ili datoteke) i zatim ga metodom *Insert* spremamo u varijablu *cachea*. Prvo navodimo ime varijable, a zatim i vrijednost koju želimo smjestiti u varijablu *cachea*. Sljedeći parametar predstavlja neku zavisnu vrijednost koja će odlučiti o isteku valjanosti *cachea* (mi smo stavili *null*, što označava da

nema zavisnih vrijednosti). Slijedi definiranje roka valjanosti – prvi parametar je određuje konkretnim vremenom (u našem slučaju na vrijeme deset minuta nakon postavljanja), dok drugi određuje rok isteka u relativnom obliku, od zadnjeg pristupa varijabli. Primjerice, da smo tu stavili vrijednost `TimeSpan.FromMinutes(1)`, varijabla bi istekla minutu nakon što je zadnji put korištena (ne kreirana!).

Parametre za istek varijable nije potrebno navoditi, no ako ih navodite, samo jedan od njih smije imati konkretnu vrijednost. Drugim riječima, ako postavite prvi na neku vrijednost (primjerice `DateTime.Now.AddMinutes(10)`), drugi mora biti `TimeSpan.Zero`. Ukoliko pak drugi ima neku vrijednost (primjerice `TimeSpan.FromMinutes(1)`), prvi morate postaviti na `DateTime.MaxValue`.

Rad s bazama podataka

U pozadini web-stranica najčešće možemo naći baze podataka. Stoga ćemo se sada posvetiti obradi kontrola koje nam u tome mogu puno pomoći – `DataGrid`, `DataList` i `Repeater`. Prvu ćemo nešto detaljnije upoznati, a druge dvije ćemo samo ukratko objasniti. Međutim, kako sve tri rade na sličan princip, i njih ćete brzo “prokužiti”.

Naravno, trebat će nam i znanja koja smo usvojili u prošlom poglavlju u kojem smo baze podataka upoznali teoretski i korištenjem u prozorskoj aplikaciji, a ponovit ćemo i neke stvari koje smo spomenuli u prvom dijelu ovog poglavlja.

No krenimo redom...

Kontrola `DataGrid`

Kontrola `DataGrid` služi za prikaz podataka iz baze u obliku tablice. Evo krajnje jednostavnog primjera – postavite na stranicu kontrolu `DataGrid` (imena `DataGrid1`), a u pozadinski kôd napišite sljedeće:

```
private void Page_Load(object sender, System.EventArgs e)
{
    if (!Page.IsPostBack)
    {
        Bindanje();
    }
}

private void Bindanje()
{
    SqlConnection sqlConn = new
```

III. DIO: DIJELOVI .NET-A

```
SqlConnection(ConfigurationSettings.AppSettings["connstr"]);

SqlDataAdapter sqlComm = new SqlDataAdapter("SELECT * FROM Orders", sqlConn);

DataSet ds = new DataSet();
sqlComm.Fill(ds);

DataGrid1.DataSource = ds;
DataGrid1.DataBind();
}
```



Nemojte zaboraviti referencu na *namespaceove* na početku pozadinskog kôda:

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



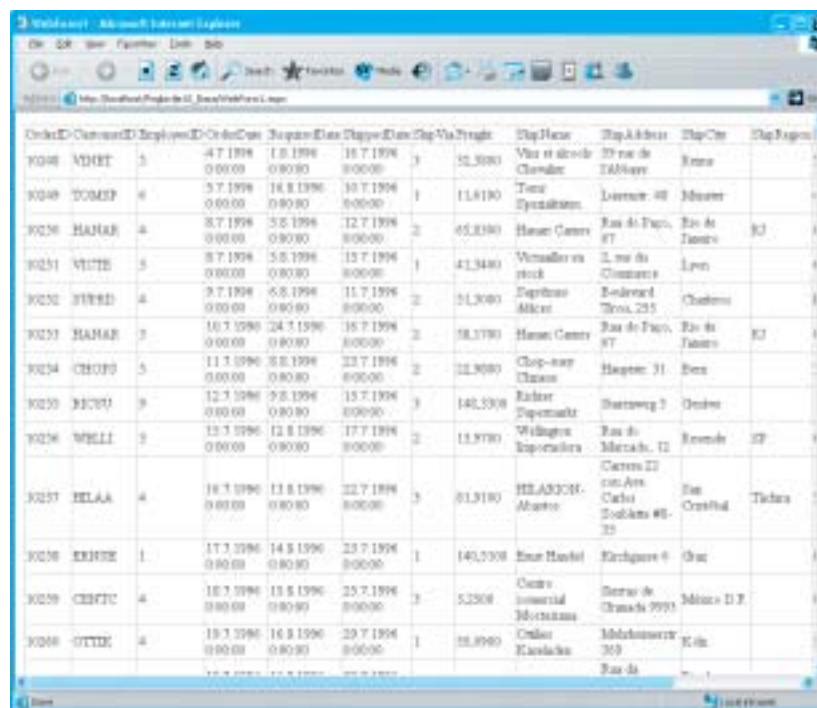
Mi ćemo u primjerima koristiti bazu podataka Northwind smještenu na lokalnom SQL Serveru. Naravno, uz sitne modifikacije prema uputama iz prošlog poglavlja, sve će raditi i s nekim drugim izvorom podataka.

Primjećujete da *connection string* izvlačimo iz konfiguracijske datoteke web.config (što je preporučljiva praksa), pa stoga u nju treba dodati sljedeće:

```
<appSettings>
  <add
    key="connstr"
    value="server=(local);database=Northwind;Integrated Security=true;"
  />
</appSettings>
```



Kod povezivanja iz web-aplikacije na SQL Server trebate imati na umu da stranice njemu pristupaju pod posebnim korisničkim računom nazvanim ASPNET. Da bi taj korisnički račun mogao pristupiti bazi podataka, potrebno ga je dodati među korisnike SQL Servera i pridružiti mu potrebna prava nad bazom podataka koja će se koristiti.



OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion
1048	VIDE	5	4.7.1998	1.8.1998	16.7.1998	3	32.3000	Via et al. de	39 rue de	Paris	
1049	TOYER	4	5.7.1998	16.8.1998	10.7.1998	1	11.6100	Tony	12 rue de	Montreuil	
1050	HANAR	4	8.7.1998	5.8.1998	12.7.1998	2	65.8300	Hanan	Rue de Paris,	Eto de	EC
1051	WHITE	5	8.7.1998	5.8.1998	15.7.1998	1	41.3400	White	1 rue de	Lyon	
1052	STEED	4	9.7.1998	6.8.1998	11.7.1998	2	51.3000	Steed	1 rue de	Chateaux	
1053	HANAR	5	10.7.1998	24.7.1998	16.7.1998	2	58.3700	Hanan	Rue de Paris,	Eto de	EC
1054	CHOPD	5	11.7.1998	8.8.1998	23.7.1998	2	12.9000	Chop	1 rue de	Paris	
1055	WHEEL	5	12.7.1998	9.8.1998	15.7.1998	3	140.3300	Wheel	1 rue de	Paris	
1056	WHEEL	5	13.7.1998	12.8.1998	17.7.1998	2	15.8700	Wheel	1 rue de	Paris	
1057	HELAA	4	16.7.1998	13.8.1998	22.7.1998	3	61.3100	HELAA	1 rue de	Paris	
1058	ERHSE	1	17.7.1998	14.8.1998	23.7.1998	1	140.3300	Erhse	1 rue de	Paris	
1059	CHETU	4	18.7.1998	15.8.1998	25.7.1998	3	5.2500	Chetu	1 rue de	Paris	
1060	OTTE	4	19.7.1998	16.8.1998	26.7.1998	1	15.8900	Otte	1 rue de	Paris	

Slika 10-26:
Web-aplikacija
koja pomoću
DataGrida
prikazuje tablicu
Orders

Pokrenemo li aplikaciju, dobit ćemo tabelarni prikaz svih zapisa koje smo dohvatili SQL-upitom u funkciji Bindanje(), kao što je prikazano na slici 10-26.

Vizualno dotjerivanje

Osim izuzetno jednostavnog postizanja ove funkcionalnosti, primijetit ćete da je dobivena tablica nevjerojatno ružna. Stoga je treba vizualno dotjerati pomoću brojnih dostupnih svojstava.

DataGrid možete urediti na dva načina – prvi je korištenjem predefiniranih stilova koji se kriju iza linka Auto Format (vidi dno slike 10-27), dok je drugi način ručno podešavanje svojstava u grupama Appearance, Layout i Style. Mi nećemo ulaziti u detalje opisujući karakteristike svakog pojedinog svojstva – vjerujemo da su im imena dovoljno intuitivna.

Zanimljivo je pogledati kako su spomenute stavke manifestirane u kodu. Dosad smo uvijek sva svojstva dodavali kao atribut, no ovdje se situacija nešto razlikuje – neka su svojstva i dalje atributi osnovne kontrole, dok su stilovi pojedinih redova zasebni tagovi između otvorenog i zatvorenog taga kontrole. Evo primjera:

III. DIO: DIJELOVI .NET-A

```
<asp:DataGrid id="DataGrid1" runat="server" BorderColor="#E7E7FF" BorderStyle="None"
BorderWidth="1px" BackColor="White" CellPadding="3" GridLines="None">
  <ItemStyle ForeColor="#4A3C8C" BackColor="#E7E7FF"></ItemStyle>
  <AlternatingItemStyle BackColor="#F7F7F7"></AlternatingItemStyle>
</asp:DataGrid>
```

Slika 10-27:
Svojstva za vizualnu prilagodbu
kontrole DataGrid



Prikaz samo određenih polja

Rijetko ćemo na stranici željeti pokazati sva polja koja smo upitom dohvatili. Ponekad ćemo moći jednostavno suziti upit nad bazom (što se preporučuje kada je ikako moguće), no ponekad ćemo "filtriranje" polja trebati napraviti na nivou kontrole. Da bismo to postigli, trebamo napraviti dvije stvari: parametru `AutoGenerateColumns` postaviti vrijednost na `false` te dodati sekciju `Columns`

unutar koje navodimo polja (stupce) koja želimo prikazati. Parametrom `DataField` određujemo ime polja, dok `HeaderText` definira tekst u zaglavlju.

```
<asp:DataGrid runat="server" AutoGenerateColumns="False">
  <Columns>
    <asp:BoundColumn DataField="OrderID"
      HeaderText="Broj narudžbe"></asp:BoundColumn>
    <asp:BoundColumn DataField="ShipName"
      HeaderText="Ime dostave"></asp:BoundColumn>
    <asp:BoundColumn DataField="ShipAddress"
      HeaderText="Adresa dostave"></asp:BoundColumn>
    <asp:BoundColumn DataField="ShipCity"
      HeaderText="Grad dostave"></asp:BoundColumn>
    <asp:BoundColumn DataField="OrderDate" HeaderText="Datum narudžbe"
      DataFormatString="{0:dd.MM.yyyy.}"></asp:BoundColumn>
    <asp:BoundColumn DataField="Freight" HeaderText="Težina"
      DataFormatString="{0:#.##} kg"
      ItemStyle-HorizontalAlign="Right"></asp:BoundColumn>
  </Columns>
</asp:datagrid>
```

Prilikom formatiranja datuma i vremena pazite – mm je oznaka za minute, a MM za mjesece.



U zadnjim smo stupcima koristili i dva nova svojstva. `DataFormatString` određuje format na koji će podatak biti prikazan, dok `ItemStyle-HorizontalAlign` definira vodoravnu poravnatost podataka u stupcu.

Izbor zapisa

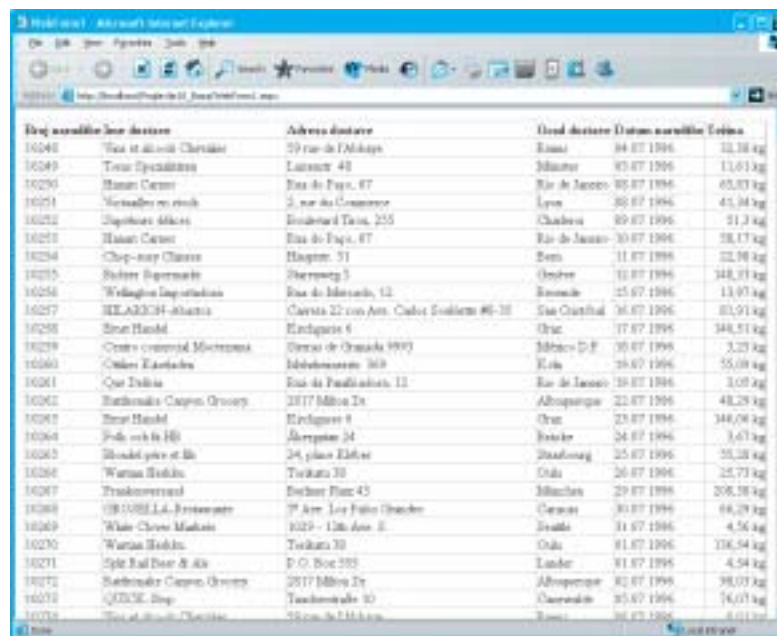
Sada ćemo u postojeću tablicu dodati još jedan stupac, no on neće sadržavati vrijednost iz baze podataka, već link. Stoga unutar bloka `<columns>` dodajte sljedeći redak:

```
<asp:ButtonColumn ButtonType="LinkButton" Text="Označi"
  CommandName="oznacavanje"></asp:ButtonColumn>
```

Kada posjetitelj klikne na spomenuti link, nastupit će događaj `ItemCommand`. Dakle, prebacit ćemo se u dizajnerski način, označiti kontrolu `DataGrid` i u prozoru `Properties` dvokliknuti na događaj `ItemCommand`. Otvorit će nam se metoda u kojoj trebamo dodati željenu funkcionalnost.

III. DIO: DIJELOVI .NET-A

Slika 10-28:
Prikaz samo nekih
polja iz baze

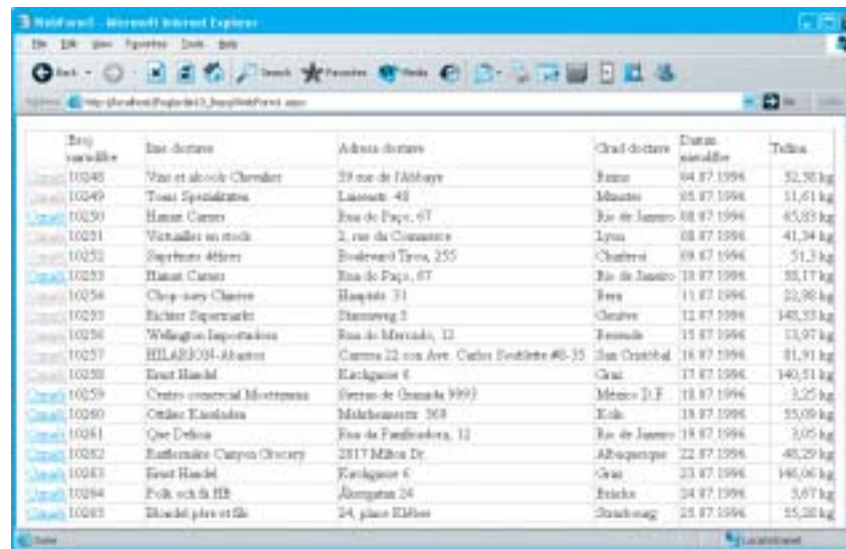


Objekat	Ime artikla	Adresa dostave	Cena artikla	Datum	Ukupna
10245	Vina et alcool Chardonnay	59 rue de l'Abbaye	32,50 kg	04.07.1996	32,50 kg
10249	Tous Spécialités	Lussigny 48	11,61 kg	05.07.1996	11,61 kg
10250	Hauts Cuvés	Rue de l'Église, 67	65,83 kg	06.07.1996	65,83 kg
10251	Vin de France	2, rue de Commerce	41,34 kg	08.07.1996	41,34 kg
10252	Superior Chardonnay	Boulevard Tiers, 255	51,3 kg	09.07.1996	51,3 kg
10253	Hauts Cuvés	Rue de l'Église, 67	35,17 kg	10.07.1996	35,17 kg
10254	Chapman Chardonnay	Haguenau 31	22,98 kg	11.07.1996	22,98 kg
10255	Superior Chardonnay	Stemmerweg 5	140,51 kg	12.07.1996	140,51 kg
10256	Wine Importation	Rue de l'Église, 12	13,97 kg	15.07.1996	13,97 kg
10257	HILARIOUX Chardonnay	Courtois 22 rue Ave. Charles Foullet 40-35	31,91 kg	16.07.1996	31,91 kg
10258	Superior Chardonnay	Stemmerweg 5	140,51 kg	17.07.1996	140,51 kg
10259	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10260	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10261	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10262	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10263	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10264	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10265	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10266	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10267	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10268	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10269	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10270	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10271	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10272	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10273	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10274	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg
10275	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg



Vrlo je bitno da se prilikom vraćanja stranica serveru (*postback*) ne izvršava metoda *Bindanje()*. Naime, radnje u njoj bi izbrisale događaj *ItemCommand* i vezana metoda ne bi bila izvršena.

Slika 10-29:
Novi stupac s
linkom



Objekat	Ime artikla	Adresa dostave	Cena artikla	Datum	Ukupna	Link
10245	Vina et alcool Chardonnay	59 rue de l'Abbaye	32,50 kg	04.07.1996	32,50 kg	Link
10249	Tous Spécialités	Lussigny 48	11,61 kg	05.07.1996	11,61 kg	Link
10250	Hauts Cuvés	Rue de l'Église, 67	65,83 kg	06.07.1996	65,83 kg	Link
10251	Vin de France	2, rue de Commerce	41,34 kg	08.07.1996	41,34 kg	Link
10252	Superior Chardonnay	Boulevard Tiers, 255	51,3 kg	09.07.1996	51,3 kg	Link
10253	Hauts Cuvés	Rue de l'Église, 67	35,17 kg	10.07.1996	35,17 kg	Link
10254	Chapman Chardonnay	Haguenau 31	22,98 kg	11.07.1996	22,98 kg	Link
10255	Superior Chardonnay	Stemmerweg 5	140,51 kg	12.07.1996	140,51 kg	Link
10256	Wine Importation	Rue de l'Église, 12	13,97 kg	15.07.1996	13,97 kg	Link
10257	HILARIOUX Chardonnay	Courtois 22 rue Ave. Charles Foullet 40-35	31,91 kg	16.07.1996	31,91 kg	Link
10258	Superior Chardonnay	Stemmerweg 5	140,51 kg	17.07.1996	140,51 kg	Link
10259	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10260	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10261	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10262	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10263	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10264	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10265	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10266	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10267	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10268	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10269	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10270	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10271	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10272	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10273	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10274	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link
10275	Centre commercial Montmartre	Centre de Commerce 999	3,25 kg	18.07.1996	3,25 kg	Link

10. POGLAVLJE: ASP.NET

Događaj ItemCommand nastupa prilikom bilo kakve aktivnosti unutar DataGrida. Među ostalima, to je i klik na jedan od linkova u našem stupcu Označi. U našem primjeru nema drugih događaja, no da ima, mogli bismo ih razlikovati prema parametru CommandName koji smo naveli u *tagu* <asp:ButtonColumn>. Koja je “komanda” nastupila, u funkciji provjeravamo ovako:

```
private void DataGrid1_ItemCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    if (e.CommandName == "oznacavanje")
    {
        DataGrid1.SelectedIndex = e.Item.ItemIndex;
    }
}
```

Kroz parametar *e* možemo otkriti u kojem se retku nalazio link koji je pritisnut. U gornjem primjeru tu vrijednost pridružujemo svojstvu DataGrid1.SelectedIndex, što će rezultirati označavanjem tog retka.

Kako smo, radi jednostavnosti primjera, isključili stilove, označeni redak vjerojatno nećete uočiti. Da biste ga mogli uočiti, treba definirati neko svojstvo unutar stila SelectedItemStyle.



Kraj	Naziv	Adresa	Grad	Datum	Težina
Češka	18246	Vina et al. Chardonnay	Brno	14.03.1996	22.18 kg
Češka	18249	Tento Specialites	Liberec	05.03.1996	11.81 kg
Češka	18250	Huani Cacao	Rio de Janeiro	05.03.1996	45.87 kg
Češka	18251	Actualites re stock	Lyon	08.03.1996	41.34 kg
Češka	18252	Superior Affairs	Charleroi	09.03.1996	11.13 kg
Češka	18255	Huani Cacao	Rio de Janeiro	06.03.1996	38.17 kg
Češka	18256	Chap-rang Chardonnay	Brno	11.03.1996	22.88 kg
Češka	18257	Encher Supermarket	Oslo	12.03.1996	149.33 kg
Češka	18258	Willington Exportadora	Brno	13.03.1996	13.37 kg
Češka	18257	HEARST Atlanta	San Cristobal	16.07.1996	81.81 kg
Češka	18258	Encher Supermarket	Oslo	17.03.1996	149.33 kg
Češka	18259	Cuervo Comercial Mexicana	Mexico DF	08.03.1996	3.23 kg
Češka	18260	Ordes Evaristo	Brno	18.03.1996	22.88 kg
Češka	18261	Don Tefino	Brno	19.03.1996	3.23 kg

Slika 10-30:
Klikom na link
“Označi” redak
se – označio.

III. DIO: DIJELOVI .NET-A

Ovo je najjednostavnije što smo mogli učiniti s tim poljem. Ipak, u praksi ćete, primjerice, željeti posjetitelja preusmjeriti na stranicu na kojoj će dobiti detaljnije informacije o toj narudžbi. U tom bi slučaju naredba u metodi izgledala, recimo, ovako:

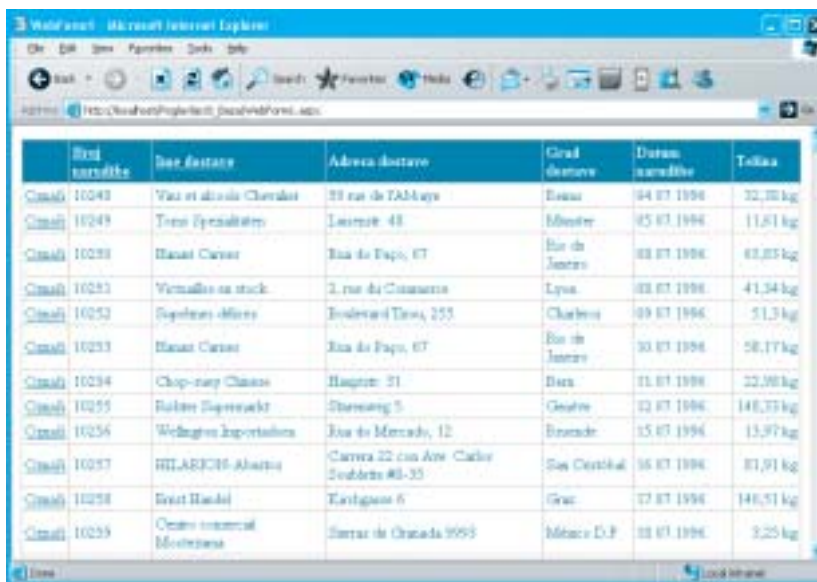
```
Response.Redirect("WebForm2.aspx?id=" + e.Item.Cells[1].Text);
```

Kao što vidite, posjetitelj bi bio preusmjeren na drugu web-formu s parametrom koji sadrži tekst iz drugog (indeksi počinju od broja 0!) stupca. U našem je primjeru to stupac s identifikacijskim brojem narudžbe, što je točno onaj parametar koji se prosljeđuje u ovakvim situacijama.

Sortiranje zapisa

Vrlo je korisno omogućiti korisniku da prilikom pregledavanja sadržaja baze sortira određene stupce. Za posjetitelja je funkcionalnost jednostavna – klikne na ime stupca po kojem želi sortirati i očekuje rezultat. Što treba napraviti da bi mu se to omogućilo?

Slika 10-31:
Tablica s mogućnošću sortiranja po drugoj i trećoj koloni



	Red kategorije	Ime dostave	Adresa dostave	Grad dostave	Datum isporuke	Tražila
Caroli	10243	Vozni staklo Chevrolet	33 rue de l'Abbaye	Evreux	04.07.1996	32,30 kg
Caroli	10249	Tona Extraditox	Lezard 48	Montreuil	05.07.1996	11,61 kg
Caroli	10258	Banast Carroz	Rue de l'Épée, 67	Rue de Jenners	08.07.1996	88,85 kg
Caroli	10252	Vozni staklo na truck	2 rue de Commerce	Lyon	08.07.1996	41,34 kg
Caroli	10252	Superflex Offroad	Bolevard Taux, 255	Charleville	09.07.1996	51,3 kg
Caroli	10253	Banast Carroz	Rue de l'Épée, 67	Rue de Jenners	30.07.1996	58,17 kg
Caroli	10254	Chop-rap Chassis	Hauptstr. 51	Bielefeld	01.07.1996	22,98 kg
Caroli	10255	Roller Supersport	Shenwang 5	Genève	02.07.1996	146,33 kg
Caroli	10254	Wolfgang Importations	Rue de Mercado, 12	Buenos Aires	05.07.1996	13,97 kg
Caroli	10257	WILABECH-Abastus	Carrera 22 con Ave. Carlos Sobrido #8-35	San Cristóbal	06.07.1996	81,91 kg
Caroli	10258	Ernst Haeckel	Karlsgasse 6	Graz	07.07.1996	146,51 kg
Caroli	10259	Casco integral Montesa	Barral de Granada 9093	México D.F.	08.07.1996	3,25 kg

U HTML-kôd kontrole treba dodati sljedeće:

```
<asp:DataGrid ... AllowSorting="True"> ...
```

Nadalje, kod definicije svake kolone po kojoj želimo uključiti sortiranje (unutar sekcije <Columns>) treba dodati ovaj parametar kojem ćemo kao vrijednost najčešće upisati ime polja u bazi:

```
<asp:BoundColumn DataField="OrderID" HeaderText="Broj narudžbe"
SortExpression="OrderID"></asp:BoundColumn>
```

To će nam osigurati promjene u sučelju koje su potrebne za sortiranje, a još nam preostaje napisati dva komadića kôda koji će samo sortiranje odraditi. Naime, prilikom sortiranja treba izmijeniti SQL-upit koji šaljemo bazi i zatim *rebindati* kontrolu. Stoga bi trebalo modificirati funkciju Bindanje() da, ovisno o parametru, doda parametar sortiranja u SQL-upit.

Mi nećemo brisati postojeću funkciju koja ne prima parametar (tako da ne moramo mijenjati dio kôda koji je koristi), nego ćemo napraviti još jednu preopterećenu inačicu s tim parametrom. Znači, uz postojeću funkciju Bindanje(), treba dodati sljedeće:

```
private void Bindanje(string sortiranje)
{
    SqlConnection sqlConn = new
        SqlConnection(ConfigurationSettings.AppSettings["connstr"]);
    SqlDataAdapter sqlComm = new SqlDataAdapter("SELECT * FROM Orders ORDER BY " +
        sortiranje, sqlConn);

    DataSet ds = new DataSet();
    sqlComm.Fill(ds);

    DataGrid1.DataSource = ds;
    DataGrid1.DataBind();
}
```

Evo i kako ćemo pozvati tu funkciju:

```
private void DataGrid1_SortCommand(object source,
    System.Web.UI.WebControls.DataGridSortCommandEventArgs e)
{
    Bindanje(e.SortExpression);
}
```

Naime, svojstvo e.SortExpression sadrži vrijednost koju smo definirali u HTML-kodu istoimenim parametrom. Drugim riječima, da smo tamo taj parametar definirali ovako:

```
SortExpression="OrderID DESC"
```

... narudžbe bi bile sortirane od one s najvećim brojem do one s najmanjim.

III. DIO: DIJELOVI .NET-A

Pregled po stranicama

Ako se u bazi nalaze veće količine podataka, njihov prikaz na jednoj stranici rezultirao bi dugim učitavanjem i velikom nepreglednošću. Zato postoji mogućnost pregledavanja podataka po stranicama. U ASP.NET-u je to izuzetno jednostavno izvesti.

Kao i obično, za početak su potrebne određene modifikacije u HTML-kodu:

```
<asp:DataGrid ... AllowPaging="True" PageSize="15"> ...
```

Prvi parametar uključuje prikaz po stranicama, dok drugi određuje veličinu jedne stranice (broj zapisa po stranici).

Slika 10-32:
Prikaz podataka
iz baze po stranicama

ID	Ime osobe	Adresa osobe	Grad osobe	Datum rođenja	Težina
00046	Van et Alencio Claudio	39 rue de L'Alouette	Reims	04.07.1936	32,38 kg
00049	Timon Dymakiewicz	Laurens 48	Münster	05.07.1936	71,61 kg
00050	Manuel Casco	Rue de Fays, 67	Eau de France	08.07.1936	65,63 kg
00051	Yves-Marie Le Goff	2, rue du Commerce	Lyon	08.07.1936	41,34 kg
00052	Clémentine Wilson	Boisrond Tiers, 258	Chambéry	08.07.1936	51,3 kg
00053	Emile Carter	Rue de Fays, 67	Eau de France	18.07.1936	58,17 kg
00054	Clayton Charles	Hauptstr. 31	Reims	13.07.1936	22,08 kg
00055	André Suprenant	Stamweg 5	Genève	13.07.1936	148,39 kg
00056	William Suprenant	Rue de Metz, 12	Reims	13.07.1936	15,97 kg
00057	ROLAND ALBERT	Carrée 23 rue Jean-Claude 80-35	San Cristóbal	18.07.1936	81,91 kg
00058	Eric Huetel	Hariguen 6	Cher	17.07.1936	140,51 kg
00059	Cécile Marcelle Mouton	Servat de Gascogne 991	Ménil II F	18.07.1936	3,25 kg
00060	Oscar Eustache	Milchstrasse 369	Ede	19.07.1936	33,09 kg
00061	Que Delia	Rue de Poulard, 12	Eau de France	19.07.1936	3,05 kg
00062	Esther Marie Chazotte	2817 Millard Dr	Albuquerque	22.07.1936	48,29 kg

Naravno, i tu nam treba komadić kôda koji će spomenutu funkcionalnost odraditi. Ovoga se puta većemo uz događaj `PageIndexChanged`:

```
private void DataGrid1_PageIndexChanged(object source,
    System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
{
    DataGrid1.CurrentPageIndex = e.NewPageIndex;
    Bindanje();
}
```


Želite li umjesto brojeva stranica na dnu koristiti izraze prethodna i sljedeća stranica, dodajte unutar *tagova* kontrole sljedeći izraz:

```
<PagerStyle Mode="NextPrev" PrevPageText="prije"
NextPageText="poslije"></PagerStyle>
```

Imajte na umu da ovaj način prikazivanja podataka po stranicama nije i najbolji. Naime, stranica iz baze dobiva sve podatke, uključujući i one koji se ne prikazuju, što kod velikih količina podataka može biti relativno sporo.



Prilagodljivi stupci

Dosad smo u primjerima svako od polja iz baze trpali u zaseban stupac. To je često zgodno rješenje, no ponekad imamo potrebu, primjerice, spojiti više polja u isti stupac. U tome će nam pomoći prilagodljivi stupci (engl. *template column*).

Te stupce definiramo na istom mjestu gdje smo definirali ostale vrste stupaca, u sekciji `<Columns>`:

```
<Columns>
  <asp:ButtonColumn ... ></asp:ButtonColumn>
  <asp:BoundColumn ... ></asp:BoundColumn>
  <asp:TemplateColumn ...>
    <!-- definicija prilagodljivog stupca -->
  </asp:TemplateColumn>
</Columns>
```

Evo kako bi izgledao kôd za prilagodljivi stupac koji možete vidjeti na slici 10-33:

```
<asp:TemplateColumn>
  <ItemTemplate>
    <b><%# DataBinder.Eval(Container.DataItem, "ShipName") %></b>,
    <%# DataBinder.Eval(Container.DataItem, "ShipAddress") %>,
    <%# DataBinder.Eval(Container.DataItem, "ShipCity") %>
  </ItemTemplate>
</asp:TemplateColumn>
```

Predložak za stupac može sadržavati bilo koje HTML-elemente, pa čak i web-kontrole, i mora biti definiran između *tagova* `<ItemTemplate>` i `</ItemTemplate>`. Unutar predloška na pokazani način možemo pristupiti i poljima u bazi te na taj način napraviti bilo kakav prikaz podataka.

III. DIO: DIJELOVI .NET-A

Slika 10-33:
Povezivanje više
polja iz baze u
isti stupac
pomoću pri-
lagodljivih stu-
paca

Idag	Naziv	Datum	Težina
10240	Vino et deinde Chardonnay, 50 rue de l'Abbaye, Brou	04.07.1996	32.38 kg
10241	Tuna Spontakritika, Lussac 48, Mâcon	05.07.1996	17.83 kg
10248	Biscuit Carroz, Rue de Paix, 67, Rue de l'Arbre	08.07.1996	45.83 kg
10250	Vermicelles en stock, 2, rue de Commerce, Lyon	08.07.1996	41.34 kg
10252	Supplément de l'ère, Boulevard Tiers, 235, Châtillon	08.07.1996	51.3 kg
10253	Biscuit Carroz, Rue de Paix, 67, Rue de l'Arbre	10.07.1996	58.17 kg
10254	Chap-easy Châlon, Hapier 31, Tera	11.07.1996	32.98 kg
10255	Biskvit Supermarket, Oberberg 3, Ostfrie	12.07.1996	948.33 kg
10256	Wellington Lagerstaden, Rue de Mécène, 12, Bessende	15.07.1996	13.97 kg
10257	BILLARDON-Abstrak, Carrera 22 rue des Carls Soubiers 40-35, San Cristóbal	16.07.1996	31.91 kg
10258	Biscuit Handel, Hapier 31, Tera	17.07.1996	540.51 kg
10259	Centre commercial Mactonema, Centre de Grande 9901, Mâcon D.F.	18.07.1996	3.25 kg
10260	Oreilles Elitistaden, Mactonema 349, Röh	19.07.1996	35.89 kg
10261	Que Delicia, Rue de l'Industrie, 12, Rue de l'Arbre	19.07.1996	3.85 kg
10262	Bardoune Campus Gracery, 2017 Mâcon D.F., Abstrak	22.07.1996	48.29 kg

Sada kad smo upoznali prilagodljive stupce, mogli bismo napisati nešto efikasniji način preusmjerenja posjetitelja na novu stranicu. Sjetite se – nedavno smo tu stvar riješili pomoću stupca tipa ButtonColumn koji je radio *postback* te tamo radio Response.Redirect na željenu stranicu. Evo kako to napraviti pomoću prilagodljivog stupca jednostavnije i brže:

```
<asp:TemplateColumn>
  <ItemTemplate>
    <a href="WebForm2.aspx?id=<%=# DataBinder.Eval(Container.DataItem, "OrderID")
    %>">Detaljnije...</a>
  </ItemTemplate>
</asp:TemplateColumn>
```



Jednom kada napišete izraz poput ovoga, Visual Studio vam neće dozvoliti vratiti se u dizajnerski pogled stranice. Ne pitajte zašto, jednostavno neće. Problem rade dvostruki navodnici u izrazu, a tome možete doskočiti tako da stvar napišete na sljedeći način (uočite jednostruke navodnike na dva mjesta!):

```
<a href='WebForm2.aspx?id=<%=# DataBinder.Eval(Container.DataItem,
"OrderID") %>'>Detaljnije...</a>
```

Ovaj problem se javlja samo kada navodnici obuhvaćaju referencu na polje u bazi.

Uređivanje zapisa

Vjerojatno se nećete iznenaditi kad vam kažemo da DataGrid omogućava i uređivanje zapisa. Postizanje ove funkcionalnosti slično je promjeni označenog retka na stranici, samo što dodajemo stupac tipa EditCommandColumn. Ovako:

```
<asp:EditCommandColumn EditText="Uredi" ButtonType="PushButton" UpdateText="Snimi"
CancelText="Odustani" />
```

Uočite da smo ovoga puta kao ButtonType stavili vrijednost PushButton (za razliku od LinkButton). To znači da će nam se u toj koloni pojaviti gumb, a ne link, iako bi oboje imali istu funkcionalnost.



I ovdje nam treba vezanje uz događaj, sad se događaj zove EditCommand, a pripadajuća funkcija mu izgleda ovako:

```
private void DataGrid1_EditCommand(object source,
    System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    DataGrid1.EditItemIndex = e.Item.ItemIndex;
    Bindanje();
}
```

Kada postavimo ovu vrijednost svojstvu EditItemIndex, redak u kojem smo kliknuli na gumb “Uredi” moći ćemo uređivati.

Poljima za koja ne želimo da budu uređivana (primjerice, identifikacijska) treba dodati parametar ReadOnly:

```
<asp:BoundColumn DataField="OrderID" SortExpression="OrderID" HeaderText="Broj
narudžbe" ReadOnly="True"></asp:BoundColumn>
```

Nakon što pokrenemo editiranje nekog retka, u njemu će se pojaviti dva nova gumba. Tekstove na tim gumbima definirali smo još na početku (parametri UpdateText i CancelText), a sada im treba napisati funkcionalnost.

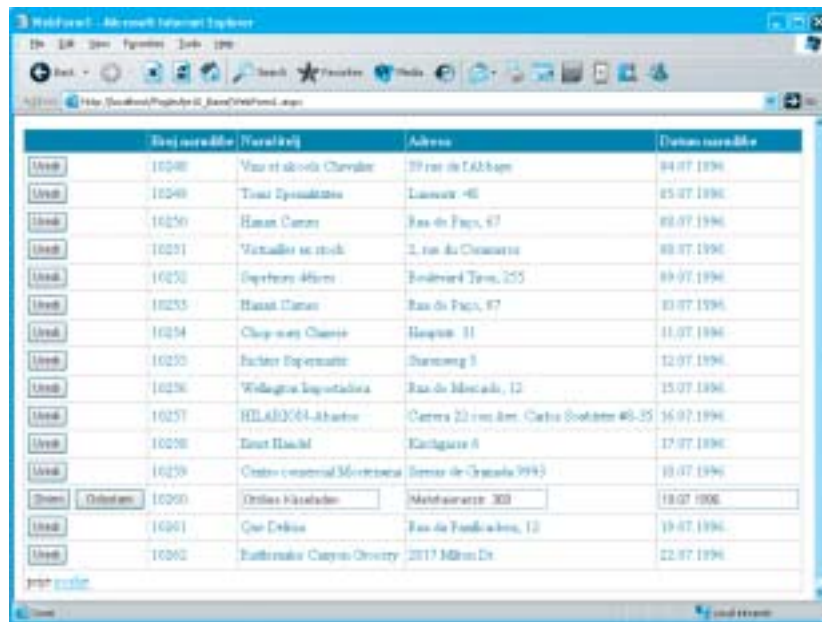
Počnimo od jednostavnijega – gumb Cancel (odnosno, u našem primjeru, Odustani) ima zadatak isključiti uređivanje retka bez snimanja promjena. Stoga svojstvu EditItemIndex treba pridružiti

III. DIO: DIJELOVI .NET-A

vrijednost -1 koja znači da se niti jedan redak ne uređuje. Kako prilikom klika na gumb Cancel nastaje događaj CancelCommand, sljedeću funkciju vezemo uz njega:

```
private void DataGrid1_CancelCommand(object source,
    System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    DataGrid1.EditItemIndex = -1;
    Bindanje();
}
```

Slika 10-34:
Uređivanje
zapisa u
DataGridu



S gumbom Update koji uključuje i snimanje promjena imamo nešto više posla. Sljedeću funkciju vezat ćemo uz događaj UpdateCommand (funkciju ćemo sjeći komentarima, no to je sve jedna funkcija):

```
private void DataGrid1_UpdateCommand(object source,
    System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    // spremanje vrijednosti iz polja obrasca u varijable
    int OrderID = Convert.ToInt32(e.Item.Cells[1].Text);
```

10. POGLAVLJE: ASP.NET

U varijablu `OrderID` spremamo vrijednost iz drugog stupca (index 1). Kako smo to polje označili kao `ReadOnly`, u njemu se nalazi samo identifikacijski broj zapisa kojem pristupamo svojstvom `Text`. Naravno, kako se radi o broju, konvertiramo ga u *int*.

```
TextBox ShipNameText = (TextBox)e.Item.Cells[2].Controls[0];
TextBox ShipAddressText = (TextBox)e.Item.Cells[3].Controls[0];
TextBox OrderDateText = (TextBox)e.Item.Cells[4].Controls[0];

string ShipName = ShipNameText.Text;
string ShipAddress = ShipAddressText.Text;
DateTime OrderDate = DateTime.ParseExact(OrderDateText.Text, "dd.MM.yyyy.",
    null);
```

Kod ostalih polja stvar je nešto složenija. Naime, unutar stupca ne nalazi se čisti tekst, već kontrola tipa `TextBox`. Kako je ona jedina kontrola u stupcu, pristupamo joj preko kolekcije kontrola uz indeks 0 (`Controls[0]`). Mi smo sigurni da se radi o kontroli tipa `TextBox`, no kompajler nije, pa radimo *castanje*. Zatim je lako doći do vrijednosti unutar kontrole (svojstvo `Text`). Kako je svojstvo `Text` tipa *string*, kod datuma smo morali raditi konverziju pomoću metode `ParseExact`.

```
// definiranje SQL-upita za Update
string sqlUpit = "UPDATE Orders SET ShipName = @ShipName, ShipAddress =
    @ShipAddress, OrderDate = @OrderDate WHERE OrderID = @OrderID";

// stvaranje konekcije prema bazi
SqlConnection sqlConn = new
    SqlConnection(ConfigurationSettings.AppSettings["connstr"]);

// otvaranje konekcije prema bazi
sqlConn.Open();

// kreiranje SQL-naredbe
SqlCommand sqlComm = new SqlCommand(sqlUpit, sqlConn);

// dodavanje parametara
sqlComm.Parameters.Add("@OrderID", OrderID);
sqlComm.Parameters.Add("@ShipName", ShipName);
sqlComm.Parameters.Add("@ShipAddress", ShipAddress);
sqlComm.Parameters.Add("@OrderDate", OrderDate);

// izvršavanje SQL-upita
```

III. DIO: DIJELOVI .NET-A

```
sqlComm.ExecuteNonQuery();

// zatvaranje konekcije prema bazi
sqlConn.Close();

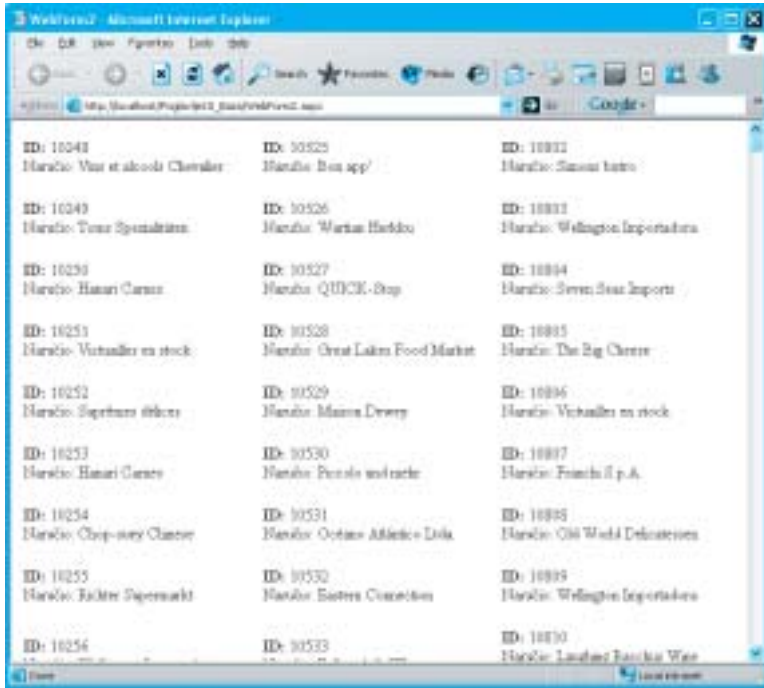
// isključivanje uređivanja retka
DataGrid1.EditItemIndex = -1;
Bindanje();
}
```

Ostatak kôda već smo susreli u prošlom poglavlju, pa nećemo ulaziti u detalje.



Gornji primjer nije otporan na gluposti koje posjetitelj može upisati u polja. Primjerice, može upisati tekst duži od predviđenog u bazi podataka ili pak unijeti krivi format datuma. Drugim riječima, dobro bi došle provjere valjanosti upisanih podataka.

Slika 10-35:
Prikaz podataka u tri stupca pomoću kontrole DataList



Oprez! Uljezi u upitu!

Gornji smo primjer mogli riješiti i malo drugačije. Primjerice, umjesto korištenja parametara u SQL-upitu, mogli smo napisati nešto poput ovoga:

```
string sqlUpit = "UPDATE Orders
SET ShipName = '" + ShipName +
'' WHERE OrderID = " + OrderID;
```

Međutim, takvo prosljeđivanje parametara SQL-upitu može biti kobno! O čemu se radi? Uzmi-mo za primjer da je posjetitelj naših stranica upisao u tekstualno polje prilikom editiranja sljedeću vrijednost:

```
'; DELETE FROM Orders; UPDATE
Orders SET ShipName = '
```

Vrlo naivno, računalo taj unos smatra legitimim i ubacuje ga u naš SQL-upit, koji ispada

ovakav:

```
UPDATE Orders SET ShipName = '';
DELETE FROM Orders; UPDATE
Orders SET ShipName = '' WHERE
OrderID = 12345
```

Zlobni posjetitelj je trikom naš upit pretvorio u tri upita od kojih je jedan od njih koban jer će izbrisati sve zapise tablice na koju se odnosi!

Ovakav napad naziva se *SQL injection*, a može ga se izbjeći jednostavnom konverzijom svakog jednostrukog navodnika koji je posjetitelj unio u dva jednostruka navodnika. Međutim, puno je bolji način koji smo pokazali u primjeru uz tekst – parametri se po službenoj dužnosti brinu o takvim, ali i brojnim drugim sitnicama.

Kontrola DataList

Najkraće rečeno, DataList je kontrola većih mogućnosti prilagodbe od DataGrida, no zato neke funkcionalnosti traže više podešavanja i programiranja.

Evo primjera kojem ćemo prikazati neka polja tablice Orders kao na slici 10-35:

```
<asp:DataList id="DataList1" runat="server" RepeatColumns="3">
  <ItemTemplate>
    <b>ID:</b> <%=# DataBinder.Eval(Container.DataItem, "OrderID") %>
    <br />
    Naručio: <%=# DataBinder.Eval(Container.DataItem, "ShipName") %>
    <p />
```

III. DIO: DIJELOVI .NET-A

```
<ItemTemplate>
</asp:DataList>
```



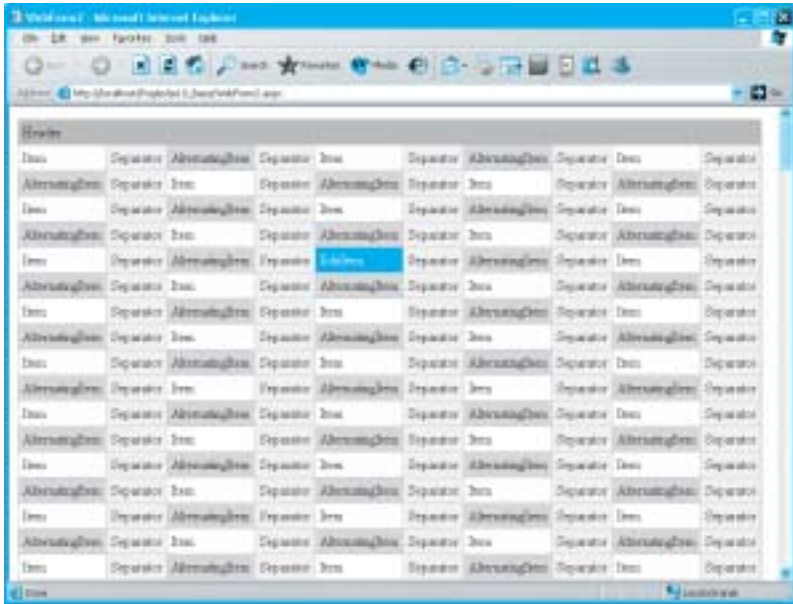
Da bi se kontrola prilikom izvršavanja stranice popunila podacima, potrebno je napraviti povezivanje s bazom na isti način koji smo to radili za kontrolu DataGrid metodom Bindanje().

Osim osnovnog predloška za zapis, kontrola DataList može sadržavati i neke druge predloške.

Predložak AlternatingItemTemplate, ukoliko je definiran, koristit će se za prikaz svakog drugog zapisa. U kombinaciji s ItemTemplateom možete postići da se zapisi naizmjenice prikazuju koristeći jedan pa drugi predložak.

Pomoću predložaka HeaderTemplate i FooterTemplate definiramo zaglavlje i podnožje tablice s podacima. Imajte na umu da se oni neće vidjeti na stranici, ako su isključena svojstva ShowHeader odnosno ShowFooter.

Slika 10-36:
Demonstracija
predložaka u
kontroli DataList

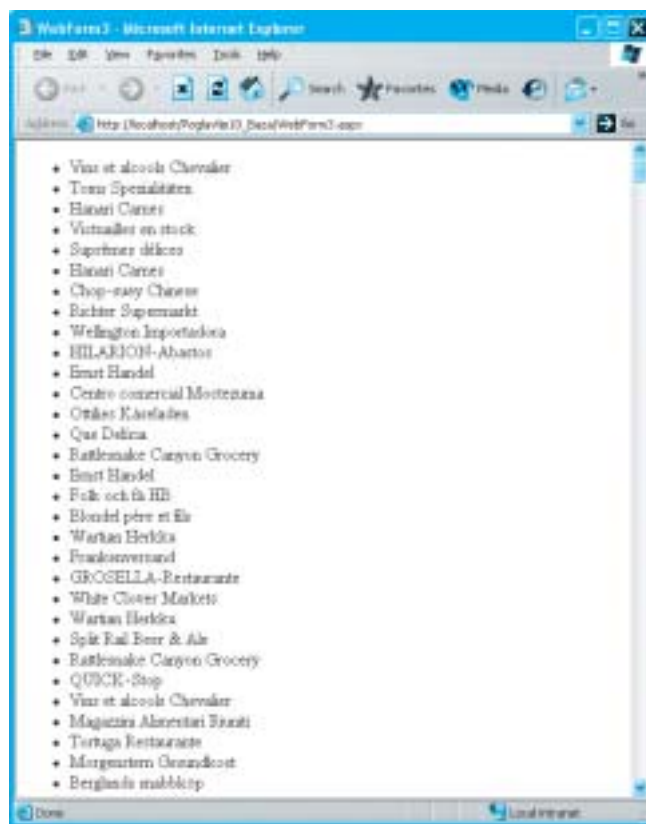


EditItemTemplate služi za definiranje predloška za zapis koji se uređuje (što nam pruža puno više mogućnosti od DataGrida), dok je SeparatorTemplate predložak za polje koje će se pojavljivati između zapisa.

Kontrola Repeater

Za one željne maksimalne kontrole nad prikazom podataka postoji kontrola Repeater. Ona nema nikakvih kozmetičkih svojstava, stilova za predloške niti zapise smješta u tablice ili *tagove* `` (ovo potonje može napraviti samo kontrola DataList ako joj promijenite svojstvo RepeatLayout). Kod nje je prikaz podataka u potpunosti u rukama programera.

Kontrola Repeater i DataList rade pomoću istih predložaka, no s nešto ograničenijim brojem mogućnosti.



Slika 10-37:
Prikaz podataka iz baze
pomoću kontrole Repeater

Evo primjera u kojem ćemo zapise iz baze prikazati kao na slici 10-37 (niti tu nemojte zaboraviti metodu Bindanje):

```
<asp:Repeater id="Repeater1" runat="server">
    <HeaderTemplate>
```

III. DIO: DIJELOVI .NET-A

```
<ul>
</HeaderTemplate>
<ItemTemplate>
    <li><%# DataBinder.Eval(Container.DataItem, "ShipName")%></li>
</ItemTemplate>
<FooterTemplate>
    </ul>
</FooterTemplate>
</asp:Repeater>
```

Na milost posjetiteljima

Na kraju razvoja svake web-aplikacije doći ćete na zadnju stepenicu – prebacivanje aplikacije na poslužitelj. To će uglavnom biti izuzetno jednostavan posao.

Prije prebacivanja treba pripremiti poslužitelj – otvoriti novi web-síte ili kreirati novu web-aplikaciju, podesiti DNS i eventualno IP-adrese.

Zatim ide prebacivanje web-aplikacije na poslužitelj, što se svodi na kopiranje datoteka. Trebat ćete prebaciti sve konfiguracijske datoteke, datoteke s HTML-kôdom (“.aspx”, “.ascx”, “.asax”...) te mapu “bin”. U njoj se nalazi datoteka s nastavkom “.dll” u kojoj je kompajlirani kôd iz svih datoteka s pozadinskim kôdom (“.aspx.cs”, “.ascx.cs”, “.asax.cs”...) tako da njih nije potrebno prebacivati na poslužitelj.

Prije prebacivanja DLL datoteke na poslužitelj treba je kompajlirati u načinu *release* (vidi kraj osmog poglavlja).

Na produkcijskom serveru ćete vjerojatno koristiti i neki drugi izvor baze podataka pa će vjerojatno biti potrebno prilagoditi (zato je vrlo dobra praksa smještati ga u konfiguracijsku datoteku).

Dodatne komponente koje ste eventualno koristili također se nalaze u DLL-datotekama u mapi “bin”. I njih je potrebno kopirati na poslužitelj.

No, naravno, osnovni uvjet i dalje postoji – poslužitelj mora imati instaliran .NET Framework.