TDA-LABORATOR 4

Structura unei aplicatii ASP.NET. Ciclul de viata al aplicatiei ASP.Net

ASP.NET este o platformă de dezvoltare aplicatii web,inclusa in platforma Microsoft .Net ,care oferă un model de programare, o infrastructură software cuprinzătoare și diverse servicii necesare pentru a construi aplicații web robuste pentru calculatoare și dispozitive mobile.

Aplicatiile pot utiliza întreaga ierarhie a claselor din Framework-ul .Net. ASP.NET web forms extinde modelul event-driven de interacțiune a aplicațiilor web. Browser-ul trimite un web form la serverul web și serverul returnează o pagina HTML ca răspuns. Toate actiunile utilizatorului pe partea de client sunt transmise inapoi la server. Serverul procesează datele si acțiunile clientului și declanșează reacțiile corespunzatoare.

Deoarece, HTTP este un protocol fara stare, frameworkul ASP.NET ofera pentru stocarea informațiilor obiecte ca ViewState, Page State, Session State sau Aplication State.

Modelul componente ASP.NET este un model care descrie obiecte corespondente pe partea de server pentru aproape toate elementele HTML, cum ar fi <form> și <input> si pentru controale Server, pentru descrierea interfeței utilizator.

ASP.NET este o tehnologie, care funcționează pe .NET Framework, care conține toate funcționalitățile web. O aplicație web ASP.NET este formata din pagini server. Runtime-ul ASP.NET transformă pagina .aspx într-o instanță a unei clase, care moștenește clasa page, clasă de bază .Net. Prin urmare, fiecare pagină ASP.NET este un obiect și toate componentele sale, adică controalele server-side, sunt de asemenea obiecte.

In acest model adaugam controale la formularul web si apoi tratam evenimentele. Modelul este asemanator cu cel dat de Windows Forms, dar lucreaza diferit.

Aplicatiile web se executa pe server. Cerintele clientului sunt procesate pe server folosind o tehnica numita *postback*, care trimite pagina pe server cand sunt indeplinite anumite conditii. Dupa ce primeste pagina, ASP.NET genereaza evenimentele pe partea de server pentru a notifica codul (a apela metodele atasate evenimentelor).

Formulare HTML

In **HTML** forma cea mai simpla de a trimite date de la client la server o constituie utilizarea formularelor web (tag <form>). Un formular web poate contine mai multe controale.

Exemplu

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Titlul Pagini</title>
</head>
<body>
```

```
        <form method="post" action="page.aspx">
        <div>
                Enter your first name: <input type="text" name="FirstName" />
                <br />
                Enter your last name: <input type="text" name="LastName" />
                <br /><br />
                <input type="submit" value="Submit" id="OK" />
        </div>
        </form>
</body>
</html>
```

Modelul Event in ASP.NET. In acest model adaugam controale la formularul web si apoi decidem ce evenimente sa tratam. Modelul este asemanator cu cel dat de Windows Forms, dar lucreaza diferit. Putem distinge urmatoarele faze in tratarea unei pagini:

1. Pagina ruleaza pentru prima data.

ASP.NET creaza pagina si controalele din pagina, executa codul de initializare si apoi creaza pagina HTML care este trimisa clientului. Obiectul Page si controalele pe care le contine sunt eliminate din memorie (sunt distruse).

2. Clientul executa actiuni asupra paginii (completare controale din pagina,se fac validari pe partea de client etc.) ce au ca efect un postback. Pagina este trimisa cu toate datele din form catre server.

3. ASP.NET intercepteaza pagina returnata si recreaza obiectele din pagina la starea pe care au avut-o ultima data.

4. ASP.NET verifica cine a generat postback si apeleaza metoda corespunzatoare acelui eveniment (de ex. clic pe un buton). In acest moment se executa cod pe server (validare pe partea de server actualizare baze de date, etc.) si apoi se modifica obiectele controalelor pentru a reflecta noua informatie.

5. Pagina este transformata in HTML, trimisa clientului si apoi distrusa din server. Daca pe aceeasi pagina apare un nou postback se reia ciclul de la psul 2.

Cicluri de viață ASP.NET. Ciclurile de viață ASP.NET sunt foarte importante pentru dezvoltatori pentru ca: evidentiaza evenimentele produse pe durata ciclului de viață și descrie modul în care codul scris de dezvoltator poate fi integrat în ciclul de viață al aplicatiei pentru a obtine efectul dorit. Ciclu de viață ASP.NET specifică modul în care ASP.NET procesează Aplicația și paginile sale. Ciclu de viață ASP.NET poate fi împărțit în două grupe: Ciclul de viață al Aplicației si Ciclul de viață al Paginii.

Ciclul de viata al aplicatiei. Evenimentele solicită metode speciale pe care le puteți suprascrie in fișierul **Global.asax** din directorul rădăcină al aplicației. ASP.NET leagă în mod automat

evenimentele aplicației de metode pentru manipulare în fișierul Global.asax utilizând convenția de denumire **Application_event**, cum ar fi Application_BeginRequest.

Metodele Application_Start și Application_End sunt event-handlere speciale, care nu reprezintă evenimente HttpApplication. ASP.NET le solicită o dată pe durata de viață a aplicației, nu pentru fiecare instanță HttpApplication.

Cele mai importante event-handlere ale aplicatiei sunt:

Application_Start -Apare atunci când porniți aplicația

Application_End -Apare atunci când opriti aplicația

Application_Error -Apare când se produce o eroare in aplicație

Evenimentele sesiunii cele mai utilizate sunt:

Session_Start -Apare la solicitarea primei pagini.

Session_End -Apare la sfarsit de sesiune.

Ciclul de viata al paginii. Atunci când se solicită o pagină, aceasta este încărcata de catre server, procesata și trimisa către browser. Apoi este stearsa din memorie. Aceste actiuni se fac in mai multe etape de catre un cod implicit. La fiecare dintre aceste etape, metodele și evenimentele sunt disponibile, si codul implicit poate fi suprascris în funcție de necesitățile aplicației. Fazele ciclului de viață al paginii sunt: Inițializarea, Instanțierea controalelor de pe pagina, Restaurarea și întreținerea starii, Executarea codului event handler si Construirea paginii HTML.

La fiecare etapă a ciclului de viață, o pagină emite evenimente, care pot fi tratate. O tratare a unui eveniment este de fapt o metoda legata la eveniment. Evenimentele ciclului de viață al unei pagini:

- **PreInit**
- **Init**
- **InitComplete**
- **PreLoad**
- **Load**
- **Controls events**
- **LoadComplete**
- **PreRender**
- **PreRenderComplete**

- **Render**
- **SaveStateComplete**
- **UnLoad**

Exemplu de testare a ordinii de tratare a evenimentelor unei pagini

```
<%@ Page language="C#" CodeFile="PageFlow.aspx.cs"
        AutoEventWireup="true" Inherits="PageFlow" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Page Flow</title>
</head>
<body>
        <form id="form1" runat="server">
        <div>
                <asp:Label id="lblInfo" runat="server"
                        EnableViewState="False">
                </asp:Label>
                        <asp:Button id="Button1" runat="server" Text="Button">
                </asp:Button>
        </div>
        </form>
</body>
</html>
```

In etapa urmatoare putem adauga metode pentru evenimentele din ciclul de viata al paginii: Page.Init, Page.Load, Page.PreRender, Page.Unload, si Button.Click, etc.

```
private void Page_Load(object sender, System.EventArgs e)
{
lblInfo.Text += "Page.Load event handled.<br />";
if (Page.IsPostBack)
{
    lblInfo.Text +="<b>This is not the first time you've seen this page.</b><br/>";
}

}

// Codul pentru evenimentul clic al butonului:
protected void Button1_Click(object sender, System.EventArgs e)
```

```
{
   lblInfo.Text += "Button1.Click event handled.<br />";
}

private void Page_Init(object sender, System.EventArgs e)
{
lblInfo.Text += "Page.Init event handled.<br />";
}

private void Page_PreRender(object sender, System.EventArgs e)
{
lblInfo.Text += "Page.PreRender event handled.<br />";
}
```

Controale. Pagina este un container pentru controalele pe care le contine

Toate controalele ASP.NET sunt instante ale unor clase, iar acestea au evenimente care sunt declanșate când un utilizator efectuează o anumită acțiune pe ele.

De exemplu, atunci când un utilizator face clic pe un buton este generat evenimentul "Click".

Pentru manipularea evenimentelor, există event handlere.

Tratarea evenimentului se face prin executarea codului scris in event handlerul corespunzator.

În mod implicit, Visual Studio creează o tratare a evenimentului, prin includerea unui event handler. Aceast event handler are numele de forma <control>_< eveniment>.

Exemplu:

Pentru butonul

<asp:Button ID="btnCancel" runat="server" Text="Cancel" />

avem event handlerul
protected void btnCancel_Click(object sender, EventArgs e)

Controalele server sunt clase in .NET Framework ce reprezinta elemente vizuale pe un formular web.

Tipuri de controale server:

Controale server HTML: clase ce implementeaza elementele standard HTML. Pentru a schimba un element ordinar HTML intr-un control server trebuie adaugat atributul **runat=”server”.**

Controale server Web: sunt ca si HTML dar mai usor de folosit.

Controale avansate: genereaza mai mult markup HTML.

Exemple: controalele **Calendar, AdRotator si TreeView**.

Controale de validare: folosite in validarea datelor din alte controale din cadrul paginii.

Controale pentru date: afiseaza o cantitate mare de date (grid, list). Suporta editare, sortare si paginare.

Controale de navigare: navigare de la o pagina la alta.

Controale Login: pentru autentificare la o baza de date.

Controalele server sunt derivate din clasa **Control(**in **System.Web.UI)**.

public class Control : IComponent, IDisposable, IParserAccessor, IUrlResolutionService, IDataBindingsAccessor,

Proprietati importante ale clasei Control:

ClientID - returneaza identificatorul clientului, identificator creat de ASP.NET.

Controls – returneaza colectia de controale descendente. Colectia Page.Controls contine controalele din pagina.

EnableViewState - controlul isi memoreaza starea dacă este true

ID – ID control. **Utilizat pentru referirea controlului!**

Page – returneaza o referinta la obiectul pagina ce contine controlul.

Parent – returneaza o referinta la parintele controlului.

Visible : bool (true, false).

DataBind() Asociaza controlul si toate controalele descendente la sursa de date specificata.

FindControl() – cauta control dupa nume.

Controale server Web sunt derivate din Clasa de baza WebControl derivata din Control.

public class WebControl : Control, IAttributeAccessor

Exemplu de Proprietati: **BackColor, BorderColor, BorderStyle, BorderWidth, CssClass, Enabled**

Sintaxă generală controale server web este:

<asp:control_name id="some_id" runat="server“... />

Corespondență controle server web - server HTML este :

<asp:Button> <input type="submit"/> <input type="button"/>

Proprietati admise: Text, CausesValidation, PostBackUrl,ValidationGroup, Click event

<asp:CheckBox> **☐**

Proprietati admise: AutoPostBack, Checked, Text,TextAlign, CheckedChanged event

<asp:FileUpload> **<input type="file">**

Proprietati admise: FileBytes, FileContent, FileName,HasFile, PostedFile, SaveAs()

Text si buton pentru a selecta fisier upload

<asp:HiddenField> **<input type="hidden">**

Proprietati admise: Value

Memoreaza o valoare care nu se afisaza

<asp:HyperLink> **<a>...**

Proprietati admise: ImageUrl, NavigateUrl, Target, Text

Afisaza un link la alta Web page.

<asp:Image> **<img/>**

Proprietati admise: AlternateText, ImageAlign, ImageUrl

<asp:ImageButton> **<input type="image"/>**

Proprietati admise: CausesValidation, ValidationGroup,Click event

<asp:ImageMap> **<map>**

Proprietati admise: HotSpotMode, HotSpots,AlternateText, ImageAlign, ImageUrl

<asp:Label> **...**

Proprietati admise: Text, AssociatedControlID

<asp:LinkButton> **<a><img/>**

Proprietati admise: Text, CausesValidation, Validation-Group,Click event

<asp:Panel> ** ... **

Proprietati admise: BackImageUrl, DefaultButton,GroupingText, HorizontalAlign,Scrollbars, Wrap

<asp:RadioButton> **☐**

Proprietati admise: AutoPostBack, Checked, GroupName,Text, TextAlign, CheckedChanged

<asp:Table> **<table>...</table>**

Proprietati admise: BackImageUrl, CellPadding,CellSpacing, GridLines,HorizontalAlign, Rows

<asp:TableCell> <td>...</td>

Proprietati admise: ColumnSpan, HorizontalAlign,RowSpan, Text, VerticalAlign, Wrap

<asp:TableRow> <tr>...</tr>

Proprietati admise: Cells, HorizontalAlign,VerticalAlign

<asp:TextBox> <input type="text"/> sau <textarea>...</textarea>

Proprietati admise: AutoPostBack, Columns, MaxLength,ReadOnly, Rows, Text, TextMode, Wrap,TextChanged

Exemple:

Controale

<asp:TextBox runat="server" ID="TextBox1" Text="This is a test“ ForeColor="red" BackColor="lightyellow" Width="250px"

Font-Name="Verdana" Font-Bold="True" Font-Size="20" />

Dimensiuni

<asp:Panel Height="300px" Width="500" id="pnl" runat="server" />

Culori

<asp:TextBox ForeColor="Red" Text="Test" id="txt" runat="server" />

<asp:TextBox ForeColor="#ff50f" Text="Test"

id="txt" runat="server" />

Focus

<form id="Form1" DefaultFocus="TextBox2" runat="server">

Butonul implicit executa metoda atasata evenimentului “click” la apasarea tastei Enter.

Este la nivel de pagina web, iar in cadrul paginii poate fi la nivel de Panel.

In <form> se foloseste proprietatea DefaultButton in care se specifica ID-ul controlului.

<form id="Form1" DefaultButton="cmdSubmit" runat="server">

Butonul implicit trebuie sa fie un control ce implementeaza interfata **IButtonControl**.

Controalele web **Button, IinkButton** si **ImageButton** implementeaza aceasta interfata.

Controalele server HTML nu implementeaza **IButtonControl**.

Controlul Panel are posibilitatea de scroll. Se foloseste proprietatea :

ScrollBars = {Vertical, Horizontal, Both, Auto}.

Both= Vertical și Horizontal

Auto= Vertical și/sau Horizontal dacă este necesar

<asp:Panel ID="Panel1" runat="server" Height="116px" Width="278px“ BorderStyle="Solid" BorderWidth="1px" ScrollBars="Auto">

**

**

<asp:Button ID="Button1" runat="server" Text="Button" />

<asp:Button ID="Button2" runat="server" Text="Button" />

**
**

...

</asp:Panel>

Evenimentul clic pe acest buton trimite in al doilea parametru un obiect ImageClickEventArgs ce are proprietatile X si Y (coordonatele unde s-a facut clic pe imagine). Se pot identifica astfel puncte de pe o imagine .

protected void ImageButton1_Click(object sender,System.Web.UI.ImageClickEventArgs e)

{

lblResult.Text = “Punctul (" + e.X.ToString() + ", " + e.Y.ToString() + "). ";

// Verificare daca sa dat clic in patratul : (20,20) si (275,100).

if ((e.Y < 100) && (e.Y > 20) && (e.X > 20) && (e.X < 275)){

lblResult.Text += “Ai apasat pe buton.";

}else{

lblResult.Text += " Ai apasat pe bordura butonului";

}}

Controalele List. Afiseaza o colectie de obiecte.

<asp:DropDownList> contine o colectie de obiecte <asp:ListItem>

In HTML, tag <select> cu atributul size="1".

<asp:ListBox> contine o colectie de obiecte <asp:ListItem>.

In HTML, tag <select> cu atributul size="x" unde x este numarul de articole vizibile.

<asp:CheckBoxList> - aliniate intr-o tabela.

<asp:RadioButtonList>

<asp:BulletedList> o lista statica ordonata sau nu.

In HTML, tag <ul> sau <ol>.

Se poate folosi si pentru a crea o lista de hyperlink-uri.

Membri:

AutoPostBack = true => actioneaza la schimbarea selectiei curente.

Items – returneaza o colectie de articole ListItem.

SelectedIndex – returneaza sau seteaza indexul articolului selectat. Pentru liste ce suporta selectia multipla trebuie scanata intreaga colectie pentru a determina articolele selectate (verificare proprietate SelectedItem).

SelectedItem – returneaza o referinta la prmul articol ListItem selectat.

DataSource - se seteaza la un obiect ce contine informatii ce dorim sa le afisam (DataSet,DataTable sau colectii). Cand se apeleaza DataBind() lista va fi completata cu obiecte din acea colectie.

DataMember – se foloseste impreuna cu asocierea de date, cand sursa de date contine mai mult de o tabela. DataMember indica ce tabela se va folosi.

DataTextField – indica ce proprietate sau camp din sursa de date va fi folosit ca text pentru articol.

DataValueField – indica ce proprietate sau camp va fi folosit ca valoare pentru articol (nu e afisat dar poate fi citit in cod).

DataTextFormatString – seteaza formatarea stringului folosit pentru redarea articolului.

Eveniment: SelectedIndexChanged – la schimbarea selectiei.

Exemplu:

```
<form id="form1" runat="server">
    <div>
    <asp:ListBox runat="server" ID="Listbox1" SelectionMode="Multiple“ Rows="5">
        <asp:ListItem Selected="true">Option 1</asp:ListItem>
```

```
            <asp:ListItem>Option 2</asp:ListItem>
        </asp:ListBox>
        <br /><br />
        <asp:DropDownList runat="server" ID="DropdownList1">
            <asp:ListItem Selected="true">Option 1</asp:ListItem>
            <asp:ListItem>Option 2</asp:ListItem>
        </asp:DropDownList>
        <br /><br />
        </div>
</form>
```

Controale de validare. Validarea se realizeaza atat pe partea de client cat si pe partea de server. Controalele de validare se declara in <form> si sunt atasate controalelor de intrare. In acest mod nu mai e necesar sa scriem cod pentru validare atat pe partea de client cat si pe cea de server. Daca procesul de validare nu este trecut cu succes, pagina nu este trimisa serverului sau nu este retransmisa clientului.

Se pot folosi mai multi validatori pentru acelasi camp.

<asp:RequiredFieldValidator> - câmpul nu trebuie sa fie vid cand pagina este trimisa serverului.

<asp:RangeValidator> - controleaza daca valorile asociate controlului sunt in intervalul specificat.

<asp:CompareValidator> - controleaza daca valoarea asociata controlului se potriveste prin comparatie cu o valoare din alt control sau o constanta.

<asp:RegularExpressionValidator> - controleaza daca valoarea specificata se potriveste cu expresia regulata specificata.

<asp:CustomValidator> - validator personalizat. Cod JavaScript pe partea de client, cod C# pe partea de server.

<asp:ValidationSummary> - arata o sinteza a erorilor pentru fiecare camp.

În continuarea acestui laborator, vom urmarii procesul de creare a unei aplicații web ASP.NET numită **SportsStore** specifica magazinelor online. Vom crea un catalog de produse online, prin care utilizatorii pot naviga si un coș de cumpărături în care utilizatorii pot adăuga și elimina produse din catalog.

Vom crea un nou proiect Visual Studio numit SportsStore folosind șablonul empty al Aplicației web ASP.NET. Vom începe prin configurarea unui model de date și a unei baze de date și apoi adăugarea celorlalte funcții de care avem nevoie.

Crearea structurii de foldere. Vom crea o structura de directoare asemanatoare cu cea care se foloseste în proiectele reale. Descriem pe scurt fiecare dintre folderele pe care le-am creat.

App_Start- Acest folder va conține în mod convențional clase care efectuează configurația inițială a aplicației la pornirea acesteia. Folosim acest folder atunci când configurăm rutarea URL.

Content - Acest folder va fi folosit în mod conventional pentru conținutul static, cum ar fi CSS.

Controls- Acest folder va conține controalele utilizatorului nostru.

Models- Acest folder va conține clasele model de date.

Models / Repository- Acest folder va conține clasele pe care le folosim pentru a implementa un depozit persistent pentru clasele noastre de model de date.

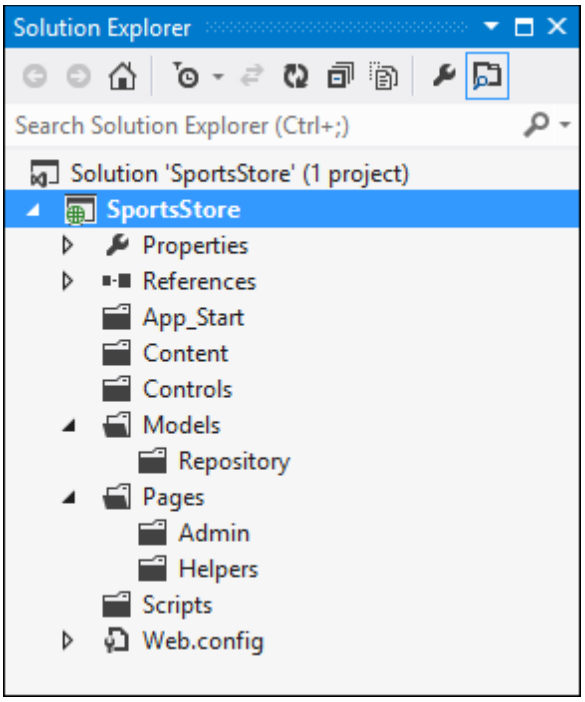
Pages - Acest folder va conține fișierele noastre Web Form.

Pages / Admin - Acest folder va contine fișierele Web Form pentru funcțiile de administrare SportsStore.

Pages / Helpers - Acest folder va fi utilizat pentru a conține clasele utilizate în Formularele Web.

Scripts -Acest folder va conține fișierele JavaScript.

Prin urmare solutia noastra **SportsStore** ar trebui să arate astfel:

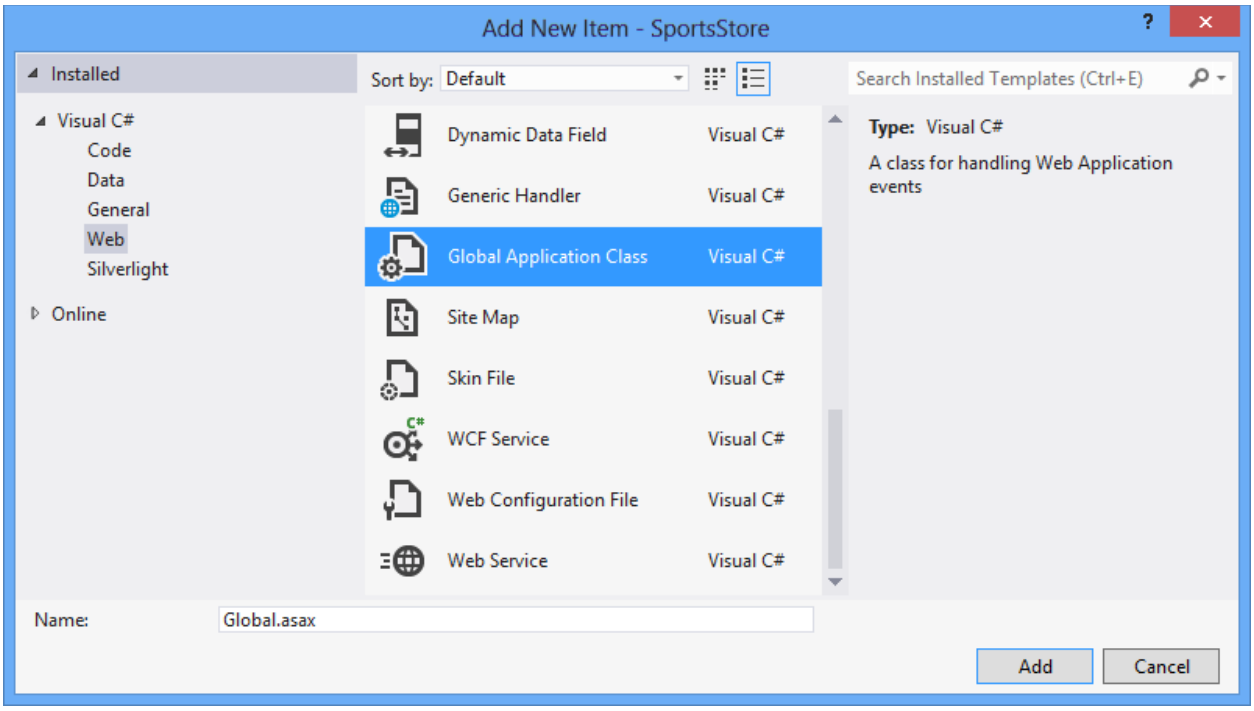


Microsoft include o serie de opțiuni de șabloane care urmăresc să inițieze procesul de dezvoltare prin adăugarea de fișiere și funcții care sunt utilizate pe scară largă în diferite tipuri de proiect.

Noi am pornit cu opțiunea de proiect empty in acest laborator, deoarece dorim să urmarim cum funcționează fiecare caracteristică importantă și cum putem adăuga sau instala aceste caracteristici în orice aplicație Web Forms.

Adăugarea clasei globale de aplicație (Global Application Class). Atunci când o aplicație Web Forms este pornită, sistemul runtime caută o clasă de aplicație global, care este utilizată pentru a răspunde la evenimentele din ciclul de viață al aplicației și care este frecvent utilizată pentru a efectua activități de configurare. Clasa de aplicație Global are un sufix .asax, iar convenția este de a avea o clasă numită Global.asax. Nu este necesar să avețm o clasă Global pentru ca aplicația Web să funcționeze, dar majoritatea aplicațiilor necesită o configurație inițială și unele dintre funcțiile pe care dorim să le utilizăm în aplicația SportsStore sunt gestionate cel mai ușor în acest fel.

Pentru a adăuga o clasă globală de aplicații, facem clic dreapta pe elementul SportsStore din Exploratorul de soluții și selectam Add => New Item din meniul pop-up. Localizam elementul Global Aplication Class (așa cum se arată în figura), ne asiguram că numele este Global.asax și apăsam butonul Add.



Visual Studio va crea fișierul Global.asax și o clasă numita Global.asax.cs. Când facem dublu clic pe fișierul Global.asax din Exploratorul de soluții, vedem de fapt fișierul Global.asax.cs care se deschide pentru editare. Putem vedea conținutul inițial al fișierului Global.asax.cs pe care Visual Studio îl creează în Exemplul 1.

Exemplul *1.* The initial contents of the Global.asax.cs file

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;
namespace SportsStore {
  public class Global : System.Web.HttpApplication {
        protected void Application_Start(object sender, EventArgs e) {}
        protected void Session_Start(object sender, EventArgs e) {}
        protected void Application_BeginRequest(object sender, EventArgs e) {}
        protected void Application_AuthenticateRequest(object sender, EventArgs e) {}
        protected void Application_Error(object sender, EventArgs e) {}
        protected void Session_End(object sender, EventArgs e) {}
        protected void Application_End(object sender, EventArgs e) {}
  }
}
```

Fiecare metodă din acest fisier ne permite să specificam ce dorim sa se execute în acest punct cheie din ciclul de viață al aplicației adica, atunci când aplicația este pornită, când aplicația este oprită, când apare o eroare și așa mai departe.

Pentru a vedea conținutul fișierului Global.asax, facem clic dreapta pe elementul Global.asax din Exploratorul de soluții și selectam View Markup din meniul pop-up. Fișierul care este deschis conține o singură linie, pe care am arătat-o în Exemplul 2.

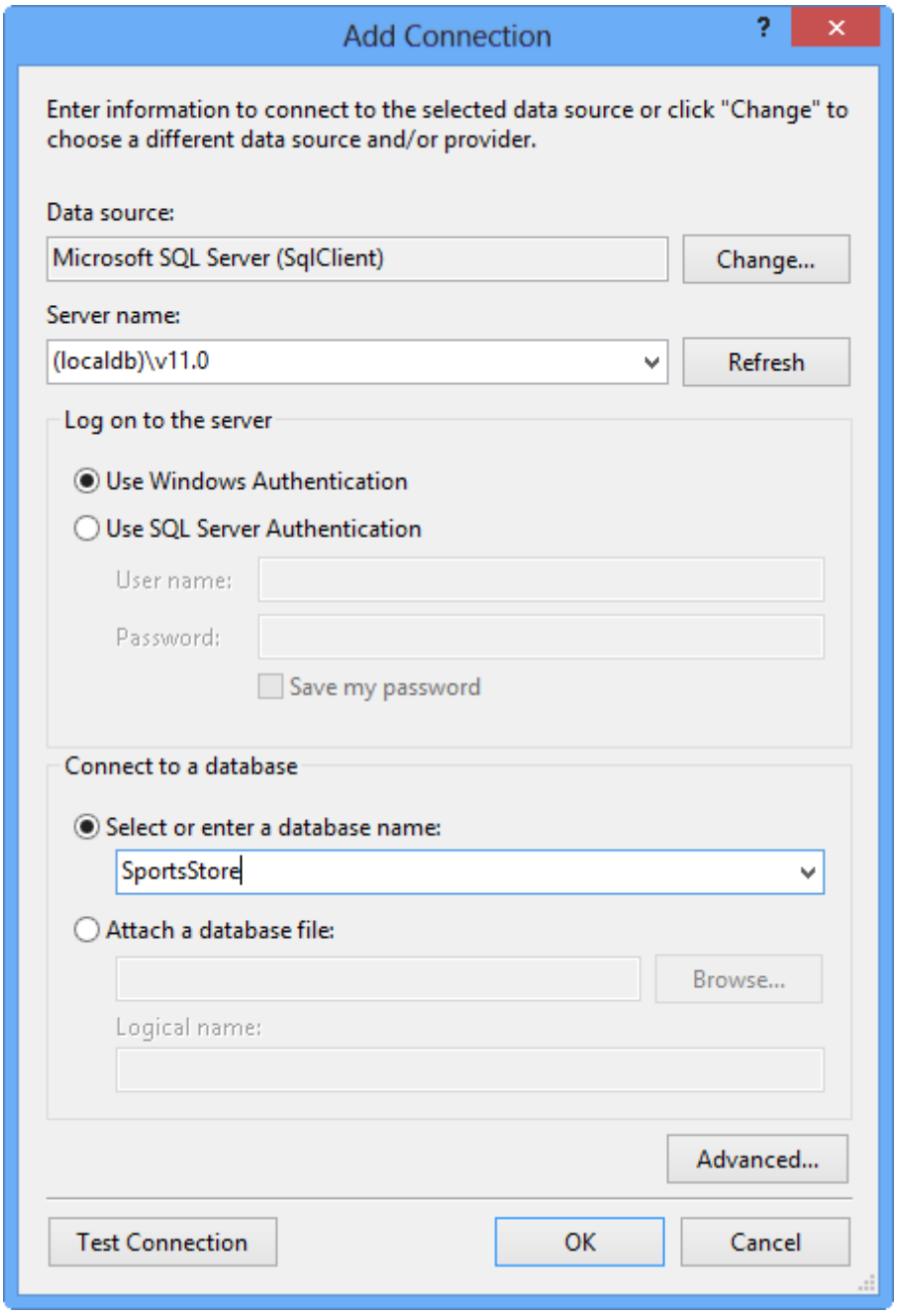
Exemplul *2.* The contents of the Global.asax file

```
<%@ Application Codebehind="Global.asax.cs" Inherits="SportsStore.Global" Language="C#"
%>
```

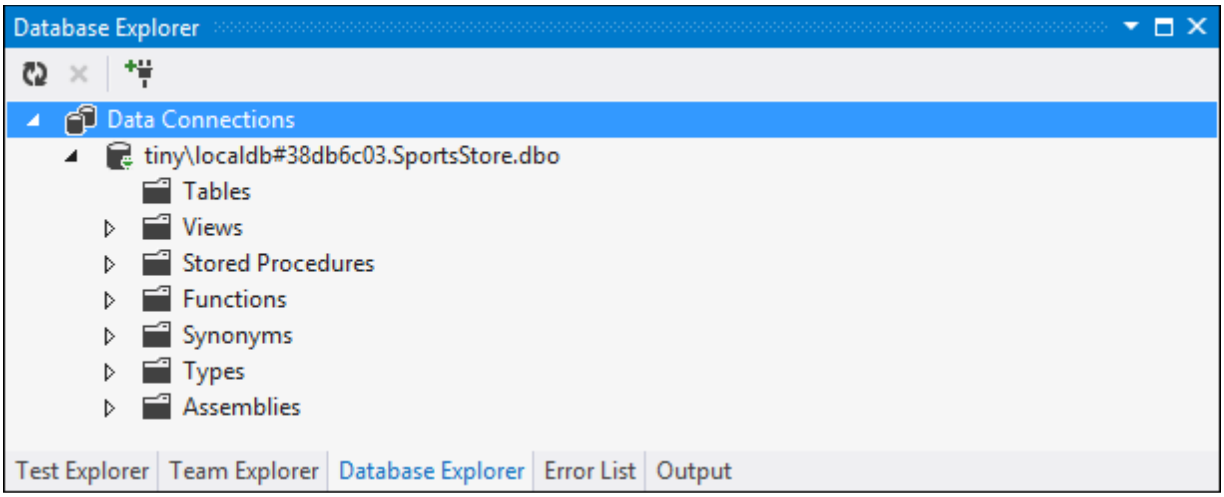
Deocamdată nu vom adăuga cod personalizat în fișierul Global.asax.cs .

Crearea bazei de date. Incepand cu Visual Studio 2012 are o caracteristica LocalDB, care este o implementare fără administrare a principalelor caracteristici SQL Server special concepute pentru dezvoltatori. Folosind această caracteristică, putem sari peste procesul de configurare a unui server de baze de date în faza de constructie a unui proiect.

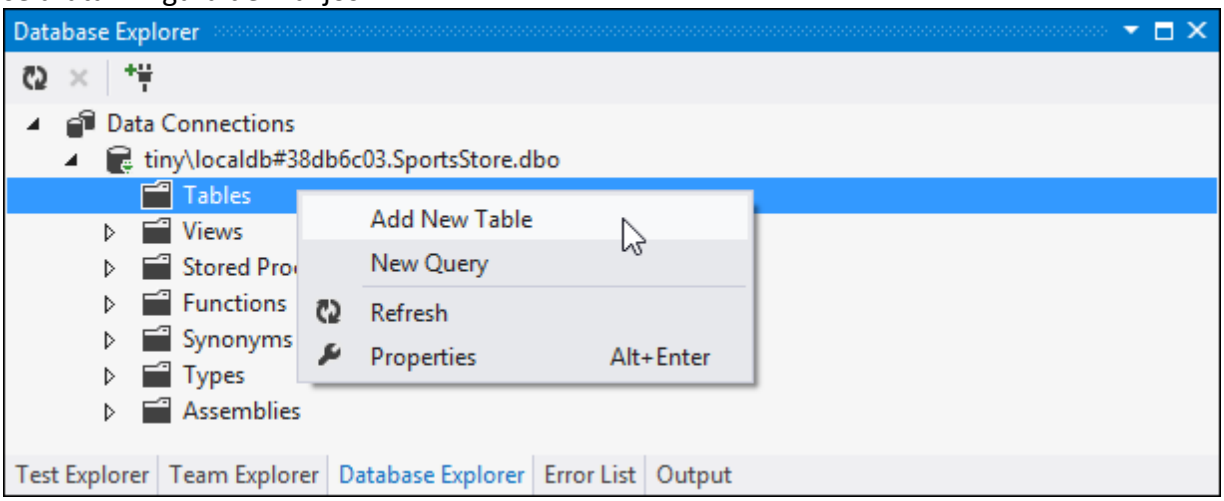
Primul pas este crearea conexiunii la baza de date în Visual Studio. Deschideți fereastra Server Explorer din meniul View și faceți clic pe butonul Connect to Database. Vom vedea caseta de dialog Add Connection. Setam numele serverului la (localdb)\MSSQLLocalDB - acesta este un nume special care indică faptul că dorim să utilizam funcția LocalDB. Ne asiguram că opțiunea Use Windows Authentication este bifată și setam numele bazei de date la SportsStore, așa cum se arată în figura de mai jos.



Facem clic pe butonul OK și ni se va solicita să cream o noua bază de date. Facem clic pe butonul Yes și o nouă intrare va apărea în fereastra Database Explorer asa cum putem vedea in figura de mai jos.



Definirea schemei bazei de date. Avem nevoie de o singură tabelă în baza noastră de date, pe care o vom folosi pentru a stoca detaliile produselor noastre. Folosind fereastra Database Explorer, extindem baza de date pe care tocmai am creat-o astfel încât să putem vedea elementul Tables și facem clic dreapta pe acesta, Selectam Add New Table din meniu, așa cum se arată în figura de mai jos.



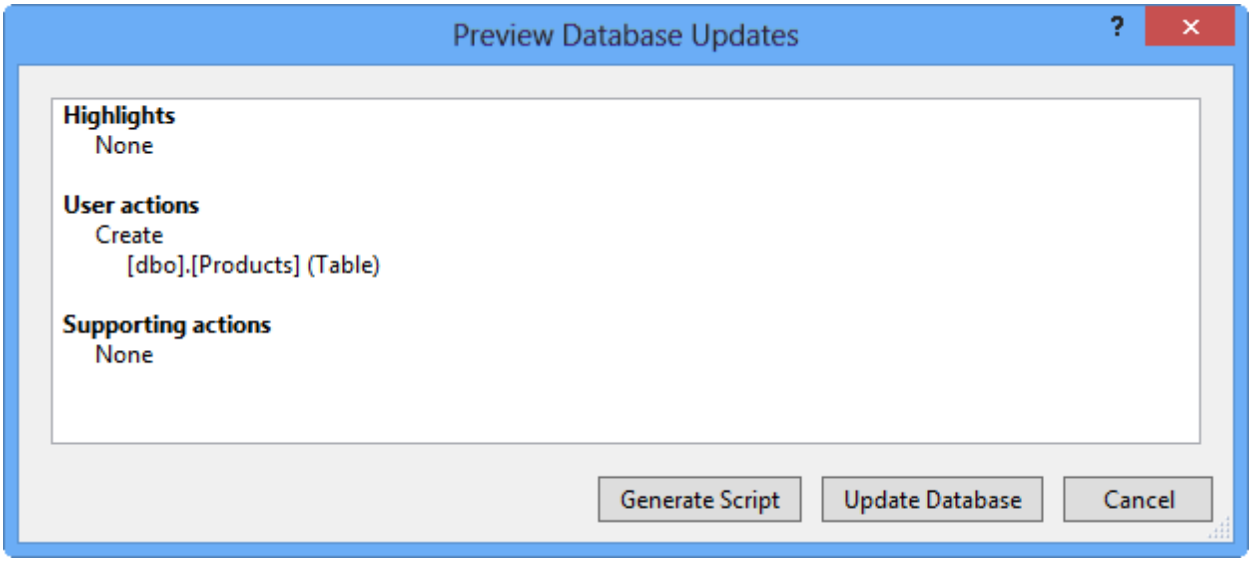
Va fi afișat un designer pentru crearea unui nou tabel. Puteți crea noi tabele in baza de date utilizând partea vizuală a proiectantului, dar vom folosi fereastra T-SQL, deoarece este un mod mai concis și mai precis de a descrie specificația tabelului de care avem nevoie. Introducem instrucțiunea SQL afișată în ***Exemplul 3*** și facem clic pe butonul Update în colțul din stânga sus al ferestrei de proiectare a tabelului.

Exemplul 3. The SQL statement to create the table in the SportsStore database

```
CREATE TABLE Products
(
[ProductID] INT NOT NULL PRIMARY KEY IDENTITY,
[Name] NVARCHAR(100) NOT NULL,
```

[Description] NVARCHAR(500) NOT NULL,
[Category] NVARCHAR(50) NOT NULL,
[Price] DECIMAL(16, 2) NOT NULL
)

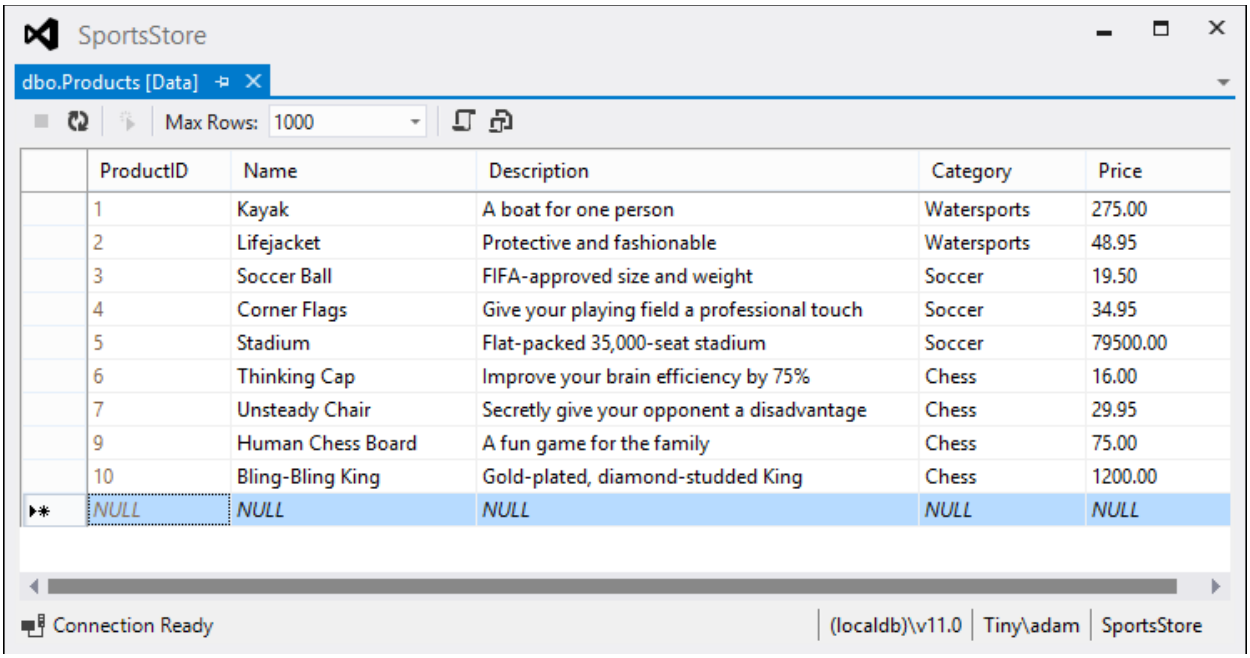
Când faceți clic pe butonul Update, vi se va afișa un rezumat al efectului declarației, așa cum se arată în figura de mai jos.



Facem clic pe butonul Update Database pentru a executa comanda SQL și pentru a crea tabelul Products din baza de date.

Adăugarea de date la baza de date. Vom adăuga manual cateva date de test în baza de date.

În fereastra Database Explorer, extindem elementul Table din baza de date SportsStore, facem clic dreapta pe tabelul Products și selectam Show Table Data. Introducem datele prezentate în tabelul de mai jos.



Crearea modelului de date și a depozitului. Avem nevoie de o modalitate de a opera pe baza de date și conținutul acesteia din aplicația noastră ASP.NET Framework. Pentru a face acest lucru, vom folosi Entity Framework, care este cadrul obiect-relațional Microsoft. Ultimele versiuni ale Entity Framework includ o caracteristică numită cod-first. Ideea este că putem defini clasele din modelul nostru și apoi generam o bază de date din aceste clase.

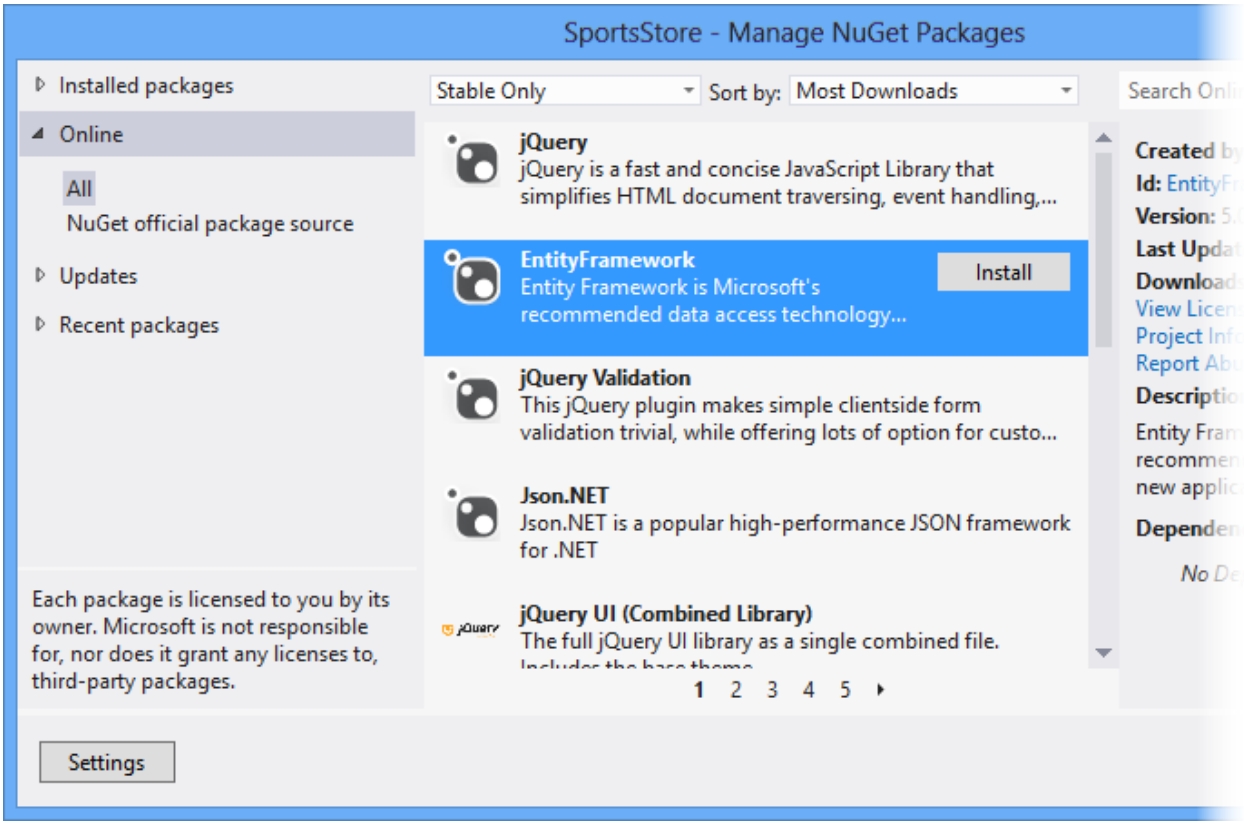
Crearea clasei model de date. Trebuie să creăm o clasă care să reprezinte rândurile din baza de date SportsStore. Fiecare dintre rândurile bazei de date constă într-o descriere a unui produs din magazinul nostru online, așa că am creat un nou fișier de clasă numit Product.cs în folderul proiectului Models. Puteți vedea conținutul acestui fișier în Exemplul 4.

Exemplul *4.* The contents of the /Models/Product.cs file

```
namespace SportsStore.Models {
        public class Product {
                public int ProductID { get; set; }
                public string Name { get; set; }
                public string Description { get; set; }
                public decimal Price { get; set; }
                public string Category { get; set; }
        }
}
```

Acest fișier definește o clasă Product simplă, cu proprietăți implementate automat, care corespund coloanelor pe care le-am creat în baza de date mai devreme.

Adăugarea Entity Framework. Cel mai simplu mod de a adăuga Entity Framework la proiectul nostru SportsStore este cu NuGet. Selectați Gestionați pachetele NuGet din meniul Proiect și căutați entitatea din categoria Online, așa cum se arată în Figura de mai jos.



Selectați pachetul EntityFramework.

Crearea Entity Framework Context. Trebuie să creăm o clasă care să asocieze modelul nostru de date Products cu baza de date pe care am creat-o. Adăugam un nou fișier clasă în folderul Models/Repository, denumit EFDbContext.cs, al cărui conținut îl puteți vedea în Exemplul 5.

Exemplul *5.* The contents of the EFDbContext.cs file

```
using System.Data.Entity;
        namespace SportsStore.Models.Repository {
                public class EFDbContext : DbContext {
                public DbSet<Product> Products { get; set; }
        }
}
```

Pentru a asocia clasa Product cu baza noastră de date, trebuie să creăm o clasă care este derivată din System.Data.Entity.DbContext și care are o proprietate pentru fiecare tabel din baza de date cu care vrem să lucrăm. Numele proprietății specifică tabelul cu tipul DbSet, iar tipul parametrului specifică modelul pe care ar trebui să îl folosească Entity Framework pentru a reprezenta rândurile din tabelul respectiv. În cazul nostru, numele proprietății este Products,

iar tipul parametrului este Product, care indică lui Entity Framework că dorim ca tipul modelului Product să fie utilizat pentru a reprezenta rândurile din tabelul Products.

De asemenea, trebuie să spunem lui Entity Framework cum să se conecteze la baza de date, lucru pe care îl facem prin includerea unui string de conexiune la baza de date in fișierul Web.config. Fișierul Web.config conține informațiile de configurare pentru o aplicație ASP.NET Framework. În Exemplul 6, puteți vedea conținutul fișierului Web.config, împreună cu completările pe care le-am făcut pentru a defini conexiunea la baza de date.

Exemplul *6.* Adding a connection string to the Web.config file

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>

  <configSections>
    <!-- For more information on Entity Framework configuration, visit
http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
  </configSections>

  <location path="admin">
    <system.web>
      <authorization>
        <deny users="?"/>
      </authorization>
    </system.web>
  </location>

  <connectionStrings>
    <add name="EFDbContext" connectionString="Data Source=(localdb)\MSSQLLocalDB; Initial
Catalog=SportsStore;Integrated Security=True" providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>

<authentication mode="Forms">
  <forms loginUrl="~/Pages/Login.aspx">
    <credentials passwordFormat="Clear">
```

```
    <user name="admin" password="secret" />
   </credentials>
  </forms>
</authentication>

   <compilation debug="true" targetFramework="4.5" />
   <httpRuntime targetFramework="4.5" />
   <pages>
    <controls>
     <add tagPrefix="SS" tagName="CatLinks" src="~/Controls/CategoryList.ascx" />
     <add tagPrefix="SS" tagName="CartSummary" src="~/Controls/CartSummary.ascx" />
     <add tagPrefix="SX" namespace="SportsStore.Controls" assembly="SportsStore"/>
    </controls>
   </pages>
   <globalization culture="en-US" uiCulture="en-US" />
  </system.web>
  <entityFramework>
   <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
    <parameters>
     <parameter value="v11.0" />
    </parameters>
   </defaultConnectionFactory>
  </entityFramework>
  <runtime>
   <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
     <assemblyIdentity name="WebGrease" publicKeyToken="31bf3856ad364e35"
culture="neutral" />
     <bindingRedirect oldVersion="0.0.0.0-1.3.0.0" newVersion="1.3.0.0" />
    </dependentAssembly>
   </assemblyBinding>
  </runtime>
</configuration>
```

Atributul name corespunde numelui clasei, care permite lui Entity Framework să descopere automat informațiile de conectare la baza de date. Pentru a descoperi valorile de care aveți nevoie pentru atributele connectionString și providerName pentru un proiect, faceți clic dreapta pe o conexiune la baza de date în fereastra Visual Studio Database Explorer și selectați Properties din fereastra pop-up, care conține informațiile de care aveți nevoie.

Crearea Product Repository. Ultima completare pe care trebuie să o facem este să adăugăm o clasă repository, care operează pe clasa EFDbContext pe care am creat-o mai devreme și care

funcționează ca o punte de legătură între logica de afaceri a aplicației noastre și baza de date. Am creat un nou fișier de clasă numit Repository.cs în folderul Models/Repository și puteți vedea conținutul noului fișier în Exemplul 7.

Exemplul *7.* The contents of the /Models/Repository/Repository.cs file

```
using System.Collections.Generic;
namespace SportsStore.Models.Repository {
        public class Repository {
                private EFDbContext context = new EFDbContext();
                public IEnumerable<Product> Products {
                        get { return context.Products; }
                }
        }
}
```

Clasa Repository definește o proprietate numită Products, care returnează rezultatele citirii proprietății cu același nume din clasa EFDbContext. Am ajuns la punctul în care putem prelua toate rândurile din baza de date și să le reprezentăm pe fiecare prin obiecte Product.

Crearea listei de produse. Acum că avem modelul de date, baza de date și depozitul (**Repository)** , putem începe să construim funcționalitatea orientată către utilizator. Am adăugat un nou formular Web la folderul Pagini numit Listing.aspx, al cărui conținut îl puteți vedea în Exemplul 8.

Exemplul *8.* The contents of the Listing.aspx Web Form file

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing.aspx.cs"
Inherits="SportsStore.Pages.Listing" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
        <title>SportsStore</title>
</head>
<body>
        <form id="form1" runat="server">
                <div>
                <%foreach (SportsStore.Models.Product prod in GetProducts()) {
                        Response.Write("<div class='item'>");
                        Response.Write(string.Format("<h3>{0}</h3>", prod.Name));
                        Response.Write(prod.Description);
                        Response.Write(string.Format("<h4>{0:c}</h4>", prod.Price));
                        Response.Write("</div>");
                }%>
                </div>
        </form>
```

```
</body>
</html>
```

Acest formular web conține cod care obține un set de obiecte product prin apelarea metodei GetProducts din clasa cod-behind și generarea unor elemente HTML de bază pentru fiecare dintre ele.

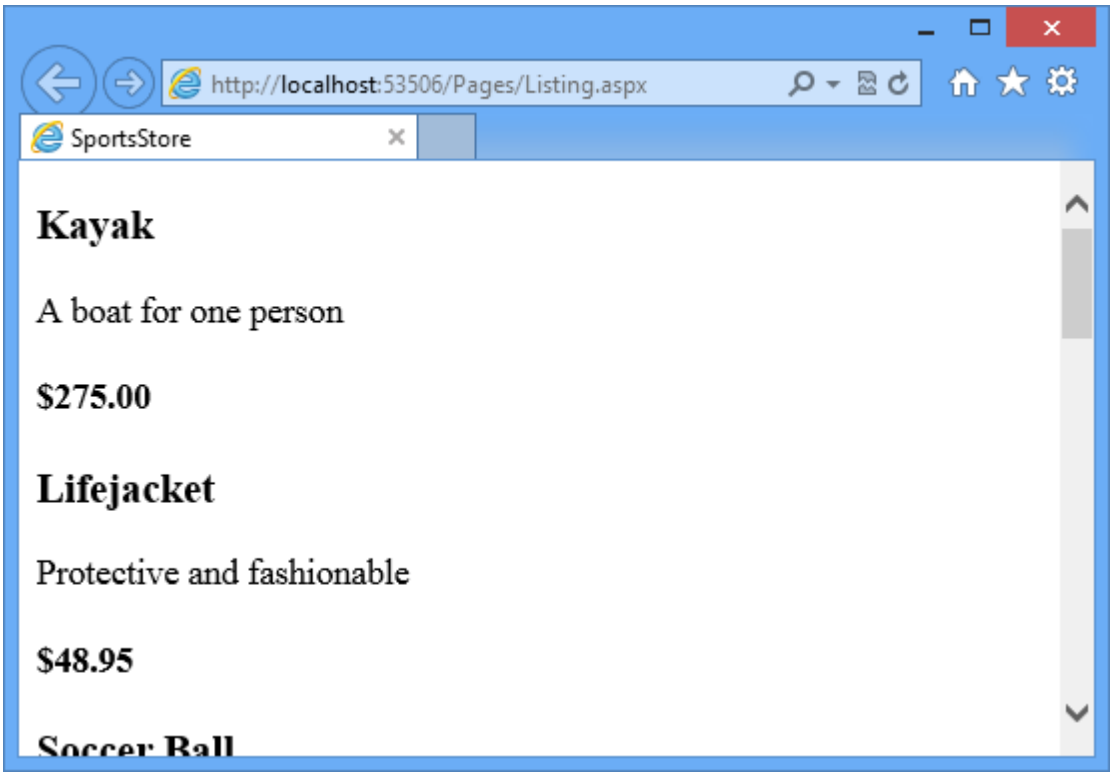
Puteți vedea metoda GetProducts în Exemplul 9, care arată conținutul fișierului Listing.aspx.cs cu codul din spatele paginii Listing.aspx pe care Visual Studio l-a creat pentru Formularul Web, cu unele completări pentru a utiliza clasa Repository in metoda GetProducts.

Exemplul 9. The contents of the Listing.aspx.cs code-behind file

```
using System;
using System.Collections.Generic;
using SportsStore.Models;
using SportsStore.Models.Repository;
namespace SportsStore.Pages {
        public partial class Listing : System.Web.UI.Page {
                private Repository repo = new Repository();

                protected void Page_Load(object sender, EventArgs e) {
                }
                protected IEnumerable<Product> GetProducts() {
                        return repo.Products;
                }
        }
}
```

Tot ce trebuie să facem pentru a aduce conținutul bazei de date în codul din spatele clasei este să creăm o nouă instanță a clasei Repository și să citim proprietatea Products. Pentru a testa noua noastră funcționalitate, selectați elementul /Pages/Listing.aspx din Solution Explorer și selectați Set As Start Page din meniul pop-up. Selectați Start Debugging din meniul Debug și Visual Studio va porni aplicația, va crea o nouă instanță a browserului selectat și navighează la adresa URL care afișează pagina. Puteți vedea rezultatele din figura de mai jos.



Puteți vedea cât de ușor a fost crearea unei baze de date, asocierea acesteia cu clasa noastră de model de date și afișarea datelor din baza de date către utilizator cu un formular Web. S-ar putea să nu arate foarte frumos, dar cu foarte puțin efort am reușit să obținem structura de bază a unei aplicații- iar această viteză și eleganță ne oferă mai mult timp pentru a ne concentra pe funcționalitatea aplicației noastre. Desigur, citim conținutul bazei de date doar în acest moment, dar vom adăuga suport pentru alte tipuri de operațiuni de date în timp ce construim aplicația SportsStore.

Desigur, pentru a ajunge la acest punct, am sărit peste o mulțime de detalii despre modul în care funcționează Entity Framework și numărul imens de opțiuni de configurare diferite disponibile.

Adăugarea paginării. Puteți vedea din figura de mai sus că toate produsele din baza de date sunt afișate într-o listă uriașă intr-o singură pagină. În continuare, vom adăuga suport pentru paginație, astfel încât să afișăm câteva produse simultan și să permitem utilizatorului să parcurgă catalogul general. Trebuie să facem acest lucru în două etape - mai întâi, trebuie să adăugăm suport pentru afișarea unui subset de produse și apoi trebuie să adăugăm legături pe care utilizatorul le poate folosi pentru a naviga de la un set de produse la altul.

Afisarea unei pagini de produse. Putem afișa un număr fix de produse pe pagină, aplicând o selectie LINQ la colecția de obiecte de produs pe care le obținem din baza de date - tot ce trebuie să știm este câte produse de afișat pe pagină și ce pagină vrea să vadă utilizatorul.

Puteți vedea cum am făcut acest lucru în Exemplul 10, care arată modificările pe care le-am făcut la clasa code-behind din fișierul /Pages/Listing.aspx.cs.

Exemplul 10. Adding support for displaying pages of products

```
using System;
using System.Collections.Generic;
using SportsStore.Models;
using SportsStore.Models.Repository;
using System.Linq;
namespace SportsStore.Pages {
        public partial class Listing : System.Web.UI.Page {
                private Repository repo = new Repository();
                private int pageSize = 4;
                protected void Page_Load(object sender, EventArgs e) {
                }
                protected IEnumerable<Product> GetProducts() {
                        return repo.Products
                                .OrderBy(p => p.ProductID)
                                .Skip((CurrentPage - 1) * pageSize)
                                .Take(pageSize);
                }
                protected int CurrentPage {
                        get {
                           int page;
                           return int.TryParse(Request.QueryString["page"], out page) ? page: 1;
                        }
                }
        }
}
```

Am specificat o dimensiune de pagină de patru produse, pe care o realizăm prin câmpul PageSize. Pentru a afla la ce pagină ne aflăm, am creat proprietatea CurrentPage, care folosește colecția Request.QueryString definită in clasa de bază pentru a vedea dacă există o valoare a paginii ca parte a adresei URL solicitate.

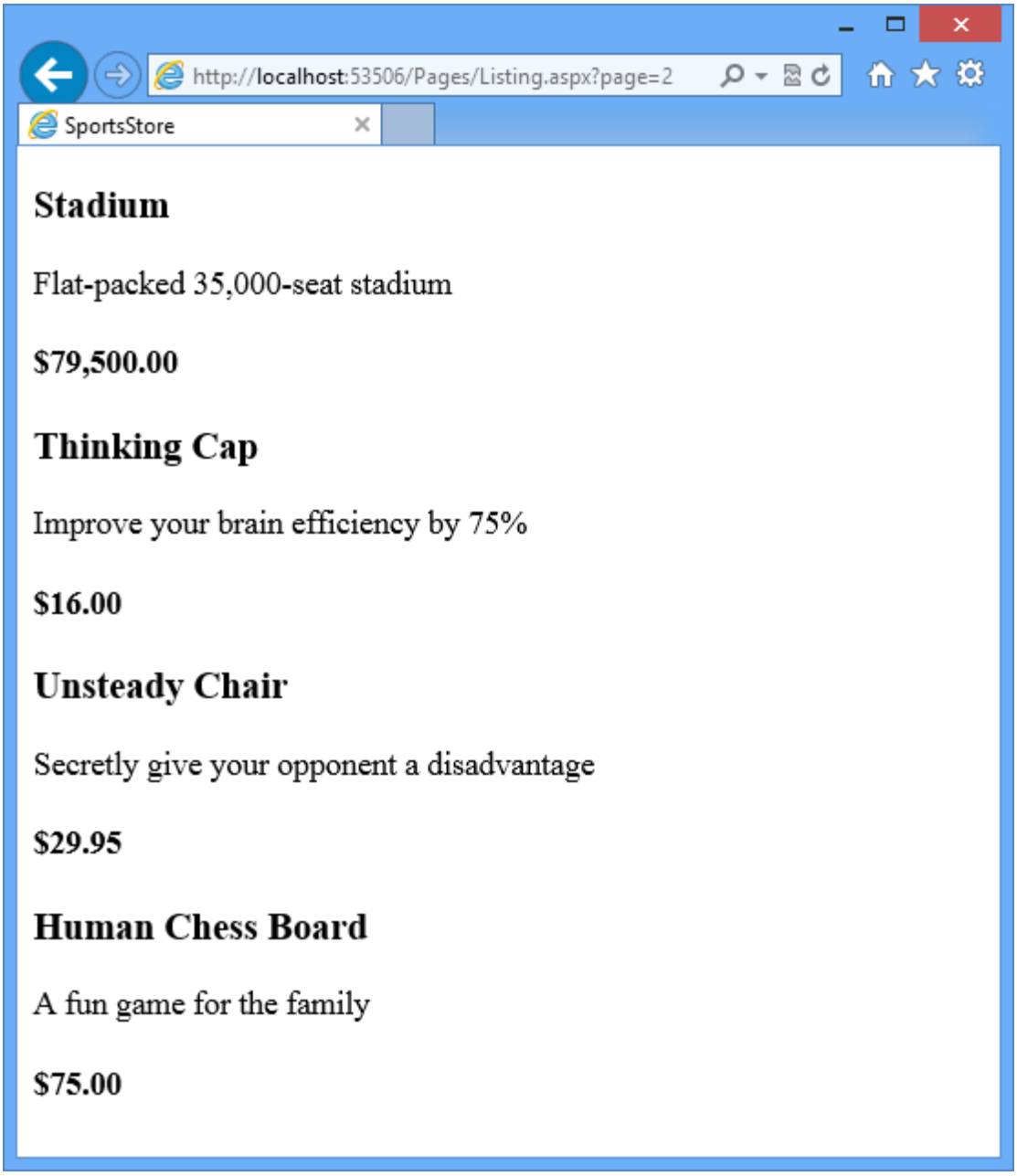
Proprietatea Request oferă acces la detalii despre cererea curentă. Deci, de exemplu, dacă formularul Web este procesat pentru a furniza o adresă URL, cum ar fi următoarea:

http: // localhost: 53506 / Pages / Listing.aspx? page = 2, apoi colecția Request.QueryString va avea o cheie de pagină cu o valoare de 2.

Valorile sunt returnate din colecția Request.QueryString ca șiruri, așa că folosim metoda int.TryParse pentru a încerca să convertim șirul într-o valoare numerică. În mod implicit, avem o valoare 1, care indică prima pagină a produselor dacă nu există o pagină specificată în șirul de

interogare sau dacă nu putem analiza valoarea. Valorile CurrentPage și PageSize ne permit să selectăm obiectele Product de care avem nevoie din depozit.

Folosim metoda LINQ OrderBy pentru a ordona obiectele Product, metoda Skip pentru a ignora obiectele Product care apar înainte de pagina dorită și metoda Take pentru a selecta obiectele Product pe care le afișăm. În figura de mai jos, puteți vedea efectul navigării către a doua pagină a produselor.

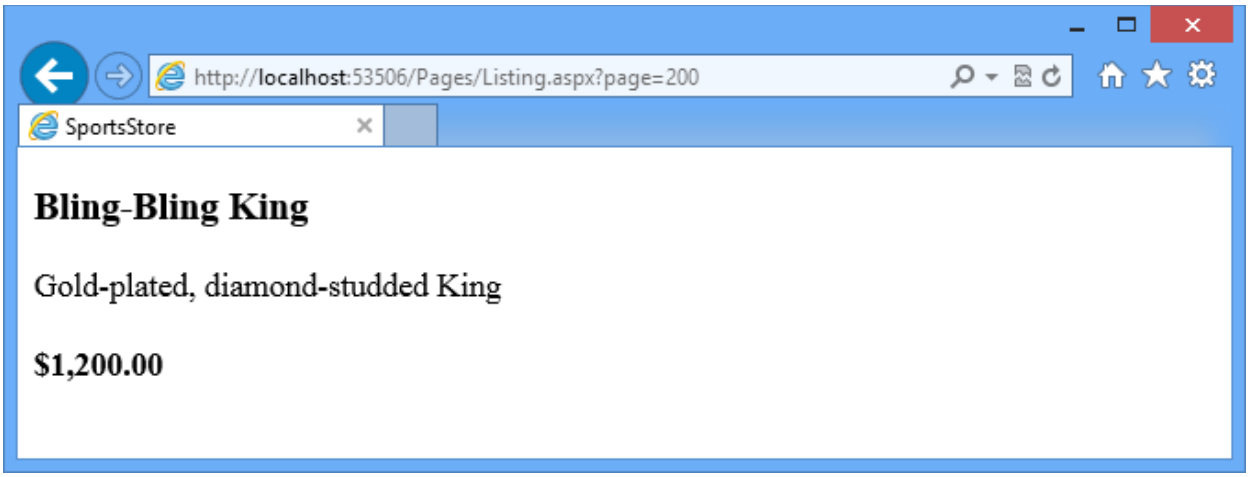


În Exemplul 11, puteți vedea cum am actualizat metoda GetProducts din fișierul /Pages/Listing.aspx.cs pentru a afisa pagina finala de cate ori depasim numarul de pagini existent.

Exemplul 11. Updating the GetProducts method to avoid empty pages

```
using System;
using System.Collections.Generic;
using SportsStore.Models;
using SportsStore.Models.Repository;
using System.Linq;
namespace SportsStore.Pages {
        public partial class Listing : System.Web.UI.Page {
                private int pageSize = 4;
                private Repository repo = new Repository();
                protected void Page_Load(object sender, EventArgs e) {
                }
                protected IEnumerable<Product> GetProducts() {
                        return repo.Products
                                .OrderBy(p => p.ProductID)
                                .Skip((CurrentPage - 1) * pageSize)
                                .Take(pageSize);
                }
                protected int CurrentPage {
                        get {
                          int page;
                          page = int.TryParse(Request.QueryString["page"], out page) ? page: 1;
                                return page > MaxPage ? MaxPage : page;
                        }
                }
                protected int MaxPage {
                        get {
                           return (int)Math.Ceiling((decimal)repo.Products.Count()/ pageSize);
                        }
                }
        }
}
```

Proprietatea MaxPage returnează cea mai mare valoare a paginii pentru care putem afișa produse. Utilizăm această valoare în getter pentru proprietatea CurrentPage, iar rezultatul este că o solicitare pentru pagina 200, de exemplu, este echivalentă cu solicitarea ultimei pagini valabile (care este pagina 3 pentru exemplul nostru, deoarece există nouă elemente în baza de date și afișăm patru articole pe pagină). În figura de mai jos, puteți vedea efectul solicitării paginii 200.



Crearea clasei de configurare a rutelor RouteConfig.cs. Configurația de rutare trebuie să fie facută atunci când începe aplicația ASP.NET Framework, astfel încât adresele URL pe care le acceptăm să fie definite înainte de primirea primei solicitări a clientului. Convenția pentru configurarea de pornire este de a crea o clasă în folderul App_Start care conține o metodă de configurare și apoi să apelam această metodă din clasa de aplicație globală Global.asax pe care am creat-o deja. Convenția de denumire pentru fișierele de clasă din folderul App_Start este <feature> Config.cs, așa că am creat un nou fișier de clasă numit RouteConfig.cs, al cărui conținut îl puteți vedea în Exemplul 12.

Exemplul 12. The contents of the /App_Start/RouteConfig.cs file

```
using System.Web.Routing;
namespace SportsStore {
        public class RouteConfig {
                public static void RegisterRoutes(RouteCollection routes) {
                        routes.MapPageRoute(null, "", "~/Pages/Listing.aspx");
                        routes.MapPageRoute(null, "list", "~/Pages/Listing.aspx");
                        routes.MapPageRoute("admin_products", "admin/products",
                        "~/Pages/Admin/Products.aspx");

                }
        }
}
```

Am folosit clasa RouteConfig pentru a defini o nouă schemă URL pentru aplicația SportsStore. Parametrul rutelor care este pasat metodei RegisterRoutes este un obiect RouteCollection. Folosim metoda MapPageRoute pe care o definește pentru a crea rute. O rută spune ASP.NET Framework cum să proceseze o adresă URL care nu corespunde unui fișier .aspx Web Forms pe disc.

Există o mulțime de moduri diferite de a configura configurația de rutare pentru o aplicație Framework ASP.NET, dar multe dintre ele necesită o explicație detaliată la care nu dorim să ajungem in acest laborator.

Actualizarea clasei globale. Trebuie să apelăm la metoda RouteConfig.RegisterRoutes la începerea aplicației, care necesită utilizarea clasei de aplicații globale Global.asax pe care am creat-o deja. În Exemplul 13, puteți vedea cum am actualizat Global.asax fișier .cs pentru a apela la metoda de configurare a rutelor.

Listing 13. Using Global.asax.cs to perform start-up configuration

```
using System;
using System.Web.Routing;
namespace SportsStore {
        public class Global : System.Web.HttpApplication {
                protected void Application_Start(object sender, EventArgs e) {
                        RouteConfig.RegisterRoutes(RouteTable.Routes);
                }
        }
}
```

Clasa System.Web.RouteTable definește o proprietate static Routes care ne oferă obiectul RouteCollection de care avem nevoie pentru a realiza configurația, lucru pe care îl facem în metoda Application_Start din clasa Global.

Gestionarea catalogului de produse. Este nevoie sa adăugam asistență pentru administrator pentru gestionarea conținutului catalogului de produse. Convenția pentru gestionarea unei colecții de articole este de a afișa o listă cu ceea ce există și de a prezenta utilizatorilor o serie de câmpuri de introducere pentru a le permite să editeze sau să insereze un element nou. Sub această funcționalitate, trebuie să oferim capacitatea de a crea, citi, actualiza și șterge articole din depozitul de date. Impreuna, aceste acțiuni sunt cunoscute sub numele de CRUD, iar nevoia de operații CRUD într-o aplicație este atât de comună încât există funcții extinse în ASP.NET Framework pentru a le face mai ușor de implementat.

In acest laborator ne multumim sa construim un nou formular Web cu numele Products.aspx, in care vom plasa controlul GridView implicit construit de system pentru tabela Products. Prin utilizarea acestui mechanism implicit, renunțăm la o parte din controlul aplicatiei și la înțelegerea modului în care funcționează aplicația noastră. Pentru proiectele reale, tendința este de a se construi manual aceste functionalitati. Dar abordarea noastra este motivate de faptul ca acest laborator are scopul de a va familiariza cu mecanismele aplicatiilor ASP.NET in primul rand.

Exercitii.

1. **Studiati si testati exemplele din textul de mai sus.**
2. **Sa se creeze o aplicatie web ASP.NET care sa aiba o pagina cu doua controale TextBox si un buton si care:**

a) Sa afiseze in primul TextBox ciclul de viata al paginii(mesaje din Page.Init, Page.Load, Page.PreRender, Page.Unload, si Button.Click.)

b) Sa afiseze in al doilea TextBox colectia de controale din pagina utilizand metoda Controls a obiectului Page.

3. **Sa se creeze o aplicatie web ASP.NET care sa actualizeze tabela CATALOG din baza de date TESTDB. Tabela CATALOG are campurile Id,marca,nume,prenume,nota1,nota2.**