24

Praktičan primer – IBuyAdventure.NET

U prethodnim poglavljima ove knjige bilo je reči o pojedinim karakteristikama ASP.NET-a, kao što su Web kontrole, povezivanje podataka, konfigurisanje i bezbednost. Iako su moćne, ove karakteristike su poput delića slagalice: konačni cilj je shvatiti kako ti delići mogu biti povezani u veću sliku.

U ovom poglavlju ćemo pokazati kako se ASP.NET koristi za stvaranje jednostavne aplikacije za elektronsku trgovinu n-sloja koja je skalabilna, a koja se ipak jednostavno kodira i razume. Osim toga govorićemo i prikazati i:

- Kako da na .NET platformi projektujete i napišete aplikaciju za elektronsku trgovinu n-sloja.
- Korišćenje provere identiteta zasnovane na HTML obrascima da bi aplikacija za elektronsku trgovinu bila bezbedna.
- Razmatranje skalabilnosti kod stvaranja lokacije za elektronsku trgovinu koja mora prerasti u Web farmu.
- Korišćenje poslovnih objekata za enkapsuliranje poslovne logike i pristup podacima.
- Korišćenje koda u pozadini za zajedničku logiku stranice.

Pregled aplikacije

Oni koji mogu da se sete ASP-a 2.0 prisetiće se da je on isporučen sa primerom aplikacije pod nazivom **AdventureWorks**. Ova aplikacija je omogućavala da kupujete planinarsku opremu za ekstremne uslove. AdventureWorks je prikazala sve osnovne mogućnosti tipične aplikacije za elektronsku trgovinu, uključujući pregled proizvoda i način izbora, registraciju korisnika i korpu za kupovinu. U ovom poglavlju ćemo koristiti ASP.NET da bismo AdventureWorks proširili u novu aplikaciju pod nazivom **IBuyAdventure.NET** ili skraćeno **IBA**. U preostalom delu ovog poglavlja ćemo dati pregled koda i funkcionalnosti ove aplikacije i govoriti o nekim od odluka o projektu koje smo doneli tokom njenog stvaranja.

Sve datoteke za aplikaciju IBuyAdventure.NET se mogu preuzeti sa Web lokacije Wroxa http://www.wrox.com uključujući i uputstva za podešavanje i instaliranje.

IBuyAdventure.NET (IBA.NET)

Ako ranije niste videli ili koristili staru ASP aplikaciju AdventureWorks, ne brinite. O aplikaciji IBA.NET ćemo govoriti od početka i preneti mali podskup mogućnosti iz prvobitne aplikacije. Ovaj pristup omogućava da pokažemo kako se ASP.NET koristi za stvaranje aplikacija za elektronsku trgovinu iz realnog života, ali bez suvišnih mogućnosti specifičnih za aplikaciju koje bi samo umanjile značaj ovog poglavlja.

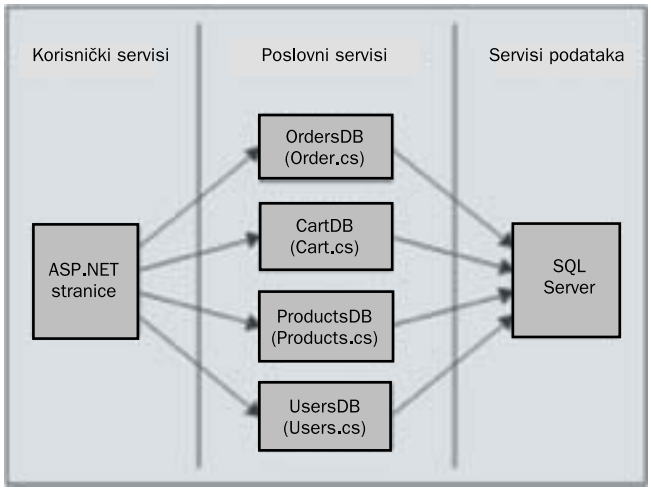
Ako imate pristup prvobitnoj aplikaciji AdventureWorks preporučujemo da uporedite prvobitni ASP kôd i ASP.NET kôd iz ovog poglavlja. Sekcije kao što su korpa za kupovinu i kôd za raspored na strani u stvari pokazuju moć i jednostavnost ASP.NET-a.

Ciljna grupa

Aplikacija IBA.NET je namenjena da entuzijastima planinarima omogući da preko Interneta iz svoje kuće ili šatora provere i kupe najnoviju planinarsku opremu. Aplikacija će omogućiti klijentima da pogledaju različite proizvode koji su grupisani prema kategoriji (čizme, pantalone, šatori itd.) i da dodaju stavke svojoj kupovnoj kartici u bilo kom trenutku. Aplikacija koristi JIT registraciju, tako da klijenti mogu da popune svoju kupovnu karticu proizvodima a da ne moraju da se registruju ili prijave dok zaista ne nastave sa proverom. Ovaj pristup je prilično uobičajen na većini velikih lokacija kao što je Amazon.com i danas zaista predstavlja neophodno svojstvo za svaku od aplikacija elektronske trgovine.

Skalabilnost – Platforma za Web rešenja

Aplikacija IBA.NET treba da bude skalabilna i u mogućnosti da podržava stotine, pa čak i hiljade istovremenih korisnika. Da bi ovaj cilj bio postignut aplikacija je projektovana u skladu sa uputstvima koje određuje **Microsoft Web Solution Platform** (http://microsoft.com/business/products/webplatform). Aplikacija zato usvaja arhitekturu *n*-sloja kao što je opisano u Windows DNA, prema kojoj je aplikacija podeljena na nekoliko slojeva za prikazivanje, poslovnu logiku i podatke:



Kada se prihvati pristup sa n-slojem, aplikacija IBuyAdventure.NET može lako da se uvede na jednom računaru ili više njih (jedan ili više fizičkih slojeva). Trebalo bi da bude moguće da se pomoću ove arhitekture uvedu ASP.NET stranice, .NET komponente i SQL Server na servere koji su njima namenjeni. Sve ASP.NET stranice u aplikaciji IBuyAdventure.NET pristupaju krajnjoj bazi podataka pomoću skupa .NET komponenti koje nameću poslovnu logiku i enkapsuliraju bazu podataka na kojoj se zasnivaju. Ove komponente su prilično uske i za većinu delova samo predstavljaju omotače za seriju ADO.NET rutina, koje su slične onim o kojima je bilo reči u poglavlju 8, i koje vraćaju `DataSet` iz metoda koji se pozivaju za uzimanje podataka.

Projektovanje skalabilnosti preduzeća

Da bila zadovoljena skalabilnost na nivou preduzeća (korišćenje Web farmi) aplikacija *ne* koristi ASP.NET stanje na nivou sesije. Sva stanja su ili pohranjena na strani klijenta u okviru skrivenih polja ili u krajnjoj bazi podataka SQL servera. Kada korisnik prvi put poseti lokaciju, ASP.NET ID sesije se koristi da pronađe korisnika i sve stavke za kupovinu koje on dodaje svojoj kupovnoj kartici. Ova informacija se pohranjuje u bazi podataka a ID sesije se koristi kao primarni ključ za identifikovanje korisnika. Iako se ovo obično ne preporučuje, tranzijentna priroda podataka kupovne kartice je prilično bezbedna. Ili možemo da generišemo GUID i koristimo ga da bismo otkrili anonimne korisnike. Nakon registracije i prijavljivanja, sve reference za ASP.NET ID sesije su pohranjene u bazi podataka i ažurirane pod imenom korisnika.

Korišćenjem ovog pristupa aplikacija može jednostavno biti uvedena u Web farmu a da nije potrebno da se koristi stanje sesije koje je specifično za pojedine servere. Svaki Web server koji prima zahtev može da koristi ID sesije ili korisničko ime (koje je obezbeđeno tokom provere identiteta zasnovane na obrascima) da bi u centralnoj bazi podataka potražio podatke o korisnicima i stavkama na kartici.

„Sticky Sessions" je termin koji se koristi da opiše ASP/ASP.NET korisničke sesije koje uvek moraju biti preusmerene na isti čeoni Web server u Web farmi. Obično su „sticky sessions" potrebne u aplikacijama koje zavise od stanja koje je pohranjeno u objektu `Session`. To je obično loša odluka kod projektovanja, jer ako računar na kome se nalazi „sticky sessions" padne ili mora biti restartovan, sva stanja sesije su izgubljena a korisnik u stvari mora da počne od početka.

Korišćenje baze podataka za pohranjivanje stanja sesije (kao što je kupovna kartica) daje određenu višeznačnost aplikaciji ali je čini prilagodljivom u slučaju grešaka. Ako na jednom ili dva Web servera dođe do greške za vreme korisničkog zahteva, neki drugi server može da obavi posao tako da informacije neće biti izgubljene. Kada koristimo kupovnu karticu zasnovanu na sesiji, ako dođe do greške na računaru na kome se sesija nalazi, naši korisnici će izgubiti sve što je stvoreno za vreme njihove sesije.

ASP.NET je projektovan na osnovu principa da na serverima *dolazi* do greške i da aplikacije/komponente *mogu* da se izgube i da padaju s vremena na vreme. Projektovanje aplikacije IBuyAdventure koja neće koristiti stanje sesije u skladu je sa ASP.NET filozofijom.

Poslovni objekti i sklopovi

Za poslovne objekte koji su napisani u jeziku C# obezbedite različite ASP.NET stranice sa svom potrebnom poslovnom logikom i kodom za pristup podacima. S obzirom na to da su sve .NET klase prilično slične po strukturi i ADO.NET kodu koji koriste za pristup bazi podataka SQL servera, nećemo davati pregled svakog metoda za svaki objekat. Umesto toga ćemo detaljno govoriti o jednoj od komponenti (`ProductsDB`) da bismo videli osnovnu strukturu komponenti a zatim koristiti alat ILDASM da bismo zbog referenciranja prikazali metode i svojstva svih preostalih poslovnih objekata. Kada budemo naišli na stranice koje koriste ove funkcije objasnićemo njihovu svrhu.

Poslovni objekat ProductsDB

Klasa `ProductsDB` obezbeđuje funkcije za uzimanje podataka o proizvodu. Ceo kôd u jeziku C# za klasu je ovde prikazan:

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

namespace IBuyAdventure
{
   public class ProductsDB
   {
      string m_ConnectionString;
      public ProductsDB( string dsn ) {
         m_ConnectionString = dsn;
      }

      public DataSet GetProduct(string productCode) {

        SqlConnection myConnection = new SqlConnection(m_ConnectionString);
         SqlDataAdapter sqlAdapter1 = new SqlDataAdapter("SELECT * FROM "
          + "Products WHERE ProductCode='"+productCode+"'", myConnection);

         DataSet products = new DataSet();
         sqlAdapter1.Fill(products, "products");

         return products;
      }
```

```
        public DataSet GetProducts(string category) {

          SqlConnection myConnection = new SqlConnection(m_ConnectionString);
           SqlDataAdapter sqlAdapter1 = new SqlDataAdapter("SELECT * FROM "
              + "Products WHERE ProductType='"+category+"'", myConnection);

           DataSet products = new DataSet();
           sqlAdapter1.Fill(products, "products");

           return products;
        }

        public DataSet GetProductCategories() {

          SqlConnection myConnection = new SqlConnection(m_ConnectionString);
           SqlDataAdapter sqlAdapter1 = new SqlDataAdapter("SELECT DISTINCT "
              + "ProductType FROM Products", myConnection);

           DataSet products = new DataSet();
           sqlAdapter1.Fill(products, "products");

           return products;
        }
     }
  }
```

Ova klasa ima četiri metoda uključujući konstruktor:

- `ProductsDB` – Inicijalizuje klasu stringom izvora podataka.
- `GetProduct` – Daje skup podataka koji sadrži detalje o jednom proizvodu.
- `GetProducts` – Daje skup podataka koji sadrži detalje o svim proizvodima u određenoj kategoriji.
- `GetProductCategories` – Daje skup podataka koji sadrži listu kategorija proizvoda.

Prva tri reda komponente deklarišu korišćeni prostor imena:

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

Sve datoteke klasa imaju ove redove koji pokazuju da koristimo standardni sistemski prostor imena, prostore imena za ADO.NET i specifične delove za SQL Server ADO.NET-a (`System.Data.SqlClient`). Koristimo elemente specifične za SQL ADO.NET-a jer obezbeđuju odlične performanse za pristup SQL Serveru pomoću TDS-a (engl. *Tabular Data Stream*, tabularni tok podataka) preko klasa `SqlConnection` i `SqlDataAdapter`. Ako je potrebna podrška za različite krajnje baze podataka možemo ponovo da napišemo kôd za klase tako što ćemo koristiti klase `OleDbConnection` i `OleDbDataAdapter` koje pristupaju bazi podataka preko OLEDB-a. O ovim klasama je bilo reči u poglavlju 8.

Što se tiče celog ADO.NET koda u poslovnim objektima ovde treba napomenuti da on ne sadrži procedure za obradu izuzetaka. Zato kôd koji koristi te klase treba da hvata izuzetke kao što je `SqlException` koji može biti ispaljen ako dođe do neke greške kod pristupanja podacima (kao kada postoje dupli redovi itd.). Obradu izuzetaka nismo uključili u ASP.NET stranice da bi mogle da ostanu sažete, ali osnovni format smo ovde prikazali:

```
try
{
  someObject.SomeMethodUsingAdoDotNet()
}
catch (SqlException e)
{
    if (e.Number ==2627)
      Message.InnerHtml = "Record exists with the same primary key";
    else
      Message.InnerHtml = e.Message;
}
```

U ovom kodu proveravamo poznate greške u kodu SQL servera koji koristi svojstvo `SqlExceptionNumber`. Ako se greška u kodu slaže sa onom koju proveravamo, biće prikazana namenska poruka o grešci. Ako se poznata greška u kodu ne pronađe biće prikazano svojstvo `Message` za izuzetak. Svojstvo `Message` objekta izuzetka obično sadrži veoma opisan i koristan tekst koji može da pomogne kod brzog razrešenja problema. U aplikaciji verovatno nećete proveravati specifičnu grešku u kodu osim ako ne želite da obavite neku vrstu radnje. Na primer, možete da proverite grešku u prethodnom kodu ako želite da obrišete red koji već postoji.

> **Uvek bi trebalo da aktivno dodajete kôd za obradu izuzetaka u aplikacije proizvoda. ADO.NET klase (uključujući `SqlException`) se nalaze u sklopu `System.Data.dll`. Koristite alatku IL Disassembler (`ildasm.exe`), WinCV klasu za pregledanje ili primer klase čitača da biste detaljnije ispitali klase.**

Konstruktor stringa veze

Klasa `ProductsDB` ima konstruktor koji prihvata string veze koji se koristi za uspostavljanje veze sa krajnjom bazom podataka. Kada prosleđujemo ovakav string sprečavamo ljude da zaborave da ga odrede, i tako, nadamo se, sprečavamo da poslovni objekti sadrže fiksno upisane stringove, što je uvek loš postupak.

Prosleđeni string je pohranjen u članu `m_ConnectionString` u kodu konstruktora:

```
...
string m_ConnectionString;

public ProductsDB( string dsn ) {
   m_ConnectionString = dsn;
}
...
```

Član `m_ConnectionString` se zatim koristi kod konstruisanja objekta `SqlConnection`:

```
public DataSet GetProduct(string productCode) {

  SqlConnection myConnection = new SqlConnection(m_ConnectionString);
    SqlDataAdapter sqlAdapter1 = new SqlDataAdapter("SELECT * FROM "
      + "Products WHERE ProductCode='"+productCode+"'", myConnection);

    DataSet products = new DataSet();
    sqlAdapter1.Fill(products, "products");

    return products;
}
```

Svi koji koriste poslovni objekat `ProductsDB` (kao i bilo koji drugi poslovni objekat) moraju stoga da proslede string veze kada stvaraju instancu klase:

```
IBuyAdventure.ProductsDB inventory = new IBuyAdventure.ProductsDB(getConnStr());
```

Funkcija `getConnStr` u ovom primeru uzima string veze iz datoteke `web.config`:

```
<configuration>
  <appSettings>
    <add key="connectionString"
         value="server=localhost;uid=sa;pwd=;database=IBuyAdventure" />
    <add key="specialOffer" value="AW048-01" />
  </appSettings>
...
```

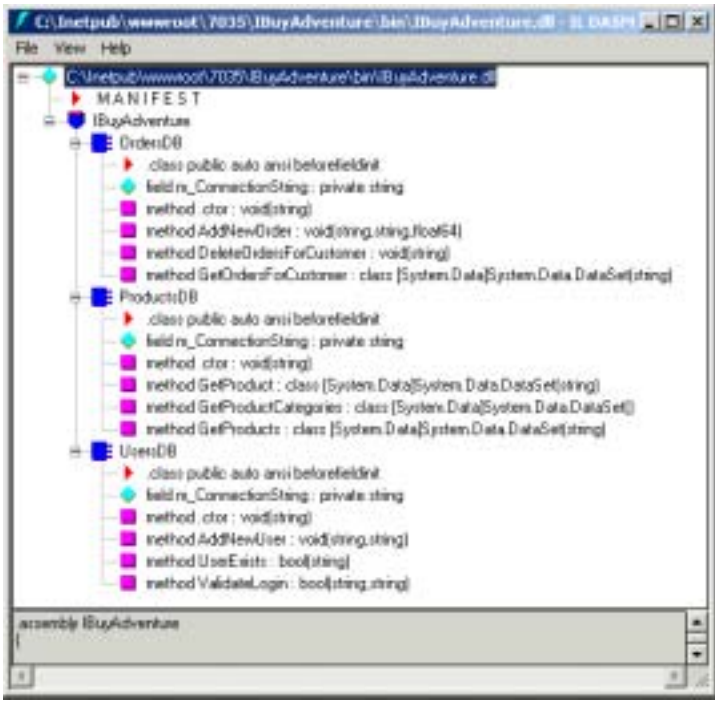
Ako koristimo datoteku `web.config` da bismo pohranili string veze za komponente (a takođe i drugu vrednost na nivou aplikacije, u prethodnom primeru) nećemo imati duplirani string veze u poslovnim objektima i ASP.NET stranicama što znatno olakšava upravljanje stringom veze ako odlučimo da bazu podataka preimenujemo ili izmenimo bilo koje svojstvo stringa veze.

Funkcija `getConnStr` je implementirana pomoću klase koda u pozadini čiji pregled ćemo dati kasnije u ovom poglavlju. Možete da koristite i datoteku za uključivanje koja će definisati ovakve funkcije u aplikaciji, ali smatramo da je pristup sa kodom u pozadini bolje rešenje.

Sada smo dali pregled klase `ProductsDB` pa ćemo sada ukratko govoriti o nekoliko slika ILDASM koje prikazuju metode za ostale poslovne objekte.

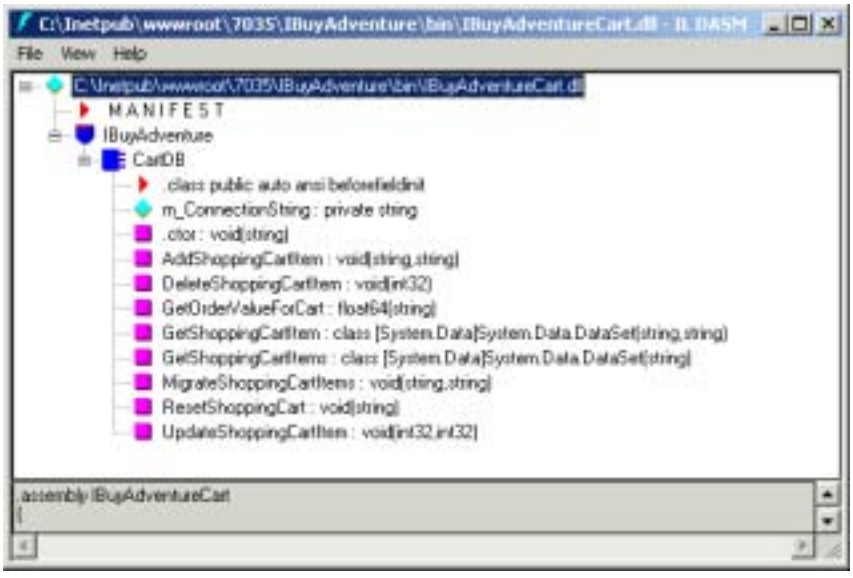
ILDASM izlaz za IBuyAdventure.dll

Sledeća slika prikazuje ILDASM izlaz za sklop `IBuyAdventure.dll`:



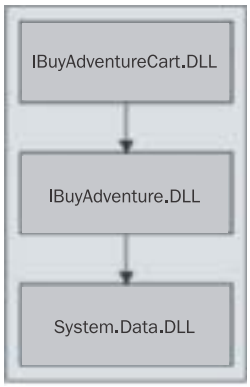
ILDASM izlaz za IBuyAdventure.dll

Sledeća slika prikazuje ILDASM izlaz za sklop `IBuyAdventureCart.dll`:



Sklopovi

Kao što je rečeno u poglavlju 17, **sklopovi** su ključni mehanizam koji se primenjuje u ASP.NET aplikaciji. Poslovni objekti za IBuyAdventure.NET su podeljeni na dva sklopa koja su zavisna od sklopa `System.Data.dll` zato što koriste ADO.NET:



`IBuyAdventureCart.DLL` sadrži poslovni objekat `CartDB` koji se koristi za manipulisanje kupovnom karticom. On zavisi od klase `ProductsDB` koja se sadrži u sklopu `IBuyAdventure.DLL`.

> **Iako sklopovi imaju oznaku tipa `.dll` oni u većem delu nisu DLL. Oznaka tipa je zadržana samo da bi potpomogla interoperabilnost između koda kojim upravlja COM+ i klasičnog COM neupravljanog koda.**

Sklop `IBuyAdventureCart.dll` nije striktno neophodan ali on pokazuje da deljenje klasa na particije u različitim sklopovima u ASP.NET aplikaciji nije težak posao. Na odluku o tome kada treba stvoriti sklopove obično utiče nekoliko realnih faktora:

- **Funkcionalnost klasa u sklopu** – sklopovi bi idealno trebalo da sadrže funkcionalnosti koje se odnose na klase.
- **Broj projektanata koji rade na aplikaciji** – sklopovi su ključne celine koje se primenjuju u ASP.NET aplikacijama tako da ima smisla da različiti projektantski timovi stvaraju svoje sklopove što bi olakšalo zajedničko projektovanje.

Kompajliranje sklopova

Ceo izvorni kôd poslovnog objekta aplikacije IBuyAdventure se nalazi u direktorijumu `components`. Ovaj direktorijum sadrži datoteku `make.bat` koja koristi C# kompajler na komandnoj liniji (`csc.exe`) da bi stvorio dva sklopa:

```
csc /out:..\bin\IBuyAdventure.dll /t:library productsdb.cs ordersdb.cs usersdb.cs
/r:System.Data.dll, System.dll, System.Xml.dll

csc /out:.. \bin\IBuyAdventureCart.dll /t:library cartdb.cs
/r:System.Data.dll, System.dll, System.Xml.dll /r:.. \bin\IBuyAdventure.dll
```

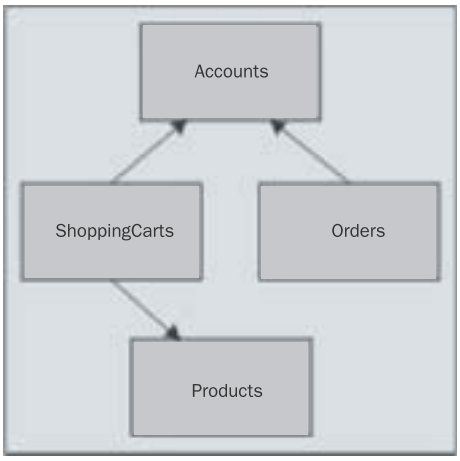
Prva naredba kompajlira poslovne objekte `ProductsDB`, `OrdersDB` i `UsersDB` koji se nalaze u odgovarajućim datotekama `productsdb.cs`, `ordersdb.cs` i `usersdb.cs`. Izlaz ove naredbe je sklop `IBuyAdventure.dll`. Druga naredba kompajlira poslovni objekat `CartDB` koji se nalazi u datoteci `cartdb.cs`. Izlaz ove naredbe je sklop `IBuyAdventureCart.dll`. Oba sklopa su kompajlirana u `bin` direktorijumu ASP.NET aplikacije tako da su raspoloživa za ASP.NET stranice.

Konvencije imenovanja

Poslovni objekti aplikacije IBuyAdventure `ProductsDB`, `OrdersDB`, i `UsersDB` su deklarisani u prostoru imena `IBuyAdventure`. To se odražava na ime sklopa u kome se objekti sadrže što omogućava da se lakše pronađe i utvrdi koje datoteke treba prebaciti kod primene aplikacije koja sadrži stranice zavisne od ovih klasa. Ista konvencija imenovanja se primenjuje na poslovni objekat `CartDB` koji je deklarisan u prostoru imena `IBuyAdventureCart` i koji se sadrži u sklopu `IBuyAdventureCart.dll`. Microsoft koristi ovu konvenciju imenovanja za većinu svojih sklopova. Izuzeci od pravila su klase glavne memorije, kao što su stringovi, koji teže da žive u sklopovima pod imenom `mscor`*`[*]`*`.dll`.

Baza podataka IBuyAdventure.NET

Aplikacijom IBuyAdventure upravlja SQL Server 7 ili 2000 baza podataka sa četiri tabele (`Accounts`, `Products`, `ShoppingCarts` i `Orders`) kao što je prikazano na sledećem dijagramu:



Poslovni objekti enkapsuliraju svaku od ovih tabela, tako da ASP.NET stranice nikada ne pristupaju direktno bazi podataka.

Tabela Accounts

Tabela `Accounts` se koristi za pohranjivanje informacija o prijavljivanju registrovanih korisnika i ima sledeću strukturu:

Ime kolone	Tip	Dužina	Opis
`CustomerName`	`nvarchar`	`50`	Ime ili elektronska adresa registrovanog korisnika. Ovo polje se koristi kao ključ za sve tabele i trebalo bi da bude jedinstveno.
`Password`	`nvarchar`	`30`	Lozinka koju korisnik određuje za vreme registracije.

Tabela Orders

Tabela `Orders` se koristi za pohranjivanje kratkog pregleda svih narudžbina koje su korisnici napravili i ima sledeću strukturu:

Ime kolone	Tip	Dužina	Opis
`CustomerName`	`nvarchar`	`50`	Ime ili elektronska adresa registrovanog korisnika. Ovo polje se koristi kao ključ za sve tabele i trebalo bi da bude jedinstveno.
`Ordered`	`datetime`	`8`	Datum kada je narudžbina napravljena.
`TotalValue`	`float`	`8`	Ukupna vrednost narudžbine.

Kada korisnik pritisne dugme `Confirm Order` i pređe na stranicu za proveru da bi potvrdio narudžbinu, u ovu tabelu se dodaje stavka. Pojedinačne stavke na kupovnoj kartici neće biti sačuvane nakon potvrde narudžbine u bazi podataka, iako će to kod komercijalnih aplikacija biti neophodno.

Tabela Products

Tabela `Products` sadrži listu svih proizvoda koje korisnik može da nabavi kod IBuyAdventure. Tabela ima sledeću strukturu:

Ime kolone	Tip	Dužina	Opis
`ProductID`	`int`	`4`	Jedinstveni ID proizvoda.
`ProductCode`	`nvarchar`	`10`	Jedinstveni kôd proizvoda.
`ProductType`	`nvarchar`	`20`	Kategorija proizvoda.
`Product Introduction Date`	`small datetime`	`4`	Datum kada je proizvod prvi put dodat u katalog.
`ProductName`	`nvarchar`	`50`	Naziv proizvoda koji je prikazan u katalogu.

Tabela se nastavlja na sledećoj strani

Ime kolone	Tip	Dužina	Opis
Product Description	nvarchar	255	Opis proizvoda.
ProductSize	nvarchar	5	Veličina proizvoda.
ProductImageURL	varchar	255	URL slike koja treba da bude prikazana za proizvod.
UnitPrice	float	8	Cena proizvoda.
OnSale	int	4	Oznaka koja pokazuje da li je cena po jedinici i prodajna cena ili ne: 1=za prodaju, 0=nije za prodaju.
Rating	float	8	Rangiranje od 1 do 5 za ovaj proizvod u smislu ukupnog kvaliteta.

Tabela IBuyAdventure ima nešto malo manje od 50 proizvoda, koji su grupisani u 12 kategorija.

Tabela ShoppingCarts

Tabela `ShoppingCarts` sadrži sve aktuelne podatke o proizvodima kupovine kartice svakog korisnika. Tabela ima sledeću strukturu:

Ime kolone	Tip	Dužina	Opis
ShoppingCartID	int	4	Automatski generisano ID polje.
ProductCode	nvarchar	10	Jedinstveni kôd proizvoda.
ProductName	char	50	Naziv proizvoda.
Description	nvarchar	255	Opis proizvoda.
UnitPrice	money	8	Cena proizvoda.
Quantity	int	4	Broj željenih komada.
CustomerName	nvarchar	50	Ime ili elektronska adresa registrovanog korisnika koji je trenutno ima određeni proizvod u svojoj korpi. Ako korisnik nije trenutno registrovan ili prijavljen onda je ovo GUID koji predstavlja anonimnog korisnika.

Kad god se stavka doda korisnikovoj kupovnoj kartici, ona se dodaje i tabeli.

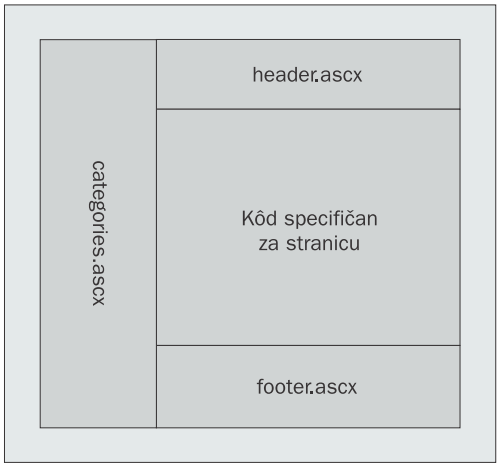
Primer aplikacije IBuyAdventure neće obrisati bazu podataka ili ukloniti redove koji se odnose na sesije koje su istekle. To bi trebalo obaviti u aplikaciji proizvoda. Trebalo bi da obradite događaj `Session_OnEnd` i tu očistite bazu podataka.

Korisnički interfejs aplikacije

Kada korisnik prvi put poseti lokaciju IBuyAdventure biće prikazana ASP.NET stranica koja daje kratak uvod u sadržaj lokacije i obezbeđuje sav standardni reklamni materijal, specijalnu ponudu i dugmad za kretanje koje očekujete od jedne aplikacije za elektronsku trgovinu:

Uvodna stranica obezbeđuje prilično intuitivan korisnički interfejs koji bi trebalo da omogući korisnicima da pretražuju, registruju i kupuju robu. Početak stranice sadrži logotip IBuyAdventure i dugmad za kretanje koja omogućavaju korisniku da se registruje, prijavi i pregleda svoje trenutne/prethodne narudžbine. Na levoj strani ekrana su dati podaci o svim kategorijama proizvoda koji su definisani u bazi podataka IBuyAdventure. Na dnu ekrana se nalazi reklamni natpis proizvoda a na preostalom delu sredine ekrana se nalaze specijalne ponude.

Sve stranice prezentacije imaju istu osnovnu strukturu kao prva stranica, tako da svaka stranica koristi najmanje tri User Controls:



User Controls u IBuyAdventure.NET

User Controls su definisane na vrhu svake stranice pomoću direktive `@Register`. Kao što je rečeno u poglavlju 4 to omogućava da se User Control poveže sa ASP.NET oznakom prefiksa (to jest, elementom namespace). Kada ASP.NET faza izvršavanja pronađe ove specijalne oznake ona zna da stvori odgovarajući User Control i da predstavi neophodan rezultat.

Direktive `@Register` koje su zajedničke za sve stranice su ovde prikazane:

```
<%@ Page Language="C#" Inherits="IBuyAdventure.PageBase"
src="components/stdpage.cs" %>
<%@ Register TagPrefix="IBA" TagName="Header" Src="UserControl\Header.ascx" %>
<%@ Register TagPrefix="IBA" TagName="Categories"
src="UserControl\Categories.ascx" %>
<%@ Register TagPrefix="IBA" TagName="Special" Src="UserControl\Special.ascx" %>
<%@ Register TagPrefix="IBA" TagName="Footer" Src="UserControl\Footer.ascx" %>
```

User Controls koje smo registrovali su zatim umetnute u stranicu na isti način kao što smo to videli u prethodnim poglavljima:

```
<IBA:Header id="Header" runat="server" />
```

Većina stranica u aplikaciji IBuyAdventure ima isti osnovni format koji sadrži HTML tabelu. Zato ćemo dati pregled kompletnog koda stranice za `Default.aspx` koja prikazuje sve User Controls koje su deklarisane, jezik, direktivu stranice koda u pozadini, podrazumevanu direktivu izlaza keša i osnovnu strukturu HTML stranice:

```
<%@ Page Language="C#" Inherits="IBuyAdventure.PageBase"
src="components/stdpage.cs" %>
<%@ Register TagPrefix="IBA" TagName="Header" src="UserControl\Header.ascx" %>
<%@ Register TagPrefix="IBA" TagName="Categories"
src="UserControl\Categories.ascx" %>
<%@ Register TagPrefix="IBA" TagName="Special" src="UserControl\Special.ascx" %>
<%@ Register TagPrefix="IBA" TagName="Footer" src="UserControl\Footer.ascx" %>
<%@ OutputCache Duration="60" VaryByParam="*" %>

<script language="C#" runat="server" >

   private String GetCustomerID() {
      if (Context.User.Identity.Name != "")
         return Context.User.Identity.Name;
      else {
         if (Session["AnonUID"] == null)
            Session["AnonUID"] = Guid.NewGuid();
         return Session["AnonUID"].ToString();
      }
   }

   void Page_Load(Object sender, EventArgs e) {
      if (Request.Params["Abandon"] == "1")
      {
         IBuyAdventure.CartDB cart = new
IBuyAdventure.CartDB(ConfigurationSettings.AppSettings["connectionString"]);
         cart.ResetShoppingCart(GetCustomerID());
         Session.Abandon();
         FormsAuthentication.SignOut();
      }
   }
</script>

<html>
  <head>
    <title>IBuyAdventure Catalog</title>
  </head>
  <body background="images/back_sub.gif">
    <form runat="server">
    <font face="Verdana, Arial, Helvetica" size="2">
      <table border="0" cellpadding="0" cellspacing="0">
        <tr>
          <td colspan="5">
            <IBA:Header id="Header" runat="server"/>
          </td>
        </tr>
        <tr>
          <td colspan="3" align="left" valign="top">
            <IBA:Categories id="Categories" runat="server"/>
          </td>
          <td>
              
          </td>
          <td align="left" valign="top">
            <h3>Welcome to IBuyAdventure!</h3>
```

```
            <p>
            <font face="Verdana, Arial, Helvetica" size="2">
              You know the drill: Proper equipment for your climb leads to
              a successful ascent. IBuyAdventure gear has been tested in
              the most extreme environments on earth, from the 8,000-meter
              peaks of the Himalayas to the sub-arctic giants of Alaska.
            <p>
            <IBA:Special runat="server"/>
            <p>
              IBuyAdventure has all the gear you need for any excursion,
              from a day hike to a major expedition. Shop with us, set up
              camp with us, and take our challenge. Join the IBuyAdventure
              expedition!
            <br>
            <br>
            <br>
            <IBA:footer runat="server"/>
            </font>
          </td>
        </tr>
        </table>
    </font>
</form>
</body>
</html>
```

Iako prva stranica izgleda prilično bogato, količina koda na stranici je u stvari veoma mala jer je veći deo HTML-a i koda enkapsuliran u tri User Controls. `default.aspx` kao i većina stranica koristi direktivu `@OutputCache` koja određuje da stranice treba keširati 60 sekundi. Tako se smanjuju dodatni poslovi za bazu podataka, ali bi trebalo razmotriti i nekoliko pitanja:

- Keširane informacije su pohranjene u memoriji tako da će količina memorije koju aplikacija koristi biti veća.
- Ista stranica će biti keširana više puta ako ima različite parametre upita, tako da ćete morati da omogućite to povećanje u skupu poslova.
- Ako je stranica keširana onda će svi izlazi za tu stranicu biti takođe keširani. To možda zvuči očigledno, ali to ne znači da se, na primer, kontrola `AdRotator` za aplikaciju Adventure Work ne rotira isto toliko često kao uobičajena lokacija (reklama se menja svakih 60 sekundi na stranicama koje koriste keširanje). Ako želimo da delovi stranice budu keširani, dok je preostali deo svaki put predstavljen kao osvežen, možemo da koristimo **fragmentarno keširanje**. Fragmentarno keširanje radi tako što se informacije keširaju u User Control – tako da je stranica `.aspx` svaki put predstavljena – ali kada treba dodati sadržaj User Controla stranici, taj sadržaj će biti povučen iz keša.

Samo jedan element <form> na strani servera

Ono što je važno napomenuti o stranici `default.aspx` je to da sadrži jedan element `<form>` sa atributom `runat="server"`. Ovaj oblik sadrži većina HTML-a stranica. Nijedan od User Controls nema element `<form>` na strani servera. Ovo je važno zato što elementi `<form>` *ne mogu* biti ugnežđeni, tako da jedan oblik mora sadržati *ceo* User Controls kôd. Ako pokušate da definišete element `<form>` sa atributom `runat="server"` bilo gde u spoljašnjem elementu `<form>`, biće generisana greška.

Korišćenje jezika C# za User Controls i kôd

Prvi red koda svih stranica u IBuyAdventure sadrži direktivu `@Page`:

```
<%@ Page Language="C#" Inherits="IBuyAdventure.PageBase"
         src="components/stdpage.cs" %>
```

Ovakvu vrstu direktive smo prvi put videli u poglavlju 4. Direktiva koju ovde koristimo obaveštava ASP.NET kompajler o dve ključne stvari o stranici:

- Ceo kôd stranice je napisan pomoću jezika C#. (Iako smo isto tako lako mogli da koristimo mnoge druge jezike.)
- Svaka stranica koristi kôd u pozadini i izvodi ga iz .NET klase `PageBase` koja obezbeđuje uobičajenu funkcionalnost.

Osnovna motivacija za korišćenje jezika C# za pisanje aplikacije IBuyAdventure je bila da pokažemo da se zaista ne razlikuje mnogo od JScripta i VB-a i da se lako čita i razume. Sam ASP.NET je napisan u jeziku C# što pokazuje da ima solidnu budućnost pred sobom. S obzirom na to da se svi .NET jezici kompajliraju iz MSIL pre nego što se izvrše. Tada nije važno u kom jeziku je kôd napisan – kao što je prethodno rečeno u ovoj knjizi, trebalo bi da koristite onaj jezik koji najbolje poznajete.

Klasa koda u pozadini koja je određena pomoću atributa `Inherits` i `Src` dovodi do toga da ASP.NET kompajler stvara stranicu koja je izvedena iz klase `PageBase` a ne klase `Page`. Implementacija klase `PageBase` je veoma jednostavna:

```
using System;
using System.Collections;
using System.Web.UI;
using System.Web.Security;
using System.Configuration;

namespace IBuyAdventure
{
   public class PageBase : Page
   {
      public string getConnStr() {
      string dsn;
      dsn = ConfigurationSettings.AppSettings["connectionString"];
      return dsn;
      }
   }
}
```

Kada se svaka stranica izvede iz ove klase funkcija `getConnStr` će biti raspoloživa u svakoj od ASP.NET stranica. Ova funkcija uzima string veze baze podataka iz datoteke `web.config` i poziva je u stranicama kod konstruisanja poslovnih objekata koji se povezuju sa krajnjim izvorom podataka. Datoteka `web.config` je keširana, pa kada joj često pristupamo u stranicama to neće negativno uticati na performanse. Ako želite da keširate samo string veze možete da ga sačuvate u kešu podataka tako što ćete na početku pristupiti datoteci `web.config` da biste uzeli vrednost kada budete stvarali stavku keša:

```
public String getConnStrCached() {
   string connectionString;

   // Proveri u ke{u ConnectionString
   connectionString = (string) Context.Cache["connectionString"];

   // Ako ConnectionString nije u ke{u, pozovi ga iz datoteke Config.web
   if (connectionString == null) {
      connectionString =
         ConfigurationSettings.AppSettings["connectionString"];

      // pohrani ke{
      Cache["connectionString"] = connectionString;
   }
   return connectionString;
}
```

Ono što treba razmotriti kod korišćenja keša podataka je to da će vrednosti koje se u njemu čuvaju *biti* ažurirane ako neko izmeni datoteku `web.config`. ASP.NET automatski stvara novi domen aplikacije i u suštini restartuje Web aplikaciju da bi obradio sve nove Web zahteve u trenutku kada je datoteka `web.config` izmenjena. S obzirom na to da se tako stvara novi keš podataka, novi string veze će biti keširan nakon prvog pozivanja `getConnStrCached`.

> **Kao što je rečeno u poglavlju 13 parametri aplikacije bi uvek trebalo da budu pohranjeni u sekciji `appsettings` datoteke `web.config`. Vrednosti iz ove sekcije su automatski keširane.**

Međutim, ako odlučite da pohranite konfiguraciju aplikacije na drugom mestu (možda u vašoj XML datoteci na centralnom serveru) keš još uvek može biti nevažeći ako su datoteke izmenjene stvaranjem zavisnosti datoteka. To omogućava da stavka keša automatski bude obrisana kada u određenoj datoteci dođe do izmene:

```
        //store to cache
        Cache.Insert("connectionString", connectionString,
          new CacheDependency(Server.MapPath("\\someserver\myconfig.xml")));
```

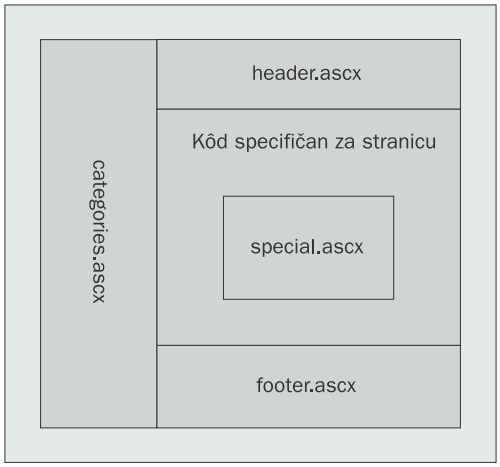
Zavisnosti datoteka su samo jedan od tipova zavisnosti keša koje podržava ASP.NET. Ostali podržani tipovi su:

- **Sređivanje** – brisanje stavki keša na osnovu njihove upotrebe, korišćenja memorije i rangiranja
- **Isticanje** – brisanje stavki keša u određenom vremenu ili nakon perioda neaktivnosti/pristupa
- **Zavisnosti datoteke i ključa** – brisanje stavki keša kada se promeni datoteka ili kada se promeni neka stavka keša

Ako želite više detalja o keširanju pročitajte poglavlje 12.

User Control Specials – Specials.ascx

Kao što ste možda primetili, stranica `default.aspx` koju smo prethodno videli, a koja implementira uvodnu stranicu, u stvari koristi dodatni User Control (`UserControlÐSpecial.ascx`) da bi prikazala današnji specijalni proizvod, tako da je struktura stranice nešto složenija nego što bi inače bila:



Ponuđeni proizvod čuva se u datoteci `web.config` što olakšava administratoru lokacije da izmeni prikazani proizvod:

```
<configuration>
  <appSettings>
    <add key="connectionString"
         value="server=localhost;uid=sa;pwd=;database=IBuyAdventure" />
    <add key="specialOffer" value="AW048-01" />
  </appSettings>
   ...
</configuration>
```

User Control `Special` čita ovu vrednost u svojoj proceduri za obradu događaja `Page_Load` tako što uzima informacije o proizvodu pomoću komponente `ProductDB` i ažurira stranicu pomoću kontrola na strani servera:

```
...
<%@ Control Inherits="IBuyAdventure.ControlBase" src="../components/
stdctrl.cs" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Configuration" %>

<script language="C#" runat="server">

   void Page_Load(Object sender, EventArgs e) {
      // Nabavi današnji specijalni proizvod.
      IBuyAdventure.ProductsDB inventory =
         new IBuyAdventure.ProductsDB(getConnStr());
```

```
        string specialOffer;
        specialOffer = ConfigurationSettings.AppSettings["specialOffer"];
        DataSet specialDetails = inventory.GetProduct(specialOffer);
        // Ažuriraj UI podacima o proizvodu
        ProductImageURL.Src = Context.Request.ApplicationPath + "/images/" +
           (String) specialDetails.Tables[0].Rows[0]["ProductImageURL"];
        ProductName.Text =
           (String) specialDetails.Tables[0].Rows[0]["ProductName"];
        ProductDescription.Text =
           (String) specialDetails.Tables[0].Rows[0]["ProductDescription"];
        ProductCode.Text =
           (String) specialDetails.Tables[0].Rows[0]["ProductCode"];
        UnitPrice.Text = String.Format("{0:C}",
           specialDetails.Tables[0].Rows[0]["UnitPrice"]);
        OrderAnchor.HRef = Request.ApplicationPath +
           "/ShoppingCart.aspx?ProductCode=" +
           (String) specialDetails.Tables[0].Rows[0]["ProductCode"];
        if  ( (int) specialDetails.Tables[0].Rows[0]["OnSale"] == 0 )
           sale.Visible = false;
    }

</script>

<table width="400" align=center border="1" cellpadding="0" cellspacing="0">
  <tr bgcolor="#F7EFDE">
    <td>
     <font face="verdana" size="2" ><b>  Today's Special! </b></font>
    </td>
  </tr>
  <tr>
    <td>
      <table>
        <tr>
          <td align="left" valign="top" width="1"10>
             <img id="ProductImageURL" runat="server">
          </td>
          <td align="left" valign="top">
            <font size="2">
              <b><asp:label id="ProductName" runat="server"/></b>,
              <asp:label id="ProductDescription" runat="server"/><br><br>
              <table>
                <tr>
                  <td>
                  <font size="2">
             Product Code: <asp:label id="ProductCode" runat="server"/><br>
                   Price: <b><asp:label id="UnitPrice" runat="server"/></b>
                  </font>
                  </td>
                  <td>
                    
               <img src="../images/saleTag1.gif" id="sale" runat="server" >
                  </td>
                </tr>
              </table>
              <br>
              <a id="OrderAnchor"
```

```
                    href='../ShoppingCart.aspx?ProductCode=AW109-15'
                    runat="server">
                   <img src="images/order.gif" width="55" height="15"
                        alt="Order" border="0">
                </a><br><br>
              </font>
            </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
```

User Control kôd je veoma jednostavan i samo ažurira serverske kontrole vrednostima iz `DataSet` dobijenim pozivanjem funkcije u `GetProduct`. Funkcija `StringFormat` je pozvana pomoću stringa formata `{0:C}` da bi `UnitPrice` bilo prikazano kao numerička vrednost koja predstavlja iznos u lokalnoj valuti.

User Controls kategorije – Categories.ascx

Lista kategorija proizvoda (`categories.ascx`) koja je na većini stranica prikazana na levoj strani (nije na stranici za proveru ili obračun) dinamički se formira pomoću kontrole `asp:DataList` i poslovnog objekta `ProductsDB`. Svojstvo `DataSource` za kontrolu je podešeno u događaju `Page_Load`:

```
<%@ Control Inherits="IBuyAdventure.ControlBase" src="../components/
stdctrl.cs" %>
<%@ OutputCache Duration="60" VaryByParam="none" %>

<script language="C#" runat="server">

   void Page_Load( Object sender, EventArgs e ) {
      String dsn = getConnStr();
      IBuyAdventure.ProductsDB inventory =
         new IBuyAdventure.ProductsDB(getConnStr());
      CategoryList.DataSource = inventory.GetProductCategories();
      CategoryList.DataBind();
   }

</script>
```

`ItemTemplate` za ovu kontrolu liste podataka je detaljno naveden u User Control i određuje raspored podataka:

```
<asp:datalist id="CategoryList" border="0" runat="server">
  <itemtemplate>
    <tr>
      <td valign="top">
        <asp:image imageurl="/IBuyAdventure/images/bullet.gif"
                   alternatetext="bullet" runat="server" />
      </td>
      <td valign="top">
        <font face="Verdana, Arial, Helvetica" size="2">
          <asp:hyperlink
```

```
            NavigateURL='<%# "/IBuyAdventure/catalogue.aspx?ProductType=" +
            DataBinder.Eval( Container.DataItem, "ProductType" )%>'
          Text='<%# DataBinder.Eval( Container.DataItem, "ProductType" )%>'
            runat="server"/>
        </font>
      </td>
    </tr>
  </itemtemplate>
</asp:datalist>
```

Kontrola `asp:DataList` je prvi put viđena u poglavlju 7. Povezana je u kolekciji sa izvorom podataka stavki i predstavlja `ItemTemplate` za svaku od njih.

Kontrola `asp:DataList` daje kao izlaz HTML tabelu; dakle `ItemTemplate` koji smo napisali daje kao izlaz element `<tr>` koji sadrži dve kolone (elementi `<td>`) koje obezbeđuju da tabela i stranica budu pravilno predstavljene. Prva kolona sadrži kontrolu `asp:image` koja predstavlja malu bitmapu u obliku kamena kao grafičku oznaku za nabrajanje, druga kolona sadrži kontrolu `asp:hyperlink` koja ima dva polja (`NavigateURL` i `Text`). Ova polja su povezana sa aktuelnim redom za `DataSet` koji vraća poslovni objekat `ProductDB`.

Hiperveza predstavljena u `ItemTemplate` omogućava korisniku da pregleda detalje o proizvodu u određenoj kategoriji. Atribut `NavigateURL` je izračunato polje koje se sastoji od utvrđenog URL-a (`/IBuyAdventure/catalogue.aspx`) i parametra dinamičkog upita `ProductType` čija vrednost je podešena da bude jednaka polju `ProductType` u aktuelnom redu skupa podataka. I na kraju, atribut `Text` je jednostavan atribut čija vrednost je takođe jednaka polju `ProductType` u aktuelnom redu skupa podataka.

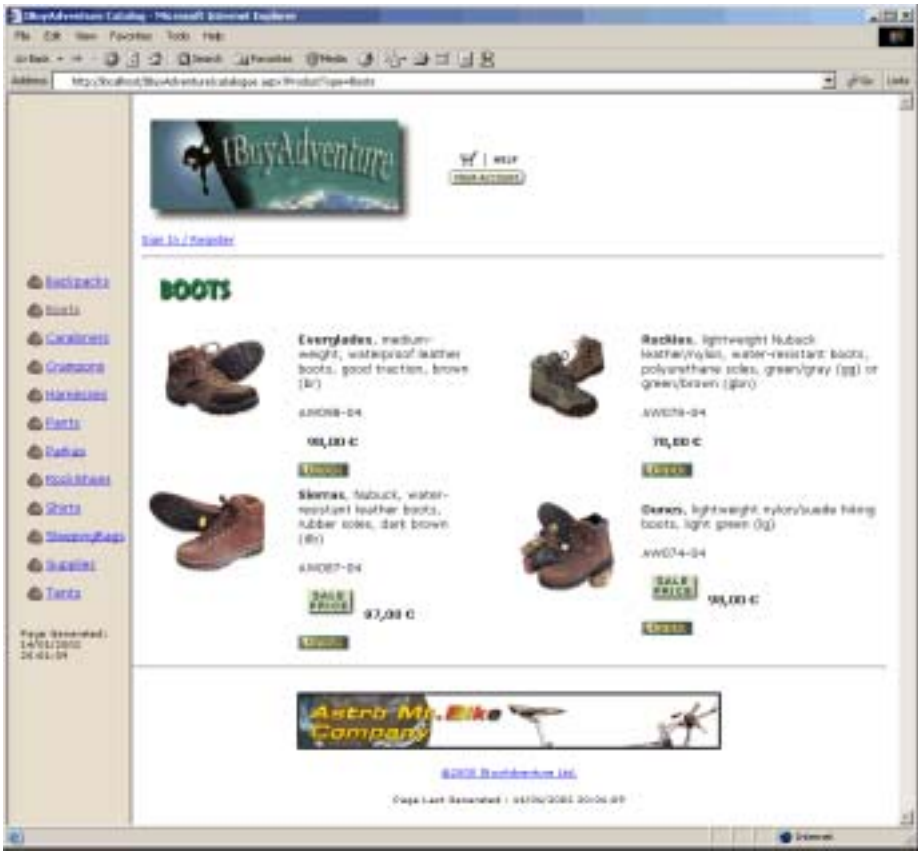
Klasa `DataBinder` se koristi za uzimanje vrednosti koje su pohranjene u ovim svojstvima. Ako vam je to neobično, klasa `DataBinder` je samo pomoćna klasa koju obezbeđuje ASP.NET da bi kôd bio jednostavniji (manje konverzija) i čitkiji, naročito ako je potrebno da se svojstvo formatira.

Ili, sami možemo direktno da pristupimo aktuelnom redu `DataSet` i da pomoću sledećeg koda uzmemo vrednost `ProductType`:

```
((DataRowView)Container.DataItem)["ProductType"].ToString()
```

Ovaj format je nešto složeniji, ali može biti bolji ako volite da koristite konverzije i više volite taj stil. Prednost ovog koda je u tome što je rano povezan, pa će se izvršavati brže nego kasno povezana sintaksa klase `DataBinder`.

Kada odaberemo jednu od hiperveza za kategoriju proizvoda, biće prikazana ASP.NET stranica `Catalogue.aspx`:



Ova stranica prikazuje proizvode iz odabrane kategorije pomoću parametra stringa upita `ProductType` u događaju `Page_Load` da bi rezultati vraćeni iz komponente `ProductDB` mogli da budu filtrirani:

```
void Page_Load(Object sender, EventArgs e) {
   if (!IsPostBack) {
      // Treba utvrditi koja kategorija proizvoda je određena i ažurirati
      // deo slike
      String productType = Request.Params["ProductType"];
      CatalogueSectionImage.Src = "images/hd_" + productType + ".gif";

      // Korisnički poslovni objekat treba da pozove kategoriju
      // proizvoda i poveže te
      // podatke sa kontrolom <asp:datalist>
      IBuyAdventure.ProductsDB inventory =
         new IBuyAdventure.ProductsDB(getConnStr());
      MyList.DataSource = inventory.GetProducts(productType);
      MyList.DataBind();
   }
}
```

Ovde je u projektovanju problem u tome što je za izmene prikazanih proizvoda korišćena hiperveza a ne povratna informacija.

Svaki od prikazanih proizvoda u odabranoj kategoriji ima nekoliko detalja:

- **Product name (naziv proizvoda)** – naziv proizvoda (Everglades, Rockies itd.)
- **Product Info (informacije o proizvodu)** – podaci o proizvodu koji će zanimati klijente i pomoći im da odluče o nabavci (da li je predmet vodootporan, kakve je boje itd.)
- **Product Code (kôd proizvoda)** – jedinstveni ID proizvoda za celu lokaciju
- **Price (cena)** – cena proizvoda
- **On Sale (rasprodaja)** – ako je cena smanjena, biće prikazana slika SALE PRICE
- **Order Button (dugme za narudžbinu)** – da bi dodao proizvod u korpu korisnik će odabrati sliku Order

Glavno telo ove stranice je takođe generisano pomoću kontrole `asp:DataList` tako što je izvor podataka podešen u događaju `Page_Load` a pomoću `ItemTemplate` se kontroliše predstavljanje svakog proizvoda. Za razliku od kategorije User Control, `asp:datalist` na ovoj stranici će koristi prednosti atributa `RepeatDirection` i `RepeatColumns`:

```
<asp:datalist id="MyList" BorderWidth="0" RepeatDirection="vertical"
   RepeatColumns="2" runat="server" OnItemDataBound="DataList_ItemBound">
```

Ovi atributi automatski obavljaju raspored na stranici i omogućavaju da aplikacija izgleda profesionalno. Sve što treba da uradimo u `ItemTemplate` je da definišemo tabelu sa dve kolone koja u prvoj koloni sadrži sliku a u drugoj daje podatke. Kontrola `asp:DataList` zatim organizuje redosled izvršavanja redova – da bi na stranici bio korišćen horizontalni redosled izvršavanja samo treba da promenimo vrednost atributa:

```
<asp:datalist id="MyList" BorderWidth="0" RepeatDirection="horizontal"
   RepeatColumns="2" runat="server" OnItemBound="DataList_ItemBound">
```

Sada dobijamo drugačiji raspored stavki:

Bez kontrole `asp:DataList` koja obezbeđuje ovu funkcionalnost, trebalo bi sami da napišemo neznatnu količinu koda da bismo to postigli.

Ako pogledate prvobitnu ASP aplikaciju Adventure Works videćete da je za nju ukupno bilo potrebno oko 100 redova koda.

Kada je neki predmet na prodaju biće prikazana ova bitmapa:

Da bismo utvrdili da li je slika prikazana ili ne, obradićemo događaj `OnItemBound` objekta `asp:DataList`. Ovaj događaj je istaknut kadgod je stvorena stavka u listi podataka. To ćemo uraditi kada podesimo svojstvo `Visible` serverske kontrole `saleItem` (elemenet `img`) na `false` ako je svojstvo `OnSale` jednako nuli. Da bismo dobili referencu za kontrolu `saleItem` treba da koristimo metod `FindControl` objekta stavke `DataList`. Ovaj metod će potražiti sve zavisne kontrole stavke `DataList` koja je dodata stranici tako da za tu stavku možemo da dobijemo referencu za kontrolu `saleItem`:

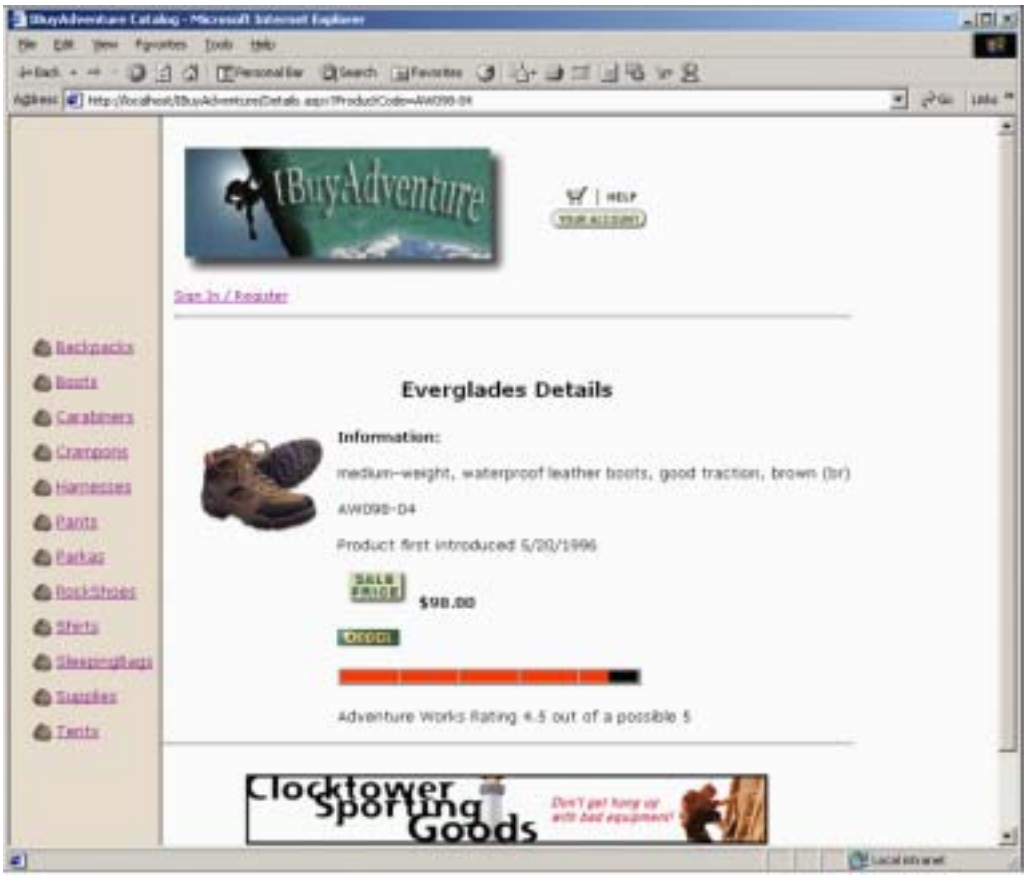
```
...
void DataList_ItemCreated(Object sender , DataListItemEventArgs e ) {

   DataRowView myRowView;
   DataRow myRow;

   myRowView = (DataRowView) e.Item.DataItem;
   myRow = myRowView.Row;

   if  ( (int) myRow["OnSale"] == 0 )
      e.Item.FindControl("saleItem").Visible = false;
...

<img src="images/saleTag1.gif" id="saleItem" runat="server" />

...
```

Kada podesimo svojstvo `Visible` na `false`, ASP.NET faza izvršavanja ne predstavlja kontrolu ili bilo koju zavisnu kontrolu – kao što bi to bilo da smo koristili nešto slično elementu `<span>` koji sadrži sliku i tekst. Ovo je veoma moćan pristup za sprečavanje parcijalnog generisanja stranice, i mnogo je čistiji od unutrašnjih `if...then` naredbi koje je klasični ASP zahtevao da napišemo.

Prednost ovog pristupa je u tome da je kôd nešto čistiji i lakši za održavanje ali, što je još važnije, je da sve izmene koje kôd napravi u kontrolama koje čuvaju svoje stanje prežive ponovne pozive (engl. *postbacks*). S obzirom na to da je unutrašnji kôd izvršen za vreme faze predstavljanja ASP.NET stranice, stanje pregledanja (stanje koje čuvaju stranica i/ili zavisne kontrole) je *već* sačuvano tako da sve izmene u unutrašnjem kodu neće biti ponovo vraćene za vreme ponovnog poziva. Razlog za korišćenje unutrašnjeg koda u ovom poglavlju je da pokažemo da iako ASP.NET aplikacije još uvek mogu da koriste unutrašnji kôd, postoje bolji (a ponekad obavezni) alternativni pristupi koji omogućavaju da se na mnogo bolji način odvoji kôd od sadržaja.

Podaci o proizvodu

Za svaki proizvod prikazan na stranici `catalogue.aspx` ime proizvoda je predstavljeno kao hiperveza. Ako je klijentu pregled proizvoda zanimljiv, on može da odabere vezu da bi video više podataka o njemu (priznajemo da u našem primeru aplikacije nema mnogo dodatnih podataka):



Dodatne informacije na ovom ekranu uključuju datum kada je proizvod prvi put uveden i rangiranje proizvoda koje određuje tim ljudi koji pregledaju IBuyAdventure. Tim uvek testira opremu koja se prvi put prodaje a zatim je rangira. Polje `Rating` u tabeli `Products` određuje liniju za rangiranje koja je prikazana za svaki proizvod. Sama linija je generisana pomoću namenske serverske kontrole koja je napisana za IBuyAdventure. Izvorni kôd za ovu kontrolu rangiranja se nalazi u direktorijumu `Controls` i trebalo bi da bude lako razumljiv ako ste pročitali poglavlje 18. Poput direktorijuma `Components` direktorijum `Controls` sadrži datoteku `make.bat` za formiranje kontrole.

Kontrola je registrovana i dodeljeno joj je ime elementa (oznaka prefiksa) na vrhu stranice sa podacima:

```
<%@ Register TagPrefix="Wrox" Namespace="WroxControls" %>
```

Iako pokazuje samo jedan proizvod, stranica `details.aspx` još uvek koristi kontrolu `asp:DataList`. Razlog za ovo je bio taj da će buduće verzije aplikacije IBuyAdventure možda moći da omoguće da u istom trenutku za više proizvoda bude dat pregled podataka da bi proizvodi mogli da se porede. Kontrola rangiranja je zato deklarisana u `ItemTemplate` za kontrolu `asp:DataList` kao svojstvo `Score` pomoću polja koje je nazvano `Rating` u tabeli baze podataka:

```
<Wrox:RatingMeter runat="server"
Score=<%#(double)DataBinder.Eval(Container.DataItem, "Rating")%>
Votes="1"
MaxRating="5"
CellWidth="51"
CellHeight="10" />
```

Dok svojstva kontrole rangiranja isprva mogu izgledati zbunjujuće, trebalo bi da razumete da je to generička kontrola koja odgovara mnogim poslovima. Ako ste videli članak o sistemu rangiranja u `ASPToday.com` onda će vam biti jasno, ali ako niste, razmotrite slučaj kada 200 ljudi rangira proizvod tako da imate 200 glasova. Za svaki glas se dodeljuje ukupni rezultat između 0 i `MaxRating`, a atribut `Score` odražava ukupan prosek svih glasova.

Kontrola rangiranja u stvari podržava više funkcionalnosti nego što je potrebno za aplikaciju IBuyAdventure, tako da ćemo svojstvo `Votes` podesiti na `1` jer samo jedan zaposleni rangira proizvode. Stvar je tome da će u budućim verzijama korisnici moći sami da rangiraju i pregledaju proizvode.

U ovom poglavlju više nećemo govoriti o funkcionalnostima kontrola rangiranja, pa ćemo da bismo vam pomogli, dati rezime svojstava koja će vas uputiti u pravom smeru:

Svojstvo	Opis
`CellWidth`	Veličina svake ćelije u liniji.
`MaxRating`	Maksimalno rangiranje koje može dodeliti jedan glas. Ova vrednost određuje broj ćelija koje linija ima.
`CellHeight`	Visina svake ćelije.
`Votes`	Broj glasova koji su konvertovani.
`Score`	Trenutni ukupni rezultat rangiranja.

Kupovna kartica

Kada pretražuje lokaciju korisnik u bilo kom trenutku može da doda predmete u svoju korpu tako što će odabrati sliku hiperveze `Order`:

Ovo dugme u obliku slike je umetnuto u stranicu `catalogue.aspx` u trenutku njenog stvaranja a kada ga pritisnete stižete do čitača pomoću koga ćete se kretati do stranice `ShoppingCart.aspx`:

```
<asp:ImageButton runat="server" id="OrderButton"
 ImageUrl="images/order.gif"
 OnCommand="OrderButton_Command"
 CommandName="Order"/>
```

Postoje dva dela koda koja treba dodati da bi stranica podržavala ovo dugme. Prvo, s obzirom na to da će se ovo dugme pojavljivati više puta na stranici, treba da povežemo svaku instancu sa određenim proizvodom. Kada korisnik pritisne dugme, znaćemo koji proizvod je odabrao.

```
   void DataList_ItemBound(Object sender , DataListItemEventArgs e ) {
      DataRowView myRowView;
      DataRow myRow;
      myRowView = (DataRowView) e.Item.DataItem;
      myRow = myRowView.Row;
      if ( (int) myRow["OnSale"] == 0 )
         e.Item.FindControl("saleItem").Visible = false;
     ((ImageButton)e.Item.FindControl("OrderButton")).CommandArgument =
         myRow["ProductCode"].ToString();
      ((ImageButton)e.Item.FindControl("OrderButton")).AlternateText =
         "Click to order " + myRow["ProductName"];
   }
```

Metod `DataList_ItemBound()` se poziva svaki put kada se iz baze podataka proizvod dodaje kontroli `DataList`. Svojstvo `CommandArgument` ćemo podesiti na određeni parametar `ImageButton` koji će biti kôd nekog određenog proizvoda. Kasnije ćete videti kako se ovo koristi za izbor odgovarajućeg proizvoda. Možemo da koristimo i svojstvo `AlternateText` da bismo podesili okvir sa podacima koji će se pojavljivati kada strelica miša preće preko dugmeta za naručivanje.

Zatim treba obraditi povratni događaj a to je kada korisnik pritisne dugme za naručivanje proizvoda koji želi da kupi. Tako će vraćamo do servera i pokrećemo događaj `OrderButton_Command`:

```
void OrderButton_Command(object sender, CommandEventArgs e) {
   if (e.CommandName == "Order") {
      String prodCode = e.CommandArgument.ToString();
      Response.Redirect ("ShoppingCart.aspx?ProductCode=" + prodCode);
   }
}
```

Kada je događaj obrađen, treba proveriti šta je `CommandName` za dugme koje pokreće ovaj događaj. Ako odgovara događaju `Order` tada znamo da je pokrenut zato što je korisnik pritisnuo dugme za određeni proizvod. Svojstvo `CommandArgument` će sadržati kôd tog proizvoda. Zatim možemo da preusmerimo izvršavanje na stranicu `ShoppingCart.aspx` i prosledimo kôd proizvoda kao parametar.

Stranica `ShoppingCart.aspx` izgleda ovako:

Kada je generisana stranica `ShoppingCart.aspx` događaj `Page_Load` proverava da li je novi proizvod dodat na karticu tako što će potražiti parametar `Request` pod nazivom `ProductCode`. On će kao string upita pomoću koda biti dodat URL-u na stranicu `Catalogue.aspx` (kao što je to prethodno pokazano). Zatim se poziva funkcija `AddShoppingCartItem` objekta `CartDB` koja kupovnoj kartici trenutnog korisnika treba da doda parametar.

Procedura za obradu događaja `Page_Load` za stranicu `Shopping.aspx` je ovde prikazana:

```
void Page_Load(Object sender, EventArgs e) {
  IBuyAdventure.CartDB cart = new IBuyAdventure.CartDB(getConnStr());
    // If page is not being loaded in response to postback
    if (Page.IsPostBack == false) {
       // If a new product to add is specified, add it
       // to the shopping cart
       if (Request.Params["ProductCode"] != null) {
          cart.AddShoppingCartItem(
             GetCustomerID(), Request.Params["ProductCode"]);
       }
       PopulateShoppingCartList();
       UpdateSelectedItemStatus();
    }
}
```

Parametar `ProductCode` postoji kao opcija zato što kupovna kartica može biti prikazana i kada odaberemo simbol kartice prikazan na liniji za kretanje. Ako je to metod pomoću koga je pristupljeno stranici, onda stavke nećemo dodavati kupovnoj kartici. Funkcija `CustomerID` koja je ovde

korišćena daje jedinstveni ID za sve trenutne korisnike, a zatim se kao parametar prosleđuje funkciji `AddShoppingCartItem`. Ako klijent nije registrovan i prijavljen, ID koji vraća funkcija `CustomerID` je aktuelni ID sesije ASP.NET-a; u suprotnom, to je aktuelno korisničko ime:

```
        String GetCustomerID() {
           if (User.Identity.Name != "") {
              return Context.User.Identity.Name;
           }
           else {
              if (Session["AnonUID"] == null)
                 Session["AnonUID"] = Guid.NewGuid();
              return Session["AnonUID"].ToString();
           }
        }
```

Implementaciju metoda `AddShoppingCartItem` poslovnog objekta `CartDB` bi u ovom trenutku trebalo ponovo pogledati, jer sadrži dva zanimljiva dela koda:

```
public void AddShoppingCartItem(string customerName, string productCode) {

   DataSet previousItem = GetShoppingCartItem(customerName, productCode);

   if (previousItem.Tables[0].Rows.Count > 0) {
      UpdateShoppingCartItem((int)
         previousItem.Tables[0].Rows[0]["ShoppingCartID"],
         ((int)previousItem.Tables[0].Rows[0]["Quantity"]) + 1);
   }
   else {

      IBuyAdventure.ProductsDB products;
      products = new IBuyAdventure.ProductsDB(m_ConnectionString);
      DataSet productDetails = products.GetProduct(productCode);

      String description =
         (String) productDetails.Tables[0].Rows[0]["ProductDescription"];
      String productName =
         (String) productDetails.Tables[0].Rows[0]["ProductName"];
      double unitPrice =
         (double) productDetails.Tables[0].Rows[0]["UnitPrice"];
      String insertStatement = "INSERT INTO ShoppingCarts (ProductCode, "
         + "ProductName, Description, UnitPrice, CustomerName, "
         + "Quantity) values ('" + productCode + "', @productName, "
         + "@description, " + unitPrice + ", '" + customerName + "' , 1)";

      SqlConnection myConnection = new SqlConnection(m_ConnectionString);
      SqlCommand myCommand = new SqlCommand(insertStatement, myConnection);
      myCommand.Parameters.Add(
         new SqlParameter("@ProductName", SqlDbType.VarChar, 50));
      myCommand.Parameters["@ProductName"].Value = productName ;

      myCommand.Parameters.Add(
         new SqlParameter("@description", SqlDbType.VarChar, 255));
      myCommand.Parameters["@description"].Value = description;
      myCommand.Connection.Open();
      myCommand.ExecuteNonQuery();
      myCommand.Connection.Close();
   }
}
```

Kao prvo, ono što je u ovom kodu zanimljivo je da on proverava da li se stavka već nalazi na kupovnoj kartici i to tako što poziva `GetShoppingCartItem`, a ako stavka već postoji on samo povećava količinu te stavke i ažurira je u bazi podataka pomoću funkcije `UpdateShoppingCartItem`.

Kad drugo, zanimljivo je da ADO.NET kôd koji dodaje novu stavku na karticu koristi klasu `SqlCommand`. S obzirom na to da opis proizvoda u IBuyAdventure može da sadrži polunavodnike, treba obezbediti da navodnici u opisu ne budu sukobljeni sa navodnicima koji se koriste za ograničavanje polja. Zato za izvršavanje upita koristimo objekat `SqlCommand` pri čemu koristimo parametre u SQL-u, kao što je `@description`, da bi bila izbegnuta sukobljenost. Vrednosti parametara su zatim određene pomoću kolekcija `Parameters` objekta `SqlCommand`:

```
myCommand.Parameters.Add(
   new SqlParameter("@description", SqlDbType.VarChar, 255));
```

Kada je SQL naredba formirana, objekat komande može biti povezan, naredba može biti izvršena a zatim se veza prekida:

```
myCommand.Connection.Open();
myCommand.ExecuteNonQuery();
myCommand.Connection.Close();
```

Prikazivanje kupovne kartice i promena narudžbine

Kupovna kartica omogućava korisnicima da odrede količinu svakog proizvoda na kartici i prikazuje cenu za komad i ukupnu cenu za naručenu količinu. U bilo kom trenutku, korisnik može da izmeni naručene količine i obriše jednu ili više stavki sa kartice tako što će potvrditi polje Remove i pritisnuti dugme Recalculate. Stavka će biti obrisana i ako korisnik upiše nula kao vrednost.

Da bismo implementirali ovu funkcionalnost, koristili smo kontrolu `asp:Repeater`. Implementacija ove funkcionalnosti u pravom ASP-u nije lak posao i zahteva dosta koda. U ASP.NET-u to je mnogo jednostavnije.

Kontrola `asp:Repeater` je korišćena kao osnova za pravljenje kupovne kartice jer kartica ne mora da koristi neku od ugrađenih funkcionalnosti za izbor ili uređivanje koje su obezbedile ostale liste kontrola kao što su `asp:DataList` i `asp:DataGrid`. Sve stavke su uvek potvrđene i obrađene za vreme ponovnog poziva a sadržaj kartice (skup podataka povezan za kontrolom `asp:Repeater`) se uvek generiše za vreme svakog ponovnog poziva.

Kontrola `asp:Repeater` je i bez grafičkog interfejsa ("lookless") (ona generiše samo HTML element koji određujemo pomoću šablona), što se dobro uklapa u dizajn stranice za kupovnu karticu – kontrola ne mora da generiše celu tabelu (početak tabele i redovi zaglavlja su deo statičkog HTML-a).

Izvor podataka/HTML/ASPX za kupovnu karticu

Izvor podataka za kupovnu karticu obezbeđuje komponenta `CartDB` koja je povezana sa kontrolom `myList asp:repeater`:

```
void PopulateShoppingCartList() {

  IBuyAdventure.CartDB cart = new IBuyAdventure.CartDB(getConnStr());
    DataSet ds = cart.GetShoppingCartItems(GetCustomerID());
```

```
          MyList.DataSource = ds;
          MyList.DataBind();
        ...
```

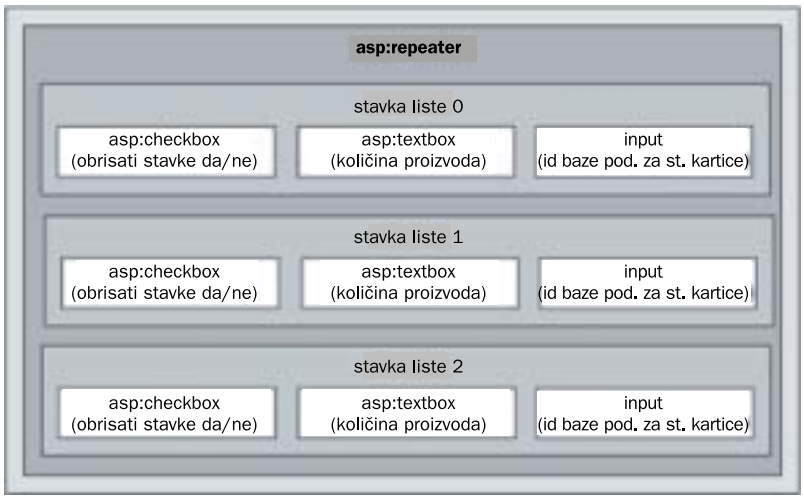
Zatim je prikazan HTML koji je korišćen za predstavljanje kupovne kartice uključujući `ItemTemplate` koji je predstavljen za svaku stavku u `MyList.DataSource`, iako su neki delovi formatiranja HTML stranice (na primer, parametri fontova) obrisani da bi bili kratki i lako čitljivi:

```
<table colspan="8" cellpadding="5" border="0" valign="top">
<tr valign="top">
  <td align="center" bgcolor="#800000">Remove</td>
  <td align="center" bgcolor="#800000">Product Code</td>
  <td align="center" bgcolor="#800000">Product Name</td>
  <td align="center" bgcolor="#800000" width="250">Description</td>
  <td align="center" bgcolor="#800000">Quantity</td>
  <td align="center" bgcolor="#800000">Unit Price</td>
  <td align="center" bgcolor="#800000">Unit Total</td>
</tr>

<asp:Repeater id="MyList" runat="server">

  <itemtemplate>
    <tr>
    <td align="center" bgcolor="#f7efde">
      <asp:checkbox id="Remove" runat="server" />
    </td>
    <td align="center" bgcolor="#f7efde">
      <input id="ShoppingCartID" type="hidden"
 value= '<%#DataBinder.Eval(Container.DataItem,"ShoppingCartID",
"{0:g}")%>'
       runat="server" />
       <%#DataBinder.Eval(Container.DataItem, "ProductCode")%>
    </td>
    <td align="center" bgcolor="#f7efde">
      <%#DataBinder.Eval(Container.DataItem, "ProductName")%>
    </td>
    <td align="center" bgcolor="#f7efde">
      <%#DataBinder.Eval(Container.DataItem, "Description")%>
    </td>
    <td align="center" bgcolor="#f7efde">
      <asp:textbox id="Quantity"
                   text='<%#DataBinder.Eval(
                    Container.DataItem, "Quantity", "{0:g}")%>' width="30"
                   runat="server" />
    </td>
    <td align="center" bgcolor="#f7efde">
      <asp:label id="UnitPrice" runat="server">
        <%#DataBinder.Eval(Container.DataItem, "UnitPrice", "{0:C}")%>
      </asp:label>
    </td>
    <td align="center" bgcolor="#f7efde">
      <%# String.Format("{0:C}",
        (((int)DataBinder.Eval(Container.DataItem, "Quantity"))
       * ((double) DataBinder.Eval(Container.DataItem, "UnitPrice")) )) %>
      </td>
```

```
    </tr>
  </itemtemplate>

</asp:Repeater>

<tr>
  <td colspan="6"></td>
  <td colspan="2" align="right">
  Total is <%=String.Format(fTotal.ToString(), "{0:C}") %>
  </td>
</tr>
<tr>
  <td colspan="8" align="right">
<asp:button text="Recalculate" OnClick="Recalculate_Click" runat="server" />
<asp:button text="Go To Checkout" OnClick="Checkout_Click" runat="server" />
  </td>
</tr>
</table>
```

Prethodno prikazani kôd je sličan onom koji smo ranije videli, pa lako može da se prati. Važno je napomenuti da su sva polja koja treba da budu raspoloživa kada dođe do ponovnog poziva označena pomoću `id` i atributa `runat="server"`. Kada klijent izvrši ponovni poziv tako što pritisne dugme Recalculate, ASP.NET stranica može da pristupi polju za potvrdu kontrole Remove, ID-u kartice u bazi podataka sa skrivenim poljem kontrole i polju kontrole Quantity za svaku stavku liste i da u skladu sa tim ažurira bazu podataka.

Za svaki red u tabeli `ShoppingCarts` za ovog klijenta, kontrola `asp:Repeater` će sadržati stavku liste koja sadrži ove tri kontrole – a kojoj se može pristupiti programski:



Da bi svaka stavka liste u okviru kontrole `asp:Repeater` bila povezana sa određenom stavkom kartice u bazi podataka, biće korišćeno skriveno polje da bi za stavku bio pohranjen jedinstveni ID:

```
<input id="ShoppingCartID" type="hidden"
       value='<%#DataBinder.Eval(
          Container.DataItem, "ShoppingCartID", " {0:g}") %>'
       runat="server">
```

Kao što je prethodno rečeno, sadržaj kupovne kartice se uvek pohranjuje u tabeli SQL servera pod nazivom `ShoppingCarts` i njime se upravlja pomoću poslovnih objekata pod nazivom `CartDB`. Da bi kupovna kartica bila popunjena stavkama, ASP.NET stranica poziva funkciju `PopulateShoppingCartList`. To se događa kod prvog učitavanja stranice (to jest, kada `Page.PostBack` ima vrednost `false`) i nakon svakog ponovnog poziva koji dovodi do izmena u bazi podataka – stavke se dodaju, brišu ili menjaju. Da bi uzela stavke sa kartice i povezala podatke kontrole `asp:Repeater`, ova funkcija koristi metod `GetShoppingCartItems` objekta `CartDB`:

```
        void PopulateShoppingCartList() {

          IBuyAdventure.CartDB cart = new IBuyAdventure.CartDB(getConnStr());

            DataSet ds = cart.GetShoppingCartItems(GetCustomerID());

            MyList.DataSource = ds;
            MyList.DataBind();
            ...
```

Kada je lista povezana, za skup podataka će biti obavljena enumeracija da bi mogla da bude izračunata ukupna vrednost stavki na kartici:

```
            DataTable dt;
            dt = ds.Tables[0];

            int lIndex;
            double UnitPrice;
            int Quantity;

            for ( lIndex =0; lIndex < dt.Rows.Count; lIndex++ ) {

               UnitPrice = (double) dt.Rows[lIndex]["UnitPrice"];

               Quantity = (int) dt.Rows[lIndex]["Quantity"];

               if ( Quantity > 0 ) {
                  fTotal += UnitPrice * Quantity;
               }
            }
        }
```

Ukupni rezultat koji je pohranjen u parametru `fTotal` je prethodno u definiciji stranice definisan kao `Double`:

```
// Ukupan rezultat za korpu
double fTotal = 0;
```

a zatim je referenciran unutar koda koji se izvršava odmah nakon kontrole `asp:Repeater`:

```
...
</asp:repeater>
<tr>
  <td colspan="6"></td><td colspan="2" align="right">
     Total is <%=String.Format("{0:C}", fTotal 0 %>
  </td>
</tr>
...
```

Kada korisnik promeni količinu naručenih proizvoda na kartici ili označi da treba obrisati stavke, on će pritisnuti dugme `Recalculate`. Ovo dugme je stvoreno pomoću kontrole `asp:button` koja ima događaj `OnClick` koji je povezan sa funkcijom `Recalculate_Click`:

```
<asp:button text="Recalculate" OnClick="Recalculate_Click" runat="server" />
```

Funkcija `Recalculate_Click` ažurira bazu podataka na osnovu izmena količina koje napravi korisnik, kao i stavki koje on doda ili obriše. Funkcija zatim uzima ažurirane stavke kartice iz baze podataka, ponovo povezuje kontrolu ponavljanja sa ažuriranim skupom podataka i na kraju stvara poruku o stanju koja obaveštava korisnika koliko stavki (ako postoje) trenutno ima na kartici. Ovim funkcijama je, po redu, poverena nadležnost u okviru procedure za obradu događaja za tri različite funkcije:

```
void Recalculate_Click(Object sender, EventArgs e) {

   // Ažuriraj kupovnu karticu
   UpdateShoppingCartDatabase();

   // Iznova formiraj listu ShoppingCart
   PopulateShoppingCartList();

   // Izmeni poruku stanja
   UpdateSelectedItemStatus();
   }
```

Metod `UpdateShoppingCartDatabase` se prvi poziva u proceduri za obradu događaja, kada povratni podaci za kontrolu `asp:Repeater` koji opisuju karticu, i sve napravljene izmene, postanu dostupni. Funkcija zato može da pristupi ovi povratnim podacima pre nego što bude potrebno da se baza podataka ažurira ili briše. Zatim će, kada se pozove metod `PopulateShoppingCartList` kupovna kartica ponovo biti pročitana iz baze podataka i povezana sa kontrolom `asp:Repeater`. Tako će stranica predstaviti korisniku ažurirani prikaz kartice.

Da bi neophodno ažuriranje baze podataka bilo obavljeno, funkcija `UpdateShoppingCartDatabase` vrši iteraciju kroz svaku od stavki liste (redovi) u kontroli `asp:Repeater` i proverava svaku stavku da bi videla da li je potrebno da bude obrisana ili izmenjena:

```
       void UpdateShoppingCartDatabase() {

          IBuyAdventure.ProductsDB inventory =
             new IBuyAdventure.ProductsDB(getConnStr());
        IBuyAdventure.CartDB cart = new IBuyAdventure.CartDB(getConnStr());
for (int i=0; i<MyList.Items.Count; i++) {
             TextBox quantityTxt =
                (TextBox) MyList.Items[i].FindControl("Quantity");
             CheckBox remove =
```

```
                (CheckBox) MyList.Items[i].FindControl("Remove");
            HtmlInputHidden shoppingCartIDTxt =
          (HtmlInputHidden) MyList.Items[i].FindControl("ShoppingCartID");

            int Quantity = Int32.Parse(quantityTxt.Text);

            if (remove.Checked == true đđ Quantity  == 0)
               cart.DeleteShoppingCartItem(
                  Int32.Parse(shoppingCartIDTxt.Value));
            else {
               cart.UpdateShoppingCartItem(
                  Int32.Parse(shoppingCartIDTxt.Value), Quantity );
         }
      }
```

Ovaj kôd koristi grub pristup jer ažurira svaku stavku na kupovnoj kartici koja nije označena za brisanje. U komercijalnoj aplikaciji bi trebalo da razmislite o skrivenom polju u kome će biti pohranjena prvobitna količina u kojoj će stavke biti ažurirane samo ako se polja sa količinama razlikuju. To potencijalno može značajno da smanji u/i baze podataka ako postoje korisnici koji menjaju naručene količine i brišu stavke. Druga alternativa bi bila da se obrade događaji `OnChange` za kontrole u listi i da se baza podataka ažurira samo kod pozivanja događaja.

Provera obrade i bezbednosti

Kada je korisnik spreman da nabavi robu koja se trenutno nalazi na njegovoj kupovnoj kartici, on može da pritisne dugme Go to Checkout na stranici za kupovnu karticu ili da odabere sliku korpe koja se nalazi na liniji za kretanje. Bezbednosni sistem koji je korišćen u IBuyAdventure ima prednost provere identiteta zasnovane na obrascima (naziva se i bezbednost zasnovana na kolačićima), o čemu je bilo reči u poglavlju 14. Kada korisnik dođe do neke od stranica na kojima je potrebna provera identiteta, ako se već nije prijavio, biće prikazana stranica `login.aspx`:

ASP.NET u fazi izvršavanja zna da prikaže ovu stranicu ako korisnik nije prijavljen zato što se sve stranice za koje je potrebna provera identiteta nalaze u direktorijumu pod nazivom `SECURE`. On sadrži datoteku `web.config` koja određuje da anonimni pristup *nije* dozvoljen:

```
<configuration>
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

Zapamtite da "?" znači "anonimni korisnici".

Korišćenje određenog direktorijuma koji sadrži bezbedne stavke je jednostavan ali i fleksibilan način za implementaciju bezbednosti u ASP.NET aplikaciji. Kada ASP.NET u fazi izvršavanja utvrdi da anonimni korisnik pokušava da pristupi stranici u bezbednom direktorijumu aplikacije, on će znati koju stranicu treba da prikaže jer datoteka `web.config` koja se nalazi u korenom direktorijumu ima element `cookie` sa atributom `loginurl` koji ga određuje:

```
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name=".ibuyadventurecookie" loginUrl="login.aspx"
             protection="All" timeout="60">
      </forms>
    </authentication>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
</configuration>
```

Ova konfiguracija u suštini govori da ako postoji kolačić `.ibuyadventurecookie` koji nije menjan, da je korisniku proveren identitet i on može da pristupi bezbednim pravcima ako je ovlašćen; ako kolačić ne postoji korisnika treba preusmeriti na URL koji određuje atribut `loginurl`.

Provera identiteta zasnovana na obrascima u Web farmama

Da bi provera identiteta zasnovana na obrascima radila u okruženju web farme, atribut `decryptionkey` elementa `cookie` mora biti podešen a ne sme biti prazan ili određen kao podrazumevana vrednost za `autogenerate`. Atribut `decryptionkey` treba da bude jednako podešen na svim računarima u okviru farme. Dužina stringa je 16 znakova za DES šifrovanje (56/64 bita) ili 48 znakova za Triple-DES šifrovanje (128 bita). Ako ipak budete koristili podrazumevanu vrednost nastaće drugačiji string za šifrovanje koji će svaki računar u farmi generisati, tako da sesija provere identiteta neće uspeti na različitim računarima jer korisnik prelazi sa servera na server. Ako se ovo dogodi izuzetak `CryptographicException` će biti ispaljen i korisniku će biti predstavljen ekran na kome se kaže da su podaci neispravni ili da se ne mogu dešifrovati.

Procedure za obradu događaja stranice Login.apsx

Dugme Login na obrascu za prijavljivanje je stvoreno pomoću kontrole `asp:button` koja ima događaj `OnClick` koji je povezan sa procedurom za obradu događaja `LoginBtn_Click`:

```
...
<td colspan="2" align="right">
  <asp:button Text=" Login " OnClick="LoginBtn_Click" runat="server" />
</td>
```

Kada se pritisne dugme pozvana je procedura za obradu događaja `LoginBtn_Click`. Ona proverava ispravnost korisnika i zatim ga preusmerava na prvobitnu stranicu. Kôd za proveru ispravnosti i preusmeravanje je ovde prikazan:

```
    void LoginBtn_Click(Object sender, EventArgs e) {
      IBuyAdventure.UsersDB users = new IBuyAdventure.UsersDB(getConnStr());
       IBuyAdventure.CartDB cart = new IBuyAdventure.CartDB(getConnStr());

       if (users.ValidateLogin(UserName.Text, Password.Text)) {

          cart.MigrateShoppingCartItems(Session.SessionID, UserName.Text);
          FormsAuthentication.RedirectFromLoginPage(
             UserName.Text, Persist.Checked);
       }
       else {
          Message.Text =
             "Login failed, please check your details and try again.";
       }
    }
```

Kôd na početku stvara dva neophodna poslovna objekta pomoću funkcije koda u pozadini `getConnStr` da bi mogli da se prikupe podaci o izvoru podataka sa kojim se treba povezati. Kada je stvoren objekat `UsersDB`, njegov metod `ValidateLogin` je pozvan da bi moglo da se utvrdi da su korisnikova ovlašćenja u redu (podaci o korisniku su pohranjeni u tabeli `Account` a ne u datoteci `web.config`). Ako su podaci neispravni, svojstvo `Text` kontrole `Message` je ažurirano da bi prikazalo grešku. Ako je prijavljivanje uspešno, doći će do sledećih koraka:

1. Pozivanjem metoda `RedirectFromLoginPage` objekta `FormsAuthentication` o kome je bilo reči u poglavlju 14 je označeno da je korisniku proveren identitet.
2. Tako se kolačić pod nazivom `.ibuyadventurecookie` vraća korisniku, tako da sada znamo da je korisniku proveren identitet.
3. Korisnik je preusmeren na stranicu koja je prva prikazala obrazac za prijavljivanje.

Ako je korisnik ranije registrovan, on može da se prijavi preko stranice Login. Tako će korisnici biti preusmereni nazad na prvobitnu stranicu sa koje je pozvana stranica Login. Ovaj kôd za preusmeravanje u stvari implementira stranica Login koju smo stvorili a koja zahteva dodatne izmene. Sledeće što ćemo videti je taj kôd.

Povratak na stranicu za vreme provere identiteta

Kada ASP.NET u fazi izvršavanja utvrdi da je pristupljeno bezbednoj stavki, on će preusmeriti korisnika na stranicu Login i uključiti parametar stringa upita pod nazivom `ReturnURL`. Kao što ime govori, ovo je stranica na koju ćemo preusmeriti korisnika kada budemo mogli da mu odobrimo pristup. Kada je ova stranica prikazana moramo da sačuvamo ovu vrednost jer će biti izgubljena za vreme ponovnih poziva kada se proverava ispravnost korisnika. Pristup koji je korišćen na ovoj stranici je pohranjivanje vrednosti u skrivenom polju za vreme događaja `Page_Load`:

```
void Page_Load(Object sender, EventArgs e)
{
   // Store Return Url in Page State
   if (Request.QueryString["ReturnUrl"] != null)
   {
      ReturnUrl.Value = Request.QueryString["ReturnUrl"];
         ((HyperLink)RegisterUser).NavigateUrl =
            "Register.aspx?ReturnUrl=" + ReturnUrl.Value;
   }
}
```

Skriveno polje je definisano kao deo obrasca `Login` i sadrži atribut `runat="server"` tako da programski možemo da mu pristupimo u procedurama za obradu događaja:

```
<input type="hidden" value="/advworks/default.aspx"
       id="ReturnUrl" runat="server" />
```

Skrivenom polju smo dodelili podrazumevanu vrednost, jer korisnik može direktno da ode na stranicu za prijavljivanje preko linije za kretanje. Bez podrazumevane vrednosti, kôd za preusmeravanje koji se izvršava nakon prijavljivanja neće raditi.

Dakle, kada korisnik pritisne dugme Login tada možemo da proverimo ispravnost njegovih podataka a zatim da ga preusmerimo na stranicu čija vrednost je pohranjena u skrivenoj kontroli `ReturnUrl`.

Korisnik koji se prvi put pojavljuje – Registracija

Ako korisnik nije ranije registrovan u aplikaciji, on može da odabere hipervezu Registration i dobiće obrazac za registraciju korisnika koji treba da popuni:

Obrazac za ovaj praktičan primer je jednostavan i u njemu se traži samo elektronska adresa i lozinka. U komercijalnoj aplikaciji ovaj obrazac bi verovatno sadržao dodatne informacije kao što je ime i adresa korisnika.

Kada se otvori stranica za registraciju (`Register.aspx`) nakon što odaberemo hiperveze na stranici za prijavljivanje (`login.aspx`), treba obezbediti prosleđivanje parametra `ReturnUrl` – tako da stranica za registraciju zna kuda da usmeri korisnika kada on popuni obrazac. Da bismo to uradili dinamički ćemo stvoriti hipervezu u obrascu za registraciju za vreme događaja `Page_Load` na stranici za prijavljivanje:

```
((HyperLink)RegisterUser).NavigateUrl =
   "Register.aspx?ReturnUrl=" + ReturnUrl.Value;
```

Trebalo bi da proverimo i da li je hiperveza označena kao serverska kontrola na stranici `logging.aspx`:

```
...
<font size="2">
<asp:HyperLink NavigateUrl="Register.aspx" id="RegisterUser"
               runat="server" />
  Click Here to Register New Account
</asp:hyperlink>
</font>
...
```

Pažljiviji posmatrači će uočiti da korisnici mogu da se prijave u bilo kom trenutku tako što će odabrati hipervezu Sign in or Register koja se nalazi u zaglavlju stranice. Kada je uspešno proveren identitet korisnika, ova hiperveza prelazi u Sign Out:

Kôd za prijavljivanje ili odjavljivanje je implementiran u zaglavlju User Control (`UserControl/header.ascx`) u kome procedura za obradu događaja `Page_Load` dinamički menja tekst kontrole `signInOutMsg` u zavisnosti od stanja provere identiteta trenutnog korisnika:

```
<%@ Import Namespace="System.Web.Security" %>
<script language="C#" runat="server">

   private void Page_Load( Object Sender, EventArgs e ) {
      updateSignInOutMessage();
   }

   private void SignInOut( Object Sender, EventArgs e ) {

      if ( Context.User.Identity.Name != "" ) {
         IBuyAdventure.CartDB cart =
            new IBuyAdventure.CartDB(
               ConfigurationSettings.AppSettings["connectionString"]);
         cart.ResetShoppingCart(GetCustomerID());
         FormsAuthentication.SignOut();
         Response.Redirect("/IBuyAdventure/default.aspx");
      }
      else {
         Response.Redirect("/IBuyAdventure/login.aspx");
      }
   }
private void updateSignInOutMessage() {

      if  ( Context.User.Identity.Name != "" ) {
```

```
            signInOutMsg.Text = "Sign Out (" + Context.User.Identity.Name+ ")";
        }
        else {
           signInOutMsg.Text = "Sign In / Register";
        }

    }
</script>
...
```

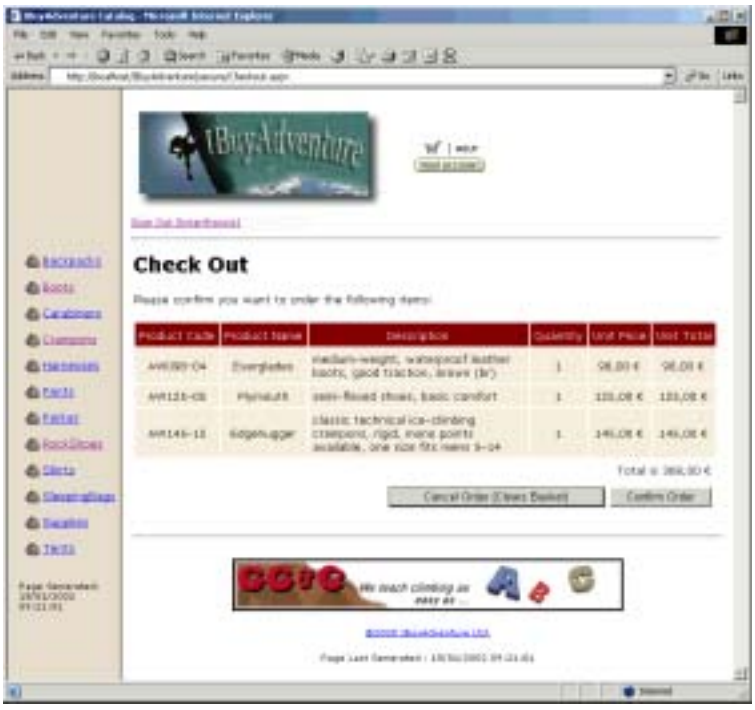
Funkcija `updateSignInOutMessage` obično ažurira tekst, a metod `SignInOut` se poziva kada korisnik odabere tekst sign in/out. Ako se korisnik odjavljuje, funkcija `CookieAuthentication.SignOut` se poziva da bi kolačić za proveru identiteta bio nevažeći. Kod prijavljivanja, korisnik je preusmeren na stranicu za prijavljivanje.

Kôd za `SignInOut` je povezan kao deo deklaracije kontrole:

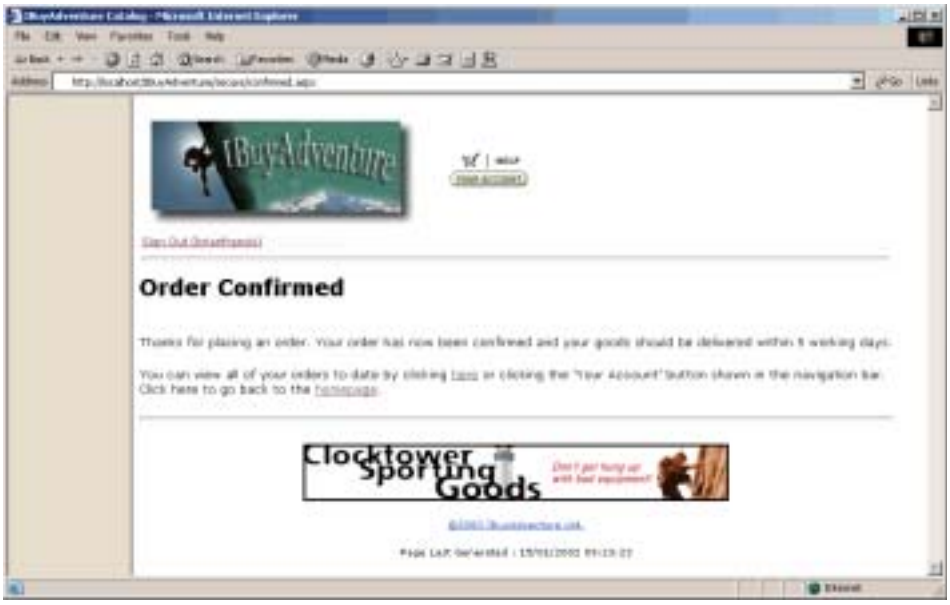
```
...
<td>
  <asp:linkbutton style="font:8pt verdana" id="signInOutMsg"
                  runat="server" OnClick="SignInOut" />
</td>
...
```

Provera obrade

Kada je korisniku proveren identitet, on prelazi na stranicu za proveru (`secure/checkout.aspx`), koja je predstavljena uz listu za kupovinu i koja traži da se potvrdi ispravnost liste:



Stranica za proveru koristi kôd koji je veoma sličan onom na stranici `ShoppingCart.aspx` osim ako izostavimo kontrole koje omogućavaju korisniku da obriše stavke ili da uredi količine. Ako korisnik potvrdi narudžbinu tako što pritisne dugme Confirm Order, biće stvoren novi zapis u bazi podataka za narudžbinu koja sadrži datum i ukupnu vrednost narudžbine. Tada će biti obrisana trenutna korpa a korisniku će biti prikazan ekran sa potvrdom:



Sledi kôd koji se poziva kod potvrđivanja narudžbine:

```
void Confirm_Order(Object sender, EventArgs e) {

   IBuyAdventure.CartDB cart = new IBuyAdventure.CartDB(getConnStr());
   double totalOrderValue;

   totalOrderValue = cart.GetOrderValueForCart(GetCustomerID());

   IBuyAdventure.OrdersDB orders = new
IBuyAdventure.OrdersDB(getConnStr());
   orders.AddNewOrder(GetCustomerID(), DateTime.Now.ToString("G",
                    DateTimeFormatInfo.InvariantInfo), totalOrderValue );

   cart.ResetShoppingCart( GetCustomerID() );
   Response.Redirect("confirmed.aspx");
}
```

Ukupna vrednost narudžbine je izračunata pomoću funkcije `GetOrderValueForCart` objekta `CartDB`. Ime korisnika koje je prosleđeno u ovu funkciju a koja je vraćena pozivanjem `GetCustomerID` će uvek biti ime koje je korisnik upisao pri registraciji, jer se stranici ne može pristupiti bez provere identiteta.

Kada se izračuna ukupna vrednost narudžbine funkcija `AddNewOrder` objekta `OrdersDB` se poziva da stvori stavku u tabeli `orders`. S obzirom na to da ćemo u bazu podataka dodati datum i vreme narudžbine, datum bi trebalo da bude u poznatom formatu – rutine za standardno formatiranje uzimaju u obzir lokaciju servera, pa bismo mogli da završimo sa formatom datuma koji SQL Server ne prepoznaje.

Da bismo ovo prevazišli koristićemo višeznačnu verziju metoda `ToString()`. Obično ovaj metod nema parametre – ali u ovom slučaju ćemo koristiti dva. Prvi određuje da želimo da generišemo format datuma i vremena. `DateTimeFormatInfo.InvariantInfo` je statički objekat koji obezbeđuje da string datuma bude formatiran na isti način bez obzira na lokaciju servera. Na kraju, sadržaj kupovne kartice je obrisan iz baze podataka a čitač je preusmeren na ekran za potvrdu narudžbine.

U komercijalnoj aplikaciji ćete želeti da sačuvate sadržaj kupovne kartice tako da ćete u stvari obraditi narudžbinu. S obzirom na to da je ovo samo jednostavan prikaz aplikacije to ovde nije prikazano.

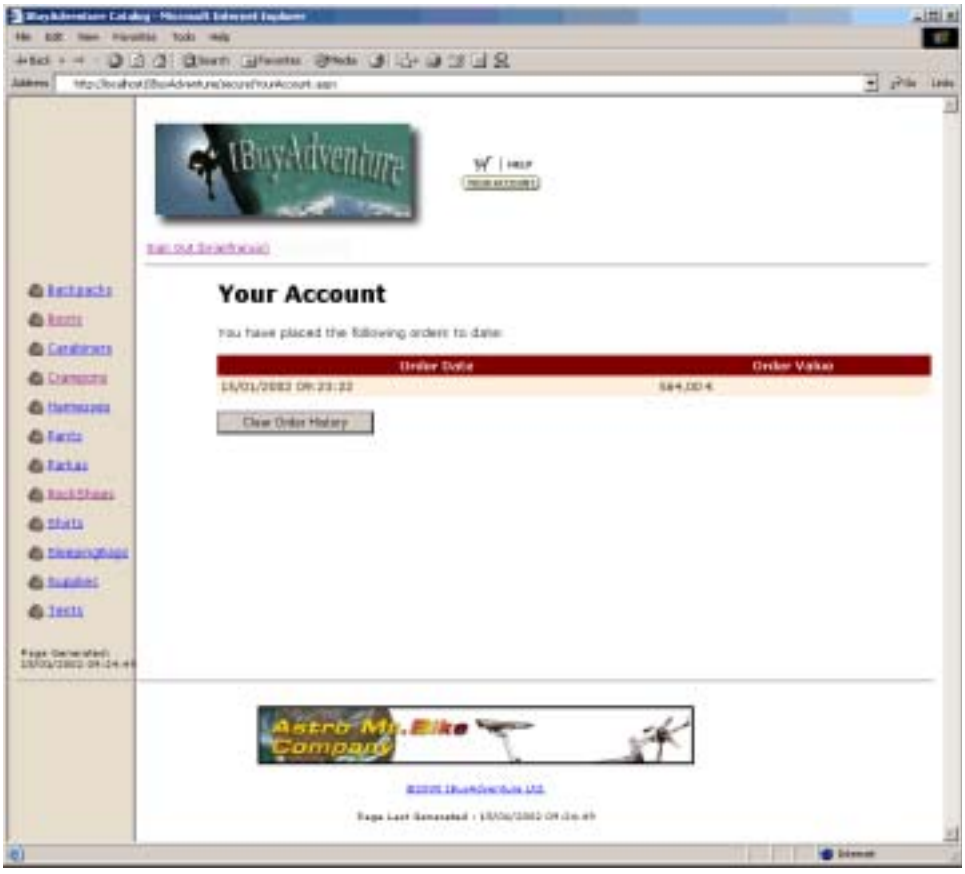
Otkazivanje narudžbine

Ako je narudžbina otkazana pre nego što je završena, trenutna kupovna kartica će biti obrisana a korisnik će biti vraćen na matičnu stranicu IBuyAdventure (`default.aspx`). Kôd za otkazivanje narudžbine je ovde prikazan:

```
void Cancel_Order(Object sender, EventArgs e) {

   IBuyAdventure.ProductsDB inventory =
      new IBuyAdventure.ProductsDB(getConnStr());
   IBuyAdventure.CartDB cart = new IBuyAdventure.CartDB(getConnStr());
   cart.ResetShoppingCart( GetCustomerID() );

   Response.Redirect("/IBuyAdventure/default.aspx");
}
```

Pregled narudžbina i vaš račun

Korisnici mogu da pogledaju pregled svojih narudžbina u bilo kom trenutku tako što će odabrati sliku Your Account koja se nalazi na vrhu svake stranice. Kada je odaberete, ova stranica će prikazati sve prethodne narudžbine do određenog datuma, pokazujući datum kada je napravljena narudžbina kao i ukupnu vrednost narudžbine:



Ova stranica je generisana pomoću kontrole `asp:Repeater` i jednostavno prikazuje stavke u tabeli `Orders` za trenutnog korisnika. Funkcija `PopulateOrderList` povezuje podatke kontrola kao i na prethodnoj stranici:

```
void PopulateOrderList() {

   IBuyAdventure.OrdersDB orders = new
IBuyAdventure.OrdersDB(getConnStr());

   DataSet ds = orders.GetOrdersForCustomer(GetCustomerID());

   MyList.DataSource = ds;
   MyList.DataBind();

   if ( MyList.Items.Count == 0 ) {
      ClearButton.Visible = false;
      MyList.Visible = false;
      Status.Text = "No orders have been placed to date.";
   }
}
```

Poslednjih nekoliko redova koda kriju dugme Clear Order History ako ne postoje narudžbine korisnika. Ako narudžbine postoje, kada se pritisne ovo dugme biće pozvana funkcija `ClearOrderHistory`:

```
void ClearOrderHistory(Object sender, EventArgs e) {

   IBuyAdventure.OrdersDB orders = new
IBuyAdventure.OrdersDB(getConnStr());
   orders.DeleteOrdersForCustomer( GetCustomerID() );
   PopulateOrderList();
}
```

Ovaj kôd briše sve narudžbine korisnika tako što poziva funkciju `DeletesOrdersForCustomer` koju obezbeđuje objekat `OrdersDB`.

Rezime

To je to, vaša prva ASP.NET aplikacija za elektronsku trgovinu!

U ovom poglavlju smo videli kako je čisto i jednostavno pisanje aplikacije elektronske trgovine pomoću ASP.NET-a. Funkcionalno bogata kontrola na strani servera i model događaja čini razvoj ASP.NET-a mnogo sličnijim tradicionalnom VB programiranju zasnovanom na događajima, što značajno smanjuje količinu koda koji treba da napišemo na stranicama.

Nije bilo prostora da u ovom poglavlju obuhvatimo svako svojstvo koje biste mogli da koristite u aplikaciji. Međutim kôd koji je ovde prikazan srećom obuhvata najčešće aspekte aplikacije, a kada se kombinuje sa informacijama koje su obezbeđene u drugim poglavljima onda pruža mogućnost da napravite sopstvene aplikacije.