Active Server Page - ASP.NET

Concepte de bază

Dinamicitatea paginilor web

- **paginile statice** se mai numesc şi **pagini HTML**, căci nu conţin alt tip de cod decât cel scris în HTML; **conţinutul unei pagini nu variază în timp sau de la un utilizator la altul**; toţi văd la fel, fără informaţii specifice contextului rulării; **conţinutul este fixat şi cunoscut înainte de lansarea cererii** către pagina respectivă.
  - securitate redusă, căci codul este vizibil din browser, cu **View / Source** sau cu **F12 / Sources**

Pentru a reflecta chestiuni legate de un context (de exemplu afişarea orei exacte, sau pentru adaptarea conţinutului paginii la profilul utilizatorului care solicită pagina html) pagina ar trebui compusă **abia după ce s-a înaintat cererea** din partea unui client.

- **paginile dinamice** sunt cele configurate abia **la momentul execuţiei**, în funcţie de context şi de resursele disponibile pe calculatorul de pe care se lansează cererea; de obicei aceste pagini au şi un nivel înalt de interactivitate. Pentru a realiza dinamicitatea, trebuie scris cod sursă care se rulează la momentul lansării unei cereri şi care **prin rulare stabileşte conţinutul ce va fi afişat** în pagină; trebuie avut însă în vedere în ce limbaj este scris acest cod, cine înţelege şi poate prelucra acest cod.

Dinamicitate client-side: codul sursă HTML este mixat cu **instrucţiuni de compunere a paginii** (într-un limbaj script, uzual **JavaScript** sau **VBScript**); pagina e **creată dinamic de către browser**, la momentul solicitării ei; gradul de dinamicitate depinde de capacităţile browser-ului (`Internet Explorer, Netscape Navigator, Opera, Firefox`); script-ul este vizibil în browser (**View / Source**) ceea ce nu e tocmai bine. Se poate folosi acest tip de dinamicitate eventual în **Intranet**, unde ştim ce se află instalat pe maşini şi cunoaştem gradul de securitate ce trebuie asigurat.

Dinamicitate server-side: codul sursă **HTML** este mixat cu **instrucţiuni de compunere** a paginii, scrise **în diverse limbaje de programare şi interpretate de un soft din sever-ul** unde se află pagina solicitată. Pagina este **compusă tot dinamic, dar în server**, la momentul solicitării ei de la distanţă, de către un client. Pagina circulă tot ca stream HTML; mai precis, în modul de prelucrare server, către client în reţea **circulă doar rezultatul prelucrării** paginii, nu şi instrucţiunile de prelucrare. De asemenea, o parte din codul de prelucrare a paginii poate fi **precompilat**, iar la momentul solicitării paginii el este doar rulat, ceea ce poate mări considerabil performanţa accesului. Este mult mai puternic, pe server putându-se instala în siguranţă, toate tipurile de software de creație pagini web.

Tipologia dinamicităţii, Client sau Server, se deduce în funcţie de cine procesează cererea: un **modul specializat din browser** sau un **software din server**.

Web server – este un software care:

- gestionează spaţiul pentru site-uri şi paginile Web ale acestora;
- **ascultă pe un port** (uzual 80), interceptează cererile şi asigură regăsirea şi disponibilitatea paginilor de pe un directorvirtual aflat în gestiunea sa;
- recunoaște protocolul HTTP

Tehnologia CGI – Common Gateway Interface

Avantaje:

- acces la resursele sistemului de operare;
- viteză bună de execuţie, dacă aplicaţia CGI este deja compilată

Dezavantaje:

- depanare dificilă;
- se execută în proces separat, consumând multe resurse.

Tehnologia ASP.NET

Modelul disponibilizat sub **.NET** de către Microsoft pentru aplicaţiile accesibile din Internet este denumit pe scurt **ASP.NET (Active Server Page)** şi are următoarele caracteristici principale:

- se bazează pe formulare Web (Web Forms) şi separă **logica prezentării** de **logica de business**;
- furnizează **controale de server** ce recunosc evenimente la nivel de server, dar sunt în final convertite în controale HTML, pentru a fi recunoscute şi tratate în orice browser;
- asigură accesul la date prin mecanismul **ADO.NET**;
- permite ***caching*** pentru date şi pentru **salvarea stării unui client**, pe **server**, pe calculatorul **client** sau pe **servere SQL specializate**.

Avantaje ASP

- este o tehnologie **orientată obiect** pentru dezvoltare rapidă de aplicaţii; elementele HTTP sunt tratate tot obiectual;
- lucrează **cu pagini compilate**, nu interpretate pas cu pas; **recompilarea se face doar când este nevoie**;
- permite accesul la toate clasele din .NET, clase care pot îndeplini sarcini complexe;
- asigură securitatea prin mecanisme *Forms-based* şi *Passport authentication;*
- tratează fişiere XML, inclusiv pentru reconfigurări, **fără a necesita restartarea server-ului;**
- oferă extensibilitate prin crearea şi integrarea de noi componente sau înlocuirea unora existente cu versiuni noi.

Instalarea componentelor web-server și stabilirea legăturii cu ASP.NET

Directorul fizic `c:\inetpub\wwwroot`;
Director virtual – Virtual Directory, este o legătură către unul sau mai multe directoare aflate fizic în afara arborelui site-ului Web.

- ne introduce în fereastra IIS conţinând lista site-urilor din evidenţa serverului Web (Default Web Site, plus altele). Trebuie ca IIS să fie însă instalat.
- Dacă nu e startat IIS îl putem starta / restarta de aici: selectat Default Web Site / Stop / Start / Restart IIS; nu necesită restartarea calculatorului.

- O altă modalitate de a ajunge în fereastra de startare IIS: **C:\Windows\System32\inetsrv\InetMgr.exe; se pot** stabili drepturi şi interdicţii pe diverse servicii ale IIS.
- Adăugare director virtual:
  Default Web Site / buton dreapta mouse / New Virtual Directory; aplicaţiile create cu Visual Studio beneficiază de **propriul director virtual**, creat odată cu aplicaţia; cele create cu Visual Studio pot rula şi în afara IIS, sub un **server web de dezvoltare** (integrat); ele vor fi aduse la finalizare (deployment) într-un director virtual, fiind înregistrate în metabaza IIS, pentru ca site-ul să fie vizibil și din afara aplicaţiei.

Control Panel / **Administrative Tools** / Computer Management / Services & Apps / IIS / Web Sites / Default Web Site cu click mouse dreapta / Properties se poate schimba portul prin care se comunică cu serverul: TCP **Port = 80** sau **81**.

Dacă pe calculator se dispune de mai multe plăci de reţea pot exista mai multe Local Area Connection (Start / Control Panel / Network and DialUp Connection); ele pot avea setări diferite. Se pot pune mai multe adrese IP chiar pe aceeaşi conexiune. O adresă IP poate avea şi pseudonim:
- în registru DNS, pe serverul de reţea;
- local, în `c:\winnt\system32\drivers\etc\hosts` ca text:

```
38.25.63.10  www.ceva.ro
```

Dificultatea realizării unor pagini web conținând date stocate în BD pe internet constă în faptul că trebuie corelate trei componente:
- **Server Web - IIS Microsoft,** pentru stocarea și regăsirea paginilor;
- **Visual Studio .NET ASP.NET,** ca mediu de programare pentru crearea paginilor;
- **Server de BD - SQL Server,** pentru gestiunea datelor.

Creare site Web

- În fereastra IIS cu buton dreapta mouse pe calculatorul dorit, se alege **New / Web site** şi se indică adresa de IP înregistrată mai sus, plus drepturile de acces.

- La încercarea de conectare de la distanţă:
  - fie se afişează directoarele din site (opţiunea `Directory Browsing` activă);
  - fie se predă controlul unui document implicit (opţiunea `Default Document` activată).

Cele două opţiuni se controlează din consola IIS (tab-urile Default documents și Directory browsing); trebuie creată însă una din paginile: **default.htm, default.asp, default.aspx, index.htm** etc.

Mixarea informaţiilor într-o pagină web

O pagină Web poate conţine:
- cod **html**, recunoscut de orice browser;
- cod într-un **limbaj de programare** recunoscut de către serverul care ţine pagina Web;
- cod **script** într-un limbaj de tip script, recunoscut de majoritatea browserelor (JavaScript, VB Script fiind cele mai răspândite limbaje de tip script).

Modul în care se mixează aceste trei tipuri de cod complică oarecum înţelegerea lucrurilor, dar este foarte important pentru că el dă ordinea în care se afişează diferitele informaţii în pagină. Din punctul nostru de vedere, reţinem două dintre tag-urile care introduc cod scris într-un alt limbaj decât html:

- **secţiunea de precizare a codului**

```
<script language="C#" runat="server">   cod C#   </script>
```

- **secţiunea de folosire a codului**

```
<% // apel cod C# %>
```

De obicei punem codul propriu-zis **în prima parte a paginii**, iar apelul codului (partea de prezentare) în interior, acolo unde dorim să apară efectul rulării, în pagină.

Uzual, se practică şi gruparea întregului cod în fişier separat, tehnică denumită "code behind"; la începutul paginii se indică numele fişierului conţinând codul, iar în interiorul paginii se trece apelul codului, ce va genera ca rezultat prezentarea paginii.

```
<%@ Page Language="c#" Src="fis.cs"%>

<% // apel cod C# %>
```

Câteva exemple complete vor clarifica cele discutate mai sus.

Exemplu `DataOra.aspx` Dinamicitate server-side

Cu un editor de text se introduce secvenţa de mai jos, într-un fişier **`DataOra.aspx`** plasat în **`c:\inetpub\wwwroot`** (tipul **`aspx`** este obligatoriu pentru a identifica programul care îl tratează):

```
<html>
<script runat="server" language="c#"> </script>
<% Response.Output.Write(  System.DateTime.Now.ToString()); %>
</html>
```

şi se apelează din Internet Explorer sub forma `http://localhost/DataOra.aspx;` ea va afişa data şi ora exactă. Observăm că avem doar cod de redare, nu şi cod propriu-zis; blocul script este gol în acest caz şi **se pune doar pentru a preciza limbajul** în care este scris codul de redare, lucru care s-ar fi putut da şi în directiva de pagină. Pagina web de mai sus exemplifică şi tipologia de dinamicitate *server side*, deoarece programul va arăta alt conţinut în funcţie de momentul solicitării paginii; codul din server este cel responsabil în acest caz, cu extragerea şi afişarea orei exacte.

Dacă nu dispunem de serverul de web IIS instalat, putem starta serverul de web de sub Visual Studio rulând o aplicaţie oarecare; cât timp serverul de dezvoltare e activ, modificăm în linia de adresă din browser doar numele fişierului ce conţine pagina (spre ex. înlocuim `default.aspx`, cu **`DataOra.aspx`,** cu **c**ondiţia ca **fişierul să fi fost plasat în acelaşi director cu aplicaţia care a startat serverul**, deoarece serverul vede directorul aplicației ca pe un director virtual **).**

Observaţie.

Save Target as... din browser salvează pagina aşa cum arată ea prelucrată (adică tradusă în html); în consecinţă, fişierul salvat cu **Save Target as**... la rularea paginii, cu extensia **html**, în **wwwroot,** va afişa la un nou apel în browser, mereu **aceeaşi oră** !

Pentru fişierul iniţial, ce conţine codul de prelucrare, extensia fişierului trebuie să fie `aspx`, pentru a anunţa serverul cui s-o paseze pentru prelucrare, deoarece acest tip de fişier conţine nu forma prelucrată a paginii, ci instrucţiunile de prelucrare !

Exemplu `DataOraJS.html` Dinamicitate client-side

Cu un editor de text se introduce secvenţa de mai jos, într-un fişier numit **`DataOraJSP.html`** plasat în **`c:\inetpub\wwwroot`** (tipul **`html`** ne indică faptul că **pagina poate fi prelucrată direct de către browser**):

```
<html>
  <script language="javaScript">
      document.write(  " <b> Dinamicitate client side:   </b>" );
      azi = new Date();
      document.write( azi.getDate(),"/ ",azi.getMonth()+1,
      "/ ",  azi.getYear(), " ",azi.getHours(),":",
      azi.getMinutes(),":", azi.getSeconds()  );
  </script>
</html>
```

Această pagină realizează aproximativ acelaşi lucru ca în exemplul anterior, dar responsabil cu această sarcină este de data aceasta **browser-ul cu care un client navighează pe această pagină**; aşadar browser-ul clientului este cel care interpretează script-ul Java şi îl execută, apoi afişează rezultatul. Ora afişată va fi cea de pe calculatorul clientului, nu cea din server; presupunând că paginile sunt vizualizate de pe aceeași mașină și ambele ceasuri arată bine ora exactă, cele două exemple de mai sus vor afişa ore diferite, dacă server-ul se află pe un alt fus orar decât calculatorul client!

Exemplu `functia.aspx`

În acest exemplu vom avea ambele categorii de cod: în prima parte se prezintă codul unei funcţii, iar în interiorul paginii se pune codul de vizualizare, care apelează această funcţie şi afişează pătratele rezultate prin rularea funcţiei.

```
<script runat="server">
double Patrate(double nr) { return (nr)*(nr); } </script>

<html>
<h2> Patratele primelor 10 numere naturale </h2>
      <table border=""2"">
            <tr>
                  <th> Numarul  </th><th> Patratul </th>
            </tr>
      <%
            for (double i = 1.0; i<=10.0; i++)
            {
                  Response.Output.Write(  "<tr><td>{0}</td><td>" +
                                    "{1:f}</td><tr>", i, Patrate(i));
            }
      %>
      </table>
</html>
```

Pagina se apelează în browser sub forma `http://localhost/functia.aspx`

Reiese că funcţia (în general, codul din blocul script) nu este activată decât în momentul când este **solicitată de un eveniment** sau **de către un alt cod executabil, pus în partea de redare** `<% %>`. În exemplul nostru, apelul `Patrate(i)` invocă partea de cod din blocul script.

Reamintim că la client ajunge pagina deja prelucrată, astfel încât dacă în browser cerem din meniu **View / Source**, vedem pătratele deja calculate, browser-ul doar afişându-le.
Neprecizând `runat="server"` se presupune că se execută implicit în browser.

Într-o pagină se poate pune cod scris doar într-un singur limbaj, în cazul nostru C#.

Sensul codului de redare de mai sus este următorul: obiectul pagină are proprietatea `Response` care este o referinţă la un obiect HTML de tip `Response`, ce conţine răspunsul dat de server la solicitarea unei pagini de către un client. Obiectul `Response` recunoaşte metodele `Response.Write()` şi `Response.Output.Write()` cu care putem adăuga linii în răspunsul returnat clientului; `Response.Output.Write()` permite chiar şi scrierea de **text formatat**, pe care un browser ştie să-l interpreteze.

O altă variantă ar fi să punem funcţia într-un fişer separat, pe care să-l cităm în clauza **`Src="fis.cs"`**, lăsând în pagină **doar partea de apel şi de redare pagină**. În ambele variante codul sursă se compilează la momentul cererii paginii; vom vedea în continuare că o alternativă mai eficientă este să **precompilăm codul**, mărind astfel viteza de răspuns la o cerere.

ASP.NET oferă alternativa obiectuală pentru crearea şi accesul la paginile Web. Documentul html este împachetat într-un obiect de clasă `WebForm1` sau `_Default`, derivată din clasa `Page`. Ceea ce este cuprins între `<script runat="server" language="c#">` şi `</script>` **se include în această clasă**, iar ceea ce se specifică între `<%` şi `%>` **poate fi imaginat ca o parte dintr-o funcţie `Render()`** a clasei **derivată** din clasa Page.

Reiese că între `<script>` şi `</script>` putem include funcţii, pe când între `<%` şi `%>` nu dăm decât elemente specifice redării paginii Web, neputând defini alte funcţii deoarece ne aflăm deja în interiorul unei funcţii (adică în funcţia **`Render()`**).

Un exemplu în care codul mixat într-o pagină web nu este apelat explicit din zona de redare, ci implicit, **pe bază de eveniment** semnalat la nivel de pagină (evenimentul **`Page_Load`**) ar fi următorul:

Exemplu `bdPage.aspx`

```
<%@ import Namespace="System.Data" %>
<%@ import Namespace="System.Data.OleDb" %>

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

private void Page_Load(object sender, System.EventArgs e)
 {
  string strSql = "SELECT codp, denum, pret FROM produse";
  OleDbConnection con =
      new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data" +
                                                 " Source=C:\prod.mdb");
  try {con.Open(); }
  catch(Exception ex) {mesaje.Text ="Eroare open conexiune"+ex.Message;}

  OleDbCommand myCmd    = new OleDbCommand(strSql,con);

  OleDbDataReader dr = myCmd.ExecuteReader();
      while (dr.Read())
       {
        mesaje.Text+="\r\n"+dr["codp"]+ " " + dr["denum"]+ " " + dr["pret"];
        mesaje.Text+="\r\n";
       }
      dr.Close(); con.Close();
 }
</script>

<html>
  <body>
      <form id="Form1" method="post" runat="server">
            <h4> Interogare BD folosind un obiect DataReader      </h4>
            <asp:TextBox id="mesaje" runat="server" TextMode="MultiLine">
            </asp:TextBox>
            <h4> End Interogare BD </h4>
      </form>
  </body>
</html>
```

Se rulează dând în linia de adresă a browser-ului `http://localhost/`**`bdPage.aspx`**; trebuie doar să ne asigurăm că există fişierul **`C:\prod.mdb`** conţinând o bază de date Access, cu tabela produse şi câmpurile: `codp`, `denum` şi `pret`.

Pregătirea paginii se face tot pe server; o problemă care apare aici este cum va călători informaţia extrasă de pe server (spre exemplu, dintr-o bază de date) până la client, pentru vizualizare în browser. Putem defini o variabilă la nivelul paginii, `string strRezultat;` ea este recunoscută şi de funcţia `Render()`, dar îşi pierde conţinutul între două cereri succesive. Putem apela la un control de tip `TextBox`, care va transporta informaţia în proprietatea sa numită `Text`, folosită drept container; acesta este mecanismul pentru care s-a optat în exemplul de mai sus, rolul de container având-l TextBox-ul `mesaje`.

La încărcarea paginii în server, se lansează evenimentul `Page_Load`, care apelează implicit funcţia de tratare cu acelaşi nume, moment în care se accesează baza de date şi se adaugă informaţia, linie cu linie, în textBox;

În rest, după cum se poate observa, lucru cu obiecte .NET este cel uzual, chiar când e vorba de obiecte pentru acces la baze de date.

Web Forms controls desemnează *server controls*

Crearea unei aplicaţii Web folosind Visual Studio

Sub .NET, **File / New / Web Sites** / **ASP.NET Web Site** şi se alege un nume de aplicaţie şi ca localizare se alege **File System**; pentru a se putea lucra rapid, cel puţin pentru faza de realizare şi depanare a aplicaţiei este preferabil această localizare.

Localizarea site-ului pe care se lucrează se poate face alegând una din opţiunile:

- **File System –** dacă se experimentează pe un site creat provizoriu într- un director local sau de pe un alt calculator din reţea expus ca partajabil (shared). Se poate crea şi un director nou, folosind icon-ul *Create New Folder* sau pur şi simplu adăugând numele noului director, în finalul unei căi de acces deja selectată. Rularea se face într-un server de web de testare, numit **ASP.NET Development Server** şi foloseşte un port, ales adhoc, nu pe cel folosit de **IIS**; avem avantajul că putem muta directorul, căci el nu e luat în evidenţa IIS (metabaza IIS) ca director virtual şi nu e accesibil din Internet.

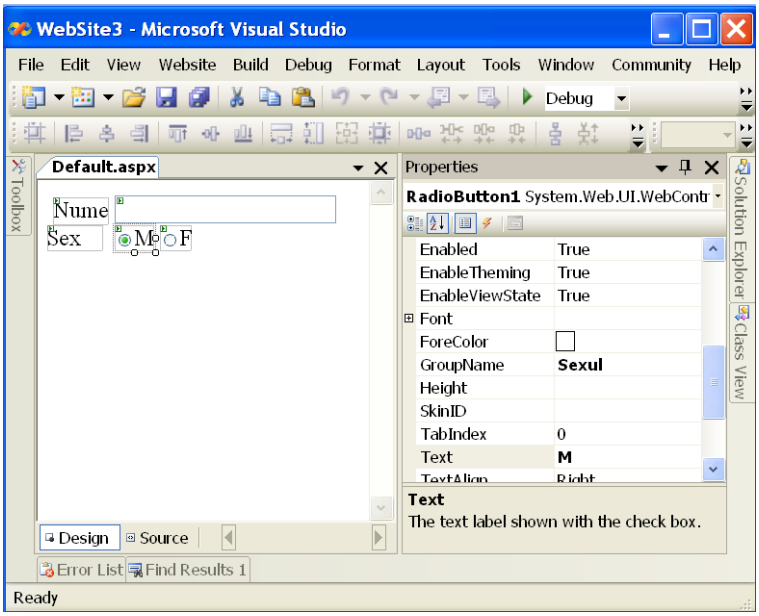
- **Local IIS** – dacă dorim ca site-ul să se afle într-un director virtual recunoscut de server-ul de web local (**IIS**). Cu icon-ul din colţul dreapta-sus *Create New Web Application* se poate crea chiar acum un nou director virtual.

- **FTP Site** - când lucrăm pe un site la distanţă (sub alt IIS, decât cel local) şi-l accesăm prin FTP, furnizând informaţiile de conectare (*FTP site, port, director, user, password*).

- **Remote Web Server** - când lucrăm pe un site la distanţă şi-l accesăm prin protocolul HTTP, furnizând URL (uniform resource locator); în acest caz este nevoie să avem instalate extensiile FrontPage; la conectare vom furniza *user* şi *password*.

Spre deosebire de Windows Forms, aplicaţiile Web au fereastra de vizualizare grafică (Designer) cu mai multe formate de vizualizare, accesibile prin comutatorii din josul paginii:

- **Design View** – vizualizarea grafică a controalelor din pagină, care permite lucru în regim grafic; controlele din ToolBox vor fi selectate din tab-ul Web Forms, nu din Windows Forms.



- **Source View** – vizualizare în format HTML a codului ce integrează formularul Web. Se observă aici descrierea html aferentă controalelor adăugate vizual pe formă.

Rularea paginii de sub Visual Studio declansează activarea unui **ASP Development Server**, folosind câte un port specific pentru comunicare; lansarea paginii din afara mediului Visual Studio se poate face direct din Internet Explorer sau alt browser, indicând adresa paginii (ex. **http://localhost:1966/WebSite3/myPage.aspx**) pe care o folosea şi mediul când rulam sub mediul integrat (extrasă din bara de adresă) şi va funcţiona dacă serverul de dezvoltare este încă activ la acel moment.

Specificarea altor detalii de trasare pagină se poate face folosind foi de stiluri.
Consultând directorul în care a fost creată aplicaţia (`C:\Inetpub\wwwroot\prima`) observăm mai multe fişiere:

- **xxx.aspx** – conţine **codul de vizualizare** a paginii Web (implicit se numeşte **default.aspx**);
- **xxx.aspx.cs** – conţine **codul sursă C#** pentru declaraţiile controalelor şi funcţiilor de tratare a evenimentelor de la nivelul paginii. Un utilizator nu vede acest cod, chiar dacă în browser alege **`View/Source`**, căci codul vizibil în **`View/Source`** este cel deja prelucrat de server.
- **global.asax** – (dacă se adaugă la proiect / click mouse dreapta pe Solution Explorer); conţine informaţiile şi codul de tratare a evenimentelor de la nivel de aplicaţie.
- **web.config** – conţine detaliile de configurare a site-ului web şi care sunt preluate cu eventuale modificări la nivel de aplicaţie.

ASP.NET începând cu versiunea 2.0 foloseşte pentru dezvoltarea aplicaţiei un server web integrat; acesta lucrează pe un port distinct de portul pe care lucrează IIS, prevenind astfel eventualele conflicte ce pot apare. Server-ul web încorporat rulează o pagină web în numele utilizatorului logat în Windows, deci cu eventuale privilegii extinse, faţă de IIS care rulează cu privilegii stricte, din considerente de securitate sporită.

În scopul prevenirii conflictelor de aparteneţă a unei clase la mai multe namespace-uri, va trebui dată calificarea completă a clasei; cum această ar putea fi prea lungă, se poate folosi **`using`** pentru a introduce un *alias* mai scurt:

```
using scurt = NumeDeSpatiuLung;
scurt.Cls ex;
```

Ierarhiile din **namespace-uri nu reflectă neapărat derivări**; la derivare clasele pot aparţine unor namespace-uri diferite.

Namespace-ul nu se confundă cu library; o bibliotecă **dll** are un corespondent fizic (fişier dll ce conţine codul executabil aferent unor clase şi funcţii), pe când namespace-urile sunt mai degrabă grupări logice ale unor clase şi funcţii.

Fazele de lucru cu o pagină Web

1. **Iniţializarea paginii** – este rezultatul unei cereri de la un browser; instanţiază controalele din pagină şi le iniţializează cu valori indicate prin *Properties*; dacă suntem pe **postback** (adică pagina nu e la prima solicitare, ci este repostată) valorile curente ale controalelor sunt extrase din *ViewState* şi folosite la iniţializare. În această fază se declanşează şi evenimentul PageInit, dar e puţin folosit, căci se declanşează **înainte** de instanţierea şi încărcarea controalelor cu valori utile; cum nu prea există prelucrări care să nu folosească controale, codul funcției de tratare PageInit nu prea are mare utilitate.

2. **Iniţializarea pe baza codului indicat de utilizator –** declanşează evenimentul PageLoad şi deci va executa codul scris de programator în funcţia de tratare PageLoad. Denumirea de PageLoad este aleasă pentru a sugera că uzual, pe server se încarcă şi pagini care au mai fost vizualizate în browser şi se întorc în server cu completări. In funcţia de tratare PageLoad, pe ramura !Page.IsPostBack se pot pune valori de iniţializare ale controalelor; pe cealaltă ramură, valorile nu trebuie puse prin cod căci se preiau automat (vezi faza anterioară) din ViewState. Dacă totuși iniţializăm controlul prin cod, atunci măcar trebuie pus EnableViewState pe *false*, altfel nu se justifică munca pierdută prin suprascrierea unor valori.

3. **Validarea prin controale de validare** – este o fază asociată cu controalele de validare introduse de ASP.NET; ea se derulează înaintea oricăror evenimente de utilizator.

4. **Tratarea evenimentelor de utilizator** – este o fază derulată după ce pagina e complet încărcată în server, iniţializată şi validată. În principiu, evenimentele de utilizator ar putea fi împărţite în două categorii:

   - **evenimente cu răspuns imediat** (generate de controalele Auto PostBack pentru evenimente ce trebuie notificate imediat serverului, pentru a fi tratate); intră în această categorie evenimentul de Click pe diverse controale care-l recunosc;
   - **evenimente de modificare** (de exemplu, TextChanged, IndexChanged într-un control de selecţie etc.; ele vor fi tratate nu imediat, ci la următoarea încărcare a paginii în server; dacă dorim tratare imediată, proprietatea AutoPostBack a controlului respectiv se pune pe *true*, ceea ce forţează notificarea imediată a severului despre producerea acestui tip de eveniment.

5. **Legarea datelor** – este o fază care se derulează în două trepte:
   - **insert**, **delete** şi **update** se execută imediat după producerea evenimentelor pe controale, dar **înainte de Page.PreRender()**;
   - **select** se execută după **Page.PreRender()**, alimentând cu date controalele legate la surse de date; această succesiune are dezavantajul că funcţiile de tratare evenimente (vezi etapa 4) nu beneficiază de cele mai recente date, aduse în controale.

   Legarea datelor **se face automat la fiecare postback**; dacă se doreşte scrierea de cod ce foloseşte datele dintr-un control cu data-binding, acest cod poate fi pus numai într-o supraîncărcare a metodei Page.OnPreRenderComplete(), care se execută după legare şi înainte

de redarea paginii ca HTML. Uzual nu e nevoie de preluat date din control, căci datele pot fi preluate direct din baza de date!

6. **Eliberarea resurselor** – se face după ce au avut loc toate transformările paginii în server şi pagina este redată ca HTML şi porneşte spre client. În acest moment se declanşează evenimentul Page.Unload şi nicio modificare asupra paginii nu mai este posibilă. *Garbage – collector*-ul eliberează toate instanţele de obiecte ce nu mai sunt referite şi se declanşează evenimentul Page.Disposed, care încheie ciclul de viaţă al paginii în server.

Pentru aşezarea mai flexibilă a controalelor în pagină se poate folosi un control de tip Table, care permite plasarea diverselor controale în celule, cu posibilitatea de *merge, resize, insert* etc. Se recomandă controlul *Table* simplu, din secţiunea HTML, nu cel din secţiunea Standard, care ar necesita resurse ASP.NET suplimentare, la runtime.

Punctul de vedere în baza căruia s-au denumit evenimentele este cel al server-ului: Page.Load desemnează încărcare pagina pe server, pentru prelucrări; Page.PreRender pe server, înainte de a o face HTML, Page.Unload – plecare din server către client şi eliberare resurse pe server.

Similar stau lucrurile şi pentru Request şi Response; spre exemplu, Response.Redirect("newPage.aspx"); precizează că în răspuns, serverul cere browser-ului să facă o cerere către o altă pagină *newPage.aspx*. Spre deosebire de aceasta, Server.Transfer("newPage.aspx"); nu presupune un du-te - vino de pagină, ci serverul în loc să prelucreze pagina cerută de browser, prelucrează o alta, dar de pe acelaşi server; în browser adresa paginii nu se schimbă.

Web server –ul folosit este aici Internet Information Services [IIS] = **inetinfo.exe**

Internet Server Application Programming Interface (**ISAPI**) conţine funcţii DLL ce se încarcă dinamic, în acelaşi proces cu serverul.

Avantaj: sunt rapide, partajabile. Pot fi împărţite în două categorii:

- **filtre** – încărcate la iniţializare server şi folosite la tratarea evenimentelor la nivel de server;
- **extensii** - încărcate la prima solicitare şi apoi partajate între aplicaţii.

Dezavantaj: la eşuare, afectează tot serverul, făcând parte din acelaşi proces.

1. IIS primeşte cererea şi după extensia de fişier vede ce DLL din ISAPI deserveşte cererea. Dacă extensia este **`.aspx`**, adică ASP.NET, atunci invocă funcţia adecvată din **`aspnet_isapi.dll`** pentru tratarea cererii (în consola IIS, tab-urile ***MIME***, respectiv **Handler Mapings** pentru a vedea **asocierile dintre extensiile de fişiere şi executabilele** ce le deservesc; ***.aspx** e asociat cu ***PageHandlerFactory***).
2. Procesul `aspnet_isapi.dll` transmite cererea ASP.NET către `worker process` (**`aspnet_wp.exe`**), care o tratează;
3. `worker process` compilează fişierul `.aspx` obţinând un assembly, crează un `application domain` şi instruieşte maşina virtuala CLR (Common Language Runtime) să execute assembly-ul în contextul `application domain` creat
4. assembly foloseşte clase din FCL (Framework Class Library) pentru a rezolva cererea şi generează un răspuns;
5. `worker process` preia răspunsul generat, îl împachetează şi-l transmite procesului din aspnet_isapi.dll, care îl transmite la rândul lui serverului IIS pentru a fi înaintat clientului, la distanţă.

Uzual, structura unei pagini include trei secţiuni distincte:

- **secţiunea de directive,** prin care se stabilesc condiţiile de mediu în care pagina se va executa, instruind HTTP runtime cum să proceseze pagina, eventuale namespace-uri folosite în zona de codificare, controale noi de utilizator etc.
- **secţiunea de cod**, introdusă prin tag-ul <*script*> şi care precizează codul executabil folosit la execuţia unor comenzi din pagină (uzual precompilat, nu neapărat script, cum s-ar putea înţelege din delimitator);
- **secţiunea de machetare**, **Page layout,** reprezentând scheletul paginii şi precizând cum se mixează elementele de cod cu elementele vizuale din pagină (controale, texte etc.)

Sintaxa directivelor este unică pentru toate directivele, iar în cazul atributelor multiple acestea sunt separate cu un spaţiu; **nu trebuie separat cu spaţiu semnul egal** (=), de asignare a valorilor unui atribut:

```
<%@ Directive_Name attribute="value" [attribute="value"...] %>
```

În cadrul unui formular web, adică între tag-urile <form id="Form1" method="post" > şi `</form>` putem pune cod C#, declarat ca script:

Eventualele erori de compilare sunt raportate doar când se rulează pe aceeaşi maşină (*localhost*), dar se poate cere raportarea erorilor şi când se rulează pe un server aflat la distanţă; în acest scop în fişierul de configurare `web.config`, se pune în rubrica adecvată codul următor:

```
<compilation
defaultLanguage="c#"
debug="true"
/>
```

Rezultatul compilării este un DLL şi este pus în directorul `c\winnt\Microsoft.Net\Framework\ ...\Temporary ASP.NET Files.` (precum şi în directorul aplicaţiei c:\inetpub\wwwroot\. . . `\.bin`). Dacă nu s-au făcut modificări în sursă, la următoarea rulare nu se mai compilează.

Se pot cere mesaje detaliate (click pe **Show Detailed Compiler**).

Concret, în fişierul `Default.aspx.cs` se defineşte dinamic o clasă ***_Default*** , derivată din clasa System.Web.UI.**Page**.

`double Patrate(double nr)` dintr-unul din exemplele de mai sus, este o metodă a acestei clase, dar clasa fiind generată dinamic dispune şi de alte metode; spre exemplu. codul repetitiv din sursa de mai sus plasat între **`<%...%>`** este pasat unei metode numită **`__Render__control1()`** pentru redare.

Utilizarea unor controale Web simple

1. Se poate rula o pagina ce face adunarea a două numere, la apăsarea pe un buton:

```
private void btnAduna_Click(object sender, System.EventArgs e)
      {
            double s = double.Parse(a.Text)+double.Parse(b.Text);
            r.Text=s.ToString();
      }
```

unde `a,b` și `r` sunt trei textBox-uri.

Să se modifice programul astfel încât în funcţie de selecţia dintr-un control DropDownList, la apăsarea pe buton să se efectueze Adunare, Scădere sau nimic (None).

Să se modifice programul astfel încât să se renunţe la buton, operaţia declanşându-se automat la schimbarea selecţiei din DropDownList.

2. Aplicatie cu o macheta de introducere date, care la Save face mutarea datelor într-un textBox multiline (sau in ListBox).

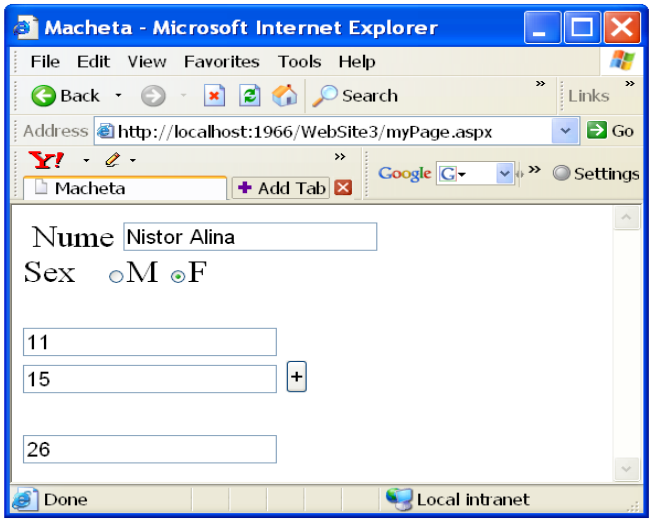
3. Pentru familiarizarea cu mediul .NET se va elabora o aplicaţie simplă, exemplificând lucrul cu controale de tipul: `Button, TextBox, RadioButton, DropDownList, AddRotator, Calendar`.

TextBox; RadioButton;Button

Mai întâi se vor aduce din Toolbox controale de tip **TextBox**, **Label**, **RadioButton** şi se vor completa corespunzător proprietăţile dorite, ca în figura de mai jos.

Se adaugă **apoi** trei textBox-uri cu identificatorii `a, b` şi `r` şi un buton inscriptionat cu „+”, care la apăsare (evenimentul Click) declanşează adunarea a două numere şi afişarea rezultatului. Funcţia pusă ca tratare a mesajului emis la de buton ar putea arăta astfel:

```
private void btnAduna_Click(object sender, System.EventArgs e)
{
      double s = double.Parse(a.Text)+double.Parse(b.Text);
      r.Text=s.ToString();
}
```



DropDownList

- Se aduce prin dragare din ToolBox un control **`DropDownList`** şi se populează adăugând câteva linii în colecţia (proprietatea) `Items`;
- i se fixează proprietatea **`AutoPostBack`** pe **`true`**, pentru a anunţa serverul de orice schimbare de selectie;
- se tratează evenimentul **`SelectedIndexChanged`** prin funcţia:

```
private void cbModel_SelectedIndexChanged(object sender, System.EventArgs e)
{
   tbTip.Text = cbModel.SelectedValue;
}
```

- funcţia preia valoarea selectată şi o pune într-un TextBox numit **`tbTip`**.

AddRotator

Fişierul **reclama.xml** se adaugă la proiect cu meniu **Project (WebSite) / Add Existing Item...**

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
<Ad>
<ImageUrl>p29.jpg</ImageUrl>
<NavigateUrl>http://localhost/WebApplication10/</NavigateUrl>
<AlternateText>Doctorat</AlternateText>
<Impressions>40</Impressions>
<Keyword>Curs</Keyword>
<Specialization>Acces INTERNET </Specialization>
</Ad>

<Ad>
<ImageUrl>p30.jpg</ImageUrl>
<NavigateUrl>http://localhost/WebApplication10/ </NavigateUrl>
<AlternateText> Biblioteca Academiei </AlternateText>
<Impressions>30</Impressions>
<Keyword>Books</Keyword>
<Specialization>Books for Professionals
</Specialization>
</Ad>

<Ad>
<ImageUrl>p31.jpg</ImageUrl>
<NavigateUrl>http://localhost/WebApplication10/</NavigateUrl>
<AlternateText> Libraria ASE </AlternateText>
<Impressions>30</Impressions>
<Keyword>Books</Keyword>
<Specialization>Academic Books</Specialization>
</Ad>

</Advertisements>
```

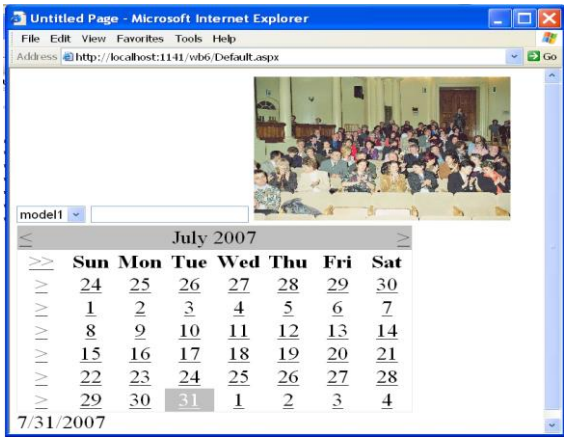
- Se adaugă un control **`AddRotator`** care este capabil să afişeze aleator, **la fiecare schimbare de pagină**, câte o imagine cu text asociat ;
- i se pune proprietatea **`AdvertisementFile`** pe valoarea `reclama.xml;`
- ne preocupăm să existe fişierele imagine citate în `reclama.xml`.

Calendar

- Se draghează un control **`Calendar`** şi i se fixează proprietatea **`SelectionMode`** pe valoarea **`DayWeekMonth`**;
- i se tratează evenimentul `SelectionChange` cu funcţia:

```
private void Calendar1_d(object sender, System.EventArgs e)
{
   string sbMesaj="";
   for( int k=0; k < Calendar1.SelectedDates.Count; k++)
         sbMesaj += Calendar1.SelectedDates[k].ToShortDateString()
         + "<br>";
   lblMesaj.Text = sbMesaj;
}
```

Funcţia preia datele selectate din calendar şi le pune într-un control Label **`lblMesaj`**.



Clasa Page

Ierarhia derivării:

```
System.Object
System.Web.UI.Control
System.Web.UI.TemplateControl
System.Web.UI.Page
```

deci clasa Page este şi un **control** având o mulţime de proprietăţi şi evenimente pe care le recunoaşte. Este şi **container** de controale, colecţia `Controls` furnizând acces la controalele de pe pagină.

Aşa cum aminteam, etapele creării paginii sunt semnalate prin evenimente ce pot fi interceptate şi folosite pentru a plasa cod de utilizator pentru executat legat de acel moment:

- **`Init`** – generat la crearea unei instanţe de tip Page; se poate folosi şi pentru a ataşa dinamic handlere altor evenimente recunoscute de pagină şi ce pot fi declanşate ulterior.

- **`Load`** – la terminarea iniţializării şi încărcarea obiectului `Page` în memoria serverului; se poate folosi şi pentru a ataşa cod de iniţializare controalelor de pe pagină.
- **`PreRender`** – declanşat înainte de redarea vizuală a paginii pe ecran; e folosit pentru eventuale retuşuri înainte de a trimite pagina în browser pentru trasare.
- **`Unload`** – la descărcarea paginii din server (page cleanup) și plecarea spre vizualizare în browser.

Tratarea evenimentelor permite aplicaţiilor să se adapteze dinamic la schimbările contextuale din mediul de rulare

> `<%...%>` e un bloc ce ține de redarea interfeţei (eventuale evaluări) şi se pune în interiorul unei metode; deci nu poate conţine definirea unei alte metode !

Tratarea unui eveniment se poate face într-una din modalităţile:

1. adăugând un **handler** la delegatul disponibil pentru acel eveniment;
2. sau **supraîncărcând metoda** (**`OnXxxx()`**) moştenită din clasa de bază.

Varianta cu **handler** este avantajoasă deoarece:

- nu cere ca tratarea să se facă într-o clasă derivată din clasa de bază (supraîncărcarea metodei (**`OnXxxx()`** se poate face doar în clasele derivate dintr-o clasă de bază);
- permite ataşarea / detaşarea **dinamică** a **mai multor metode** de tratare a aceluiaşi eveniment;
- aceeaşi funcţie de tratare se poate aplica şi altor evenimente fără a o rescrie în diverse clase.

În .NET există clasa care descrie un delegat generic, numit **`EventHandler;`** instanţierea clasei produce un obiect delegat şi presupune furnizarea prototipului funcţiilor de tratare ce le va conţine.

Declaraţia prototipului acestui delegat este **deja facută** în mediul de programare:

```
public delegate void EventHandler(object sender, EventArgs e);
```

şi arată că acest tip de delegat poate ţine referinţa oricărei funcţii ce primeşte un obiect generic şi un bloc de parametri cu argumentele specifice unui eveniment generic şi returnează `void`.

Când adăugăm unui eveniment o funcţie de tratare, se instanţiază un nou delegat, care prin constructor primeşte şi referinţa pe care o va transporta:

```
this.Load += new EventHandler(tratareLoad);
```

Se observă aici **evenimentul** (**`Load`**), **delegatul generic** (**`EventHandler`**) şi numele **funcţiei de tratare** (**`tratareLoad`**).

Mecanismul `AutoEventWireup="true"`

În aplicaţiile Web mai există un mecanism de ataşare a unor funcţii de tratarea evenimentelor unei pagini Web numit **mecanism AutoEventWireup**. Este cel de-al treilea mecanism de ataşare cod executabil unor evenimente, alături de supraîncărcarea funcţiilor moştenite din clasa de bază, respectiv mecanismul delegării.

El se activeză numai dacă pagina are atributul `AutoEventWireup = "true"` şi se bazează pe definirea de către programator a unor **funcţii de tratare cu nume impus**, de forma **`Page_EventName()`**.

Presupunem că se creează o aplicaţie Web numită AprindereAutomată. În fereastra de vizualizare se alege formatul HTML şi se modifică codul sursă pentru a arăta astfel:

```
<%@ Page Language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="true"
Inherits="AprindereAutomata.WebForm1"%>
<!--Exemplificare mecanism AutoEventWireup -->
<HTML>
      <body>
            <script runat="server">
protected void Page_Load(Object o, EventArgs e)
      {Response.Write("Mesaj din functia de tratare eveniment Load .<br>");}
protected void Page_Init(Object o, EventArgs e)
      {Response.Write("Acum se produce eveniment Page_Init .<br>");}
protected void Page_PreRender(Object o, EventArgs e)
      {Response.Write("Mesaj pentru eveniment PreRender .<br>");}
            </script>
            <hr>
      </body>
</HTML>
```

Atenţie când textul sursă se preia cu **Copy / Paste**, ghilimelele trebuie să fie cele drepte, nu "smart" !

Clauza `AutoEventWireup="true"`, altfel Visual Studio o pune implicit pe false.

Exemplul funcţionează de sine stătător, fără a necesita construire de delegaţi de evenimente şi ataşare de funcţii de tratare. Preţul plătit este că funcţiile de tatare au nume impus, corelat cu evenimentul la care se referă: **`Page_Load`**, **`Page_Init`**, respectiv **`Page_PreRender`**.

Nu se recomandă folosirea simultană a mecanismelor `delegate` şi `AutoEventWireup="true"` deoarece unele din funcţiile de tratare pot fi apelate de două ori; explicaţia constă în faptul că în cadrul mecanismului `AutoEventWireup` ASP.NET leagă automat la

delegaţii corespunzători, metodele cu numele impus, iar când programatorul ataşează metodele cu += le leagă a doua oară!

Exemplul de mai sus se putea da şi ca fişier text, ataşabil la un proiect vid, nemaifiind necesară trimiterea la fişierul cu cod sursă `Codebehind="WebForm1.aspx.cs"`, care oricum nu aducea în plus ceva major.

Mecanismul **`code-behind`** separă codul necesar descrierii interfeţei utilizator (user interface) de codul ce detaliază algoritmii de prelucrare (business logic). Prima categorie se pune în fişierul de tip **.aspx**, cea de-a doua în fişierul **.cs**; astfel este posibil şi lucru în echipă, designerii ocupându.se de proiectarea vizuală a paginii, iar programatorii de prelucrări.
Cele două fişiere sunt legate prin atributul `Codebehind` din directiva `Page`; acesta precizează fişierul cu cod sursă . Linkeditarea apare astfel ca fiind iniţiată din pagina de utilizator.

Directiva Page precizează atribute necesare compilării şi vizualizării paginii Web.
Src - citează numele fişierului sursă ce conţine clasa code-behind care este compilată dinamic la solicitarea unei pagini Web. Când folosimVisual Studio .NET nu este nevoie de acest atribut doarece clasa code-behind este precompilată.

Inherits – furnizează numele clasei code-behind ce conţine codul paginii ASPX şi din care clasa generată dinamic va moşteni prin derivare.

Desi **`Src`** furnizează aceeaşi informaţie ca atributul **`Codebehind`**; între ele există o deosebire majoră: `Src` e necesar când **compilarea se face la momentul vizualizării paginii**. Visual Studio a optat pentru **precompilarea** codului ce însoţeşte pagina Web. Pentru că mediul Visual Studio separă şi el codul de vizualizare de cel de prelucrare, are totuşi nevoie la momentul proiectării aplicaţiei de numele fişierului ce ţine codul sursă C#, pentru a-l compila din vreme; acesta e indicat prin **`Codebehind`**. Aşadar, **`Codebehind`** **este un atribut înţeles şi folosit numai de VS** (nu şi de ASP.NET la execuţia paginii) pentru localizarea fişierului sursă ce conţine definirea claselor folosite în pagină.

> **Asocierea `Src` şi `Inherits`** sugerează o pagină fără compilare prealabilă.
> **Asocierea `Codebehind` şi `Inherits`** sugerează o pagină cu precompilare

Precompilarea are următoarele avantaje:
- ne asigură că am rezovat toate erorile de compilare;
- nu mai trebuie dat beneficiarului codul sursă, ci doar dll-urile;
- chiar când pe calculatorul beneficarului există altă versiune de .NET, clasa foloseşte versiunea cu care a fost ea creată.

Meniu **Project / Show All Files** permite vizualizarea în `Solution Explorer` a tuturor fişierelor folosite în cadrul unui proiect, altfel implicit sunt vizibile doar o parte dintre ele.
Fişierul de configurare **`Web.config`** conţine toate clauzele de configurare a proiectului, inclusiv `debug = "true"`, necesară când aplicaţia nu poate fi rulată în mod depanare, căci ea a fost compilată fără opţiuni de depanare. Punând-o pe true, ne asigurăm că recompilarea se face cu adăugarea tuturor facilităţilor de depanare.

Ildasm.exe (Intermediate Language Disassembler) poate vizualiza EXE şi DLL într-un format accesibil utilizatorului.

Tipologia controalelor dintr-o aplicaţie ASP.NET

- **HTML Controls –** preluate din HTML clasic; accesul la informaţii este greoi (trebuie analizată pagina de răspuns).
- **HTML Server Controls** – identice cu cele de mai sus, dar prin **`runat = "server"`** devin accesibile prin program şi rulează pe server; asigură acces uşor la informaţii prin intermediul proprietăţilor şi evenimentelor.
- **Web Server Controls** – specifice ASP.NET; acoperă toate controalele de mai sus şi ceva în plus; oferă alt model, mai consistent, de programare.
- **Validation Controls**
- **User Controls, Composite Controls şi CustomControls** – extensii sau adaptări, create de utilizator

Web Server Controls

- orientate obiect; built-in şi recunoscute automat de browser-ele de nivel înalt;
- unele produc postback imediat la click sau la schimbare de valori;
- unele lasă mesajele primite să ajungă şi mai sus, la containerul controaleleor.

Au numele prefixat cu **asp:** , ca mai jos:
<asp:Label runat="server"/>

**Label - **
TextBox - <input type="text">, <input type="password">, <textarea>

Proprietatea ***AutoPostBack*** a unui textBox precizează când pagina va fi transmisă server-ului automat la orice modificare a textului sau când evenimentul **TextChanged** va fi tratat de client. Este efectivă doar dacă browser-ul suportă ***client-side scripting***. ASP.NET ataşează controlului un mic script ce face ca evenimentul să fie transmis serverului cu ocazia altui eveniment, uzual Click de buton.

Image <img>
`ImageUrl` indică locaţia URL în care rezidă imaginea de afişat

CheckBox şi **RadioButton - <input type="checkbox">** şi **<input type="radio">**
Proprietatea ***AutoPostBack*** precizează când pagina va fi transmisă server-ului la schimbarea stării butoanelor.
Button - <input type="submit">
LinkButton - hyperlink , <a> tratat doar de browsere ce suportă client-side scripting
ImageButton - <input type="image"> `ImageClickEventsArgs` încapsulează informatii despre coordonatele punctului din imagine unde s-a dat click.
Repeater, DataList, DataGrid suportă ***bubbling*** de evenimente, adică posibilitatea ridicării automate a evenimentelor către controlul părinte.

După modul în care sunt sesizabile la nivelul serverului, evenimentele din pagina web se împart în două categorii:

- **Postback,** care sunt anunţate imediat serverului;
- **Non-postback,** care **nu** sunt anunţate imediat serverului, pentru a nu mări excesiv traficul în reţea.

Evenimentele **non-postback** sunt evenimente mărunte, cu rol mai ales local, la nivelul paginii (spre exemplu, completarea unor rubrici într-un formular - `TextChanged`), urmând ca după mai multe astfel de evenimente să se genereze un eveniment de tip **postback (** `Click` pe un buton `Submit`**)** care să permită transferul spre server al tuturor modificărilor produse în pagină.

Controalele care generează evenimente deţin o proprietate booleană **AutoPostBack**, cu valori implicite, prin care se specifică dacă evenimentele generate de acestea sunt **postback** sau **non-postback.** După valoarea implicită a acestei proprietăţi putem distinge:

- controale **postback :** Button, Calendar, DataGrid, DataList, ImageButton, LinkButton, Repeater.
- controale **non-postback:** CheckBox, CheckBoxList, DropDownList, ListBox, RadioButtonList, RadioButton, TextBox.

Modificând valoarea proprietăţii **AutoPostBack** putem schimba dinamic "vizibilitatea" acestor evenimente la nivelul serverului. Putem observa însă, că la un moment dat, toate evenimentele generate de un control sunt fie postback, fie non-postback.

Categorii de controale			
tag HTML	**Categoria**	**Nume HTML**	**Descriere**
`<input>`	Input	HtmlInputButton HtmlInputCheckBox HtmlInputFile HtmlInputHidden HtmlInputImage HtmlInputRadioButton HtmlInputText	<input type=button | submit | reset> <input type=checkbox> <input type=file> <input type=hidden> <input type=image> <input type=radio> <input type=text | password>
`<img>`	Input	HtmlImage	Image
`<textarea>`	Input	HtmlTextArea	Text multilinie
`<a>`	Container	HtmlAnchor	Ancoră
`<button>`	Container	HtmlButton	Customizable output format, usable with IE 4.0 and above browsers
`<form>`	Container	HtmlForm	Maximum of one HtmlForm control per page; default method is POST
`<table>`	Container	HtmlTable	Tabelă, formată din linii, ce conţin celule
`<td> <th>`	Container	HtmlTableCell	Celulă de tabelă sau celulă antet de tabelă
`<tr>`	Container	HtmlTableRow	Linie de tabelă
`<select>`	Container	HtmlSelect	Meniu pull-down
	Container	HtmlGenericControl	Control HTML generic

Gestiunea datelor

- în **fişiere**, folosind clasele `File`, `FileStream`, `StreamReader`, `StreamWriter`, `BinaryReader` şi `BinaryWriter`;
- în **baze de date**, folosind clasele ADO.NET specifice fiecărui SGBD;
- în **fişiere XML**, care pot ţine atât date, cât şi descrierea lor, clasele folosite fiind `XmlNode`, `XmlDocument`, `XmlTextReader` şi `XmlTextWriter`.

Controlul **GridView** sub Visual Studio pentru aplicaţiile Web, se aseamănă cu cel din Windows Forms (DataGridView), dar are câteva particularităţi ce ţin de o mai bună gestiune a resurselor, având în vedere că BD se află pe server la distanta, iar transferul de volume mari de date către client presupune bandă largă de transfer şi necesită timp considerabil. În aplicaţiile Web, gridul:

1. permite paginarea, adică posibilitatea preluării datelor din DataSet sau din sursa de date, pe măsura defilării în sus şi în jos, în grid;
2. lucrează deconectat de la baza de date, ca şi în Windows Forms, dar disponibilizează datele ca text, implicit read-only; anumite proprietăţi ale gridului permit însă introducerea în regim de editare, când pentru linia editată se disponibilizează o linie de TextBox-uri ce mediază această editare.

Caching

În lucru cu baze de date, avem de ales între:

- a avea date care beneficiază de cele mai recente actualizări, cu preţul extragerii lor direct din baza de date;
- a ţine datele în memorie cache, mai rapid accesibilă decât baza de date, dar reflectând situaţia de la momentul citirii lor din baza de date; modificările survenite între timp ar putea fi semnificative, mai ales în cazul accesului simultan al mai multor utilizatori pe acceaşi bază.

Memoria cache se rezervă uzual pe server, **în cadrul procesului** ce execută cererile în contul aplicaţiei client. Se poate rezerva şi **în memoria globală**, când deserveşte mai mulţi clienti (sau chiar pe disc, depinzând de configurarea serverului, de către administrator).
`Cache` este un obiect în care punem date (aici întreg dataSet-ul) identificate prin nume. Funcţiile `Page_Load` şi `LeagaGrid` arată în această situaţie astfel:

```
private void Page_Load(object sender, System.EventArgs e)
{
  if(!IsPostBack)
      {// daca pagina e încărcată prima data, incarca datele in cache
            oleDbDataAdapter1.Fill(dataSet1);
            Cache["ProdCache"]=dataSet1;
            LeagaGrid();
      }

}

private void LeagaGrid()
{
      dataSet1=(DataSet)Cache["ProdCache"];
      dg.DataSource= dataSet1; dg.DataBind();
}
```

Lucrul cu un control GridView şi bază de date SqlServer sub Visual Studio

Configurare ASP.NET 3.5 pentru a lucra cu Microsoft SQL Server 2005 sau **SQL Server Express 2008**

1. Trebuie să existe instalat serverul de web IIS sau se descarcă de pe site-ul Microsoft. Testare locală se poate face şi cu **serverul de dezvoltare** furnizat de Visual .NET.
2. Se instalează VS 2008 şi implicit .NET Framework 3.5
3. Se instalează **SQL Server Express** , iar la instalare se lasă opţiunea implicită de autentificare: **Windows Authentication**, nu optăm pentru varianta SQL Authentication, unde ar fi trebuit să introducem *username* şi *password* pentru un utilizator înregistrat în prealabil în BD.

Pentru generalitate, din punctul de vedere al bazei de date, în cele ce urmează se vor aborda exemple de lucru pe trei direcţii:`

- SQL **Server Express 2008**, pe calculatorul local
- SQL **Server 2005, la distanţă**
- **Microsoft Access**, pe calculatorul local

Inregistrarea utilizatorului ASPNET la o bază de date sub SQL Server Express 2008

4. Se deschide SQL Server Management Studio Express (daca nu il aveti instalat, se poate face download de la adresa http://msdn.microsoft.com/vstudio/express/sql/download/
5. La deschidere apare fereastra **Connect to server**, în care se cer informaţiile de conectare la server (Server name: numele calculatorului dvs.\SQLExpress; Authentication:Windows Authentication).
6. In fereastra din stânga (Object Explorer), se deschide Security->Logins. Se face right click pe Logins şi alegeţi din meniul contextual New Login... In fereastra Login New, la Login name: numele calculatorului dvs.\ASPNET şi apăsaţi OK.
7. Tot în fereastra din stânga, Databases->prod->Security->Users->right click->New User...(In noua fereastra deschisa-> User Name:ASPNET; Login name: numele calculatorului dvs.\ASPNET; în fereastra de jos "Database role membership:" selectaţi db_datareader şi db_datawriter şi apăsaţi OK).

Detalii interesante puteţi găsi la adresa: http://www.spaanjaars.com/QuickDocId.aspx?quickdoc=395

Popularea unei baze de date sub SQL Server 2008 pe local, prin import din Access

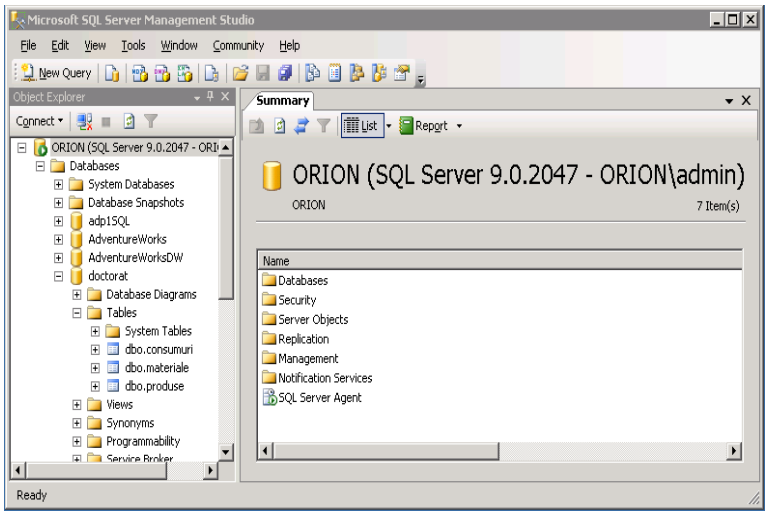
Start / All programs / MS SqlServer 2008 / Import and Export Data (32 bits); se declanşează un wizard în care se alege **Data Source MS Access**, cu **Browse** fişierul ce contine BD sursă, apoi se dau coordonatele BD destinaţie.

Pregătirea unei baze de date sub SQL Server la distanţă

Pentru lucru pe un server (în cazul nostru Orion) plasat în altă parte decât pe calculatorul local se poate proceda astfel:

4. se deschide un terminal Remote Desktop (**Run mstsc** sau Remote Desktop din Programs / Accesories sau un Remote Desktop Web Connection , de exemplu, aris.ase.ro/tsweb)
5. se indică serverul Orion sau IP 10.2.64.73 şi se cere conectarea la acesta;
6. în fereastra de conectare se indică user admin şi password meca2007/1
7. conectat la calculatorul respectiv, se activează din **Programs**, **Sql Server Management Studio**
8. urmează conectarea la baza de date de pe serverul ORION folosind Autentificare Windows

9. la conectare vedem şi baza de date doctorat, cu trei tabele deja construite şi populate cu date de test.



Dacă nu avem aceste elemente deja create, sub Sql Server **Management Studio** se pot face operaţii de genul creării unei noi baze de date, adăugării de tabele sau actualizarea datelor dintr-o tabelă. În această manieră a fost creată baza de date doctorat, cu tabelele produse, materiale şi consumuri.

Crearea şi popularea unei BD SqlServer folosind Server Explorer sub Visual Studio

- In **Server Explorer** pe **Data Connections**, mouse dreapta, **Create New Sql Server DB;**
- in fereastra care apare:
  - Server name: 10.2.64.73
  - **Use SQL Server Authentication**
  - user : **sa** password : **as**
  - se dă un nume bazei de date
- Selectând **conexiunea** creată cu această ocazie, cu mouse dreapta pe **Tables** se cere **Add new table** şi apoi se descriu câmpurile şi se salvează tabela sub un nume dorit.
- Este bine ca unul din câmpuri să fie declarat drept cheie (cu icon sau mouse dreapta / **Set Primary key**) pentru a beneficia ulterior de generare automată a comenzilor INSERT, UPDATE, DELETE.
- Selectând tabela creată, cu mouse dreapta se alege **Show Table Data**, iar în gridul care apare se introduc pe rând tuplurile dorite.
-

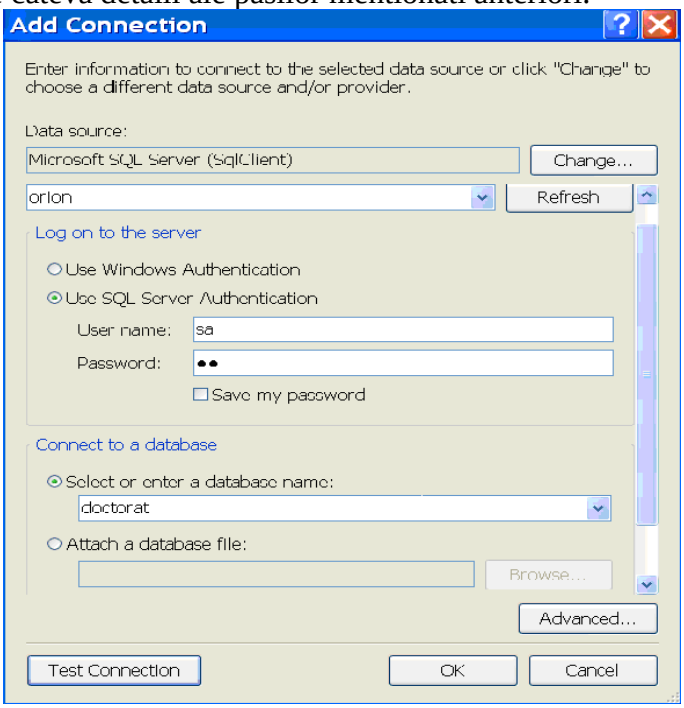
Smart tag – este un artificiu semnalat printr-o mică săgeată plasată în coltul dreapta-sus al unor controale complexe (GridView, TreeView, Calendar etc.) prin care se accede la un meniu de **configurare rapidă** a controlului.

Etape de lucru cu un control grid

1. dacă nu există, se **crează o bază** de date ***masteratBDI*** sub SqlServer**.** Ne asigurăm că ea conţine şi tabela ***produse***, în care există câmpurile: *codp, denum, pret.*
2. în Designer, se dragheaza un control **GridView** din ToolBox (uzual in categoria Data) şi se denumeşte ***dg***;
3. **optional**, cu butonul mouse dreapta pe grid sau din smart-tag, se poate cere **AutoFormat**, având posibilitatea de a alege dintre mai multe formate de grid şi scheme de colorare; în **Properties**, i se pot fixa apoi diverse alte proprietăţi de vizualizare (headerStyle, backColor, border style etc.); în general vorbind,
   - proprietăţile din grupul Appearance afectează caracteristicile estetice la **nivelul întregului grid**;
   - proprietăţile din grupul Styles afectează caracteristicile la **nivel de linie** din grid;
   - proprietăţile la **nivel de coloană** din grid pot fi stabilite doar prin intermediul **smart tag** –ului (colţul dreapta sus, al gridului), de unde se va selecta Edit Columns.
4. din **smart-tag**, Edit columns, chiar înainte de legare grid la o sursa de date, se debifeaza ***Autogenerate fields***;
5. din **smart-tag**, **Choose data source** se alege <New Data Source>, care declanseaza un wizard pentru configurare sursa de date; configurarea se putea face si prin dragarea din ToolBox a unui control **SqlDataSource** (dacă nu există în ToolBox, poate fi adus cu buton dreapta mouse **Choose Items**... din galeria de controale), denumit implicit *SqlDataSource1**;*** îl vom redenumi în Properties, ***SqlDataSourceOrion***; daca esueaza configurarea sau este incorect facuta, se reconfigurează obiectul *SqlDataSourceOrion* cu buton dreapta mouse **Configure Data Source,** care declanşează din nou wizard-ul;
6. configurarea presupune:
   - alegerea unei conexiuni din Server Explorer, sau configuram una noua, având grijă să selectăm corect provider-ul Sql Server, dar ar putea fi şi Oracle, MS Access etc.
   - se indică baza de date (sau fişierul conţinând baza de date, dacă BD e orientată pe fişiere ca în Access, Fox etc) sau se doreste lucru cu fisier detasat de la BD;
   - se compune fraza SELECT folosită la încărcarea datelor din baza de date; pe butonul **Advanced** pot fi configurate şi comenzile UPDATE, INSERT şi DELETE, dacă se doreşte şi acest lucru. Este posibilă generarea automată a acestor comenzi doar dacă unul din câmpuri (în cazul nostru `codp`) este declarat **câmp cheie**; altfel trebuie mai întâi sub SQL Server Management Studio modificata descrierea tabelei, declarand un camp drept cheie.

   Observaţie: pentru o sursă de date Access se poate folosi un obiect mai simplu, de tip *AccessDataSource*.
7. Daca definirea si legarea sursei de date la grid se termina cu succes, coloanele din tabela aleasa apar in smart-tag grid / Edit columns atat ca **BoundField**, cat si la **Dynamic**; daca nu, se dezleaga gridul si se re-leaga la sursa de date existenta, solicitand apoi din smart-tag / Edit columns / Refresh schema.
8. Se adauga la grid, prin smart-tag / Edit columns / **Command field** butoanele pentru Edit si Delete aferente cate unei linii din grid; se testeaza functionalitatea lor; daca nu functioneaza, cel mai probabil nu au fost generate automat comenzile Update si Delete care sunt declansate de butoanele Edit si Delete ale gridului (vezi pas 6).

Prezentam in continuare cateva detalii ale pasilor mentionati anteriori.



Se poate alege opţiunea ca şirul de conexiune să poată fi păstrat în fişierul web.config, fiind accesibil astfel din orice pagină a aplicaţiei web; în acest caz, se vor memora **automat** în web.config numele dat conexiunii şi valoarea propriu-zisă, printr-o secvenţă de genul:

```
<connectionStrings>
      <add name=" masteratBDIConnectionString" connectionString="Data
Source=orion;Initial Catalog=masteratBDI;User ID=sa;Password=as"
providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Acest lucru, ne permite ulterior să folosim numele doctoratConnectionString pentru a localiza legătura la baza de date doctorat.

9. Controlul GridView poate deja folosi acum sursa de date creată, fixându-i-se acestuia proprietatea Data Source pe *SqlDataSourceOrion*; la rulare gridul va afişa datele din tabela aleasă.

La legarea sursei de date la un grid, partea declarativă a gridului (vezi codul sursă în Designer, tab-ul Source) se dezvoltă rapid, prin preluarea informaţiilor despre coloanele sursei de date. După legare, denumirile generice ale coloanelor din grid pot fi actualizate rapid prin click pe Refresh Schema din "smart tag".

dg.AutoGenerateColumns pe true, dacă vrem să apară toate coloanele şi în ordinea din sursa de date.
dg.AutoGenerateColumns pe false şi adăugăm în Properties, Columns (sau din smart tag, Edit Columns) doar ce coloane dorim şi în ce ordine dorim; procedăm aşa şi când adăugăm coloane cu butoane de editare a datelor din grid.

Coloanele au conţinutul de un anumit tip; cele mai folosite tipuri de coloane sunt:

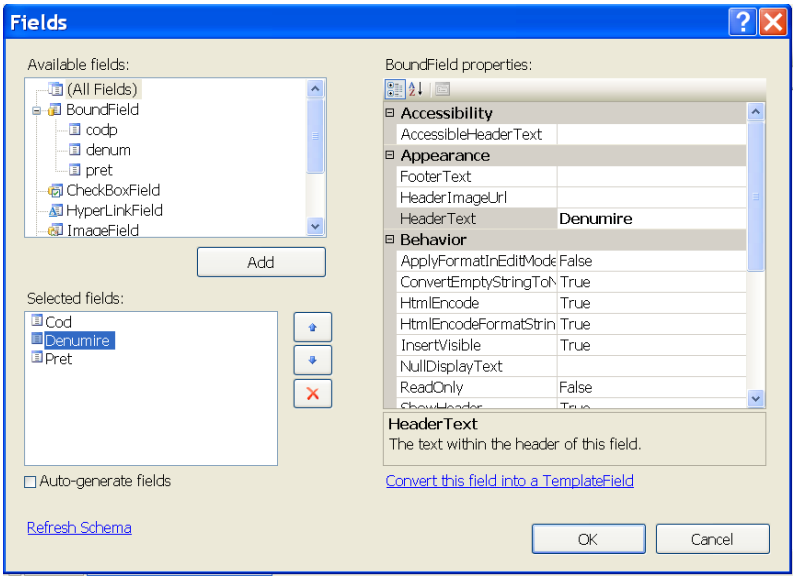
- BoundField – când coloana este legată, adică afişează conţinutul unui câmp din sursa de date;
- ButtonField – când coloana afişează câte un buton pentru fiecare item din listă.

- CheckBoxField – când coloana afişează un *checkBox* pentru fiecare item din listă, semnificând true/false, pentru câmpurile cu valori logice, stocate la nivel de bit, în SQL Server;
- CommandField – când coloana afişează butoane pentru **selecţie, editare, ştergere** etc.

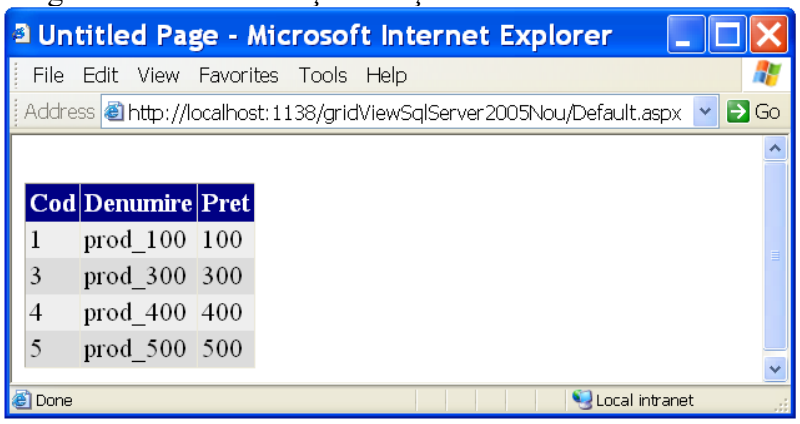
Se observă că pentru a nu crea confuzie de terminologie între coloană tabelă şi coloană grid, coloanele din grid au fost denumite câmpuri (field).

Se pot combina ambele variante, caz în care coloanele autogenerate sunt puse primele.
O tehnică mai subtilă ar fi următoarea:

- se lasă dg.AutoGenerateColumns pe false aşa cum e pusă implicit la dragarea obiectului;
- la legarea sursei nu se observă nimic în grid, dar apăsând Refresh Schema aferentă controlului data source, AutoGenerateColumns comută **automat** pe true şi mediul adaugă câte o coloană de tip **BoundField** pentru fiecare câmp din data source;
- putem acum să schimbăm ordinea coloanelor, sau în BoundField Properties denumirile din cap de coloană, să ştergem coloanele care nu ne interesează, sau să le păstrăm făcându-le invizibile (doar programatic): dg.Columns[2].Visible = false;



La rulare, gridul este legat automat la sursă şi va afişa datele din tabela indicată în fraza Select.



Paginarea gridului pentru a afişa doar o parte din date

- cu **smart tag** (Enable Paging bifat) sau în **Properties** (Allow Paging *true*), se declară opţiunea ca gridul să poată afişa pe mai multe pagini informaţia;
- în **Properties,** la PagerSettings / Mode se stabileşte modul de navigare între pagini (Numeric, salt direct la pagina dorită) şi numărul de indecşi de pagină afişaţi simultan (PageButtonCount, 10);
- în **Properties**, la Page Style se precizează câte înregistrări sunt vizibile pe o pagină (Page Size, 4);

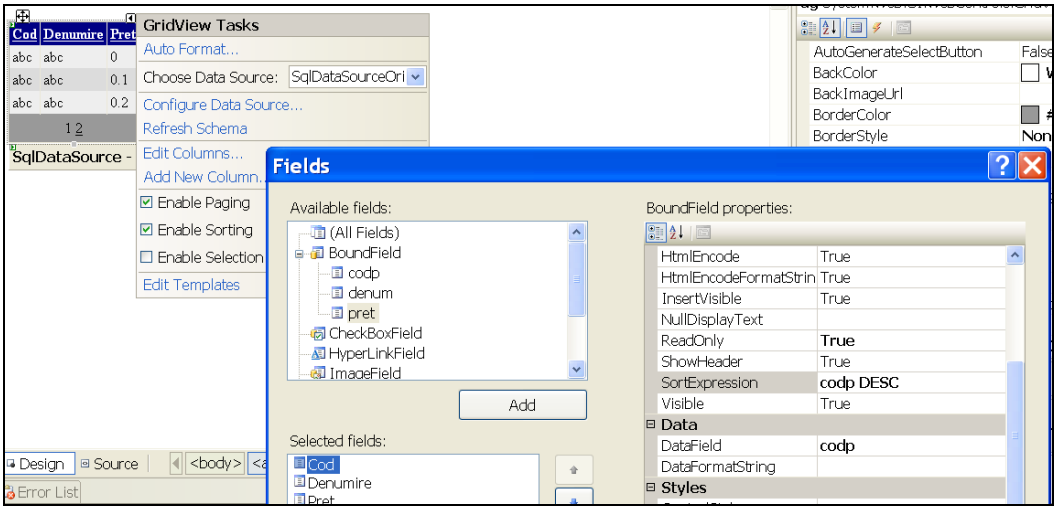
- CurrentPageIndex va conţine indexul paginii curent selectată;
- la schimbarea paginii se generează automat (şi poate fi tratat) evenimentul PageIndexChanged.

Această paginare automată se realizează uşor datorită moştenirilor prin derivare din clasa ICollection şi este posibilă pentru un GridView bazat pe o sursă de date cu proprietatea **DataSourceMode** pusă pe **DataSet**, deoarece doar acesta se află pe filiera de moştenire amintită.

Sortarea datelor din grid după una sau mai multe coloane

Sortarea datelor din grid se poate face:

a. chiar din **fraza select**, prin care se încarcă dateSet-ul; această comandă poate conţine şi clauza **ORDER BY**. Are dezavantajul că va sorta datele ori de câte ori le reîncarcă din BD, dar are avantajul unei sortări rapide;
b. folosind obiectul **DataView** asociat dataSet-ului; orice dataSet are o vizualizare implicită sau explicită; un obiectul **DataView** e specializat pe vizualizări sub diverse forme de prezentare, inclusiv sortări complexe; concret, proprietatea **DataSource** a gridului poate cita obiectul **DataView**, care asigură vizualizarea datelor din dataSet.
c. folosind facilităţi ale gridului (AllowSorting pe true); cu **smart tag** se alege **Edit Columns;** pe coloanele dorite, se stabileşte proprietatea **SortExpression**, care permite citarea unei expresii de sortare, formată din nume de câmpuri însoţite de cuvintele cheie ASC şi DESC; eventual se cere şi Refresh Schema pentru a reactualiza schema de legare a câmpurilor.



La execuţie, dând click pe antetul unei coloane se face sortarea după expresia de sortare corespunzătoare câmpului respectiv.

Editarea datelor din grid

Ideea de bază a actualizării în grid constă în a adăuga o coloană cu butoane de comandă, câte unul pentru fiecare linie din grid; se tratează evenimentul *Click* buton pentru a executa operaţia corespunzătoare butonului liniei respective şi se reîncarcă gridul automat.

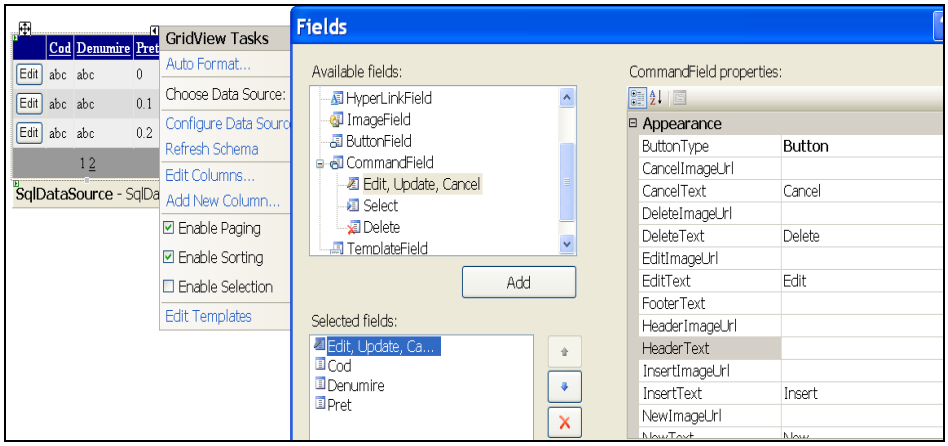
În cazul editării, apăsarea butonului Edit declanşează transformarea celulelor legate de câmpuri non-cheie, în textBox-uri. Butonul Edit se substituie pe durata editării cu alte două butoane, Update (apăsat când dorim ca modificările să fie reţinute) şi Cancel (când se renunţă la modificările operate în textBox-uri).

Paşii de lucru:

- cu **gridul** selectat, în **smart tag / Edit Columns** se adaugă cu **Add** o nouă coloană de data acesta de tip Command Field, mai precis un câmp „Edit, Update, Cancel”; s-ar fi putut

realiza acelaşi lucru şi cu **Properties** / AutoGenerateEditButton pe true, dar aveam mai puţine posibilităţi de control asupra câmpului;

- cu săgeată sus-jos se alege poziţia câmpului nou introdus, în raport cu celelalte existente în grid, iar din fereastra din dreapta se stabilesc alte proprietăţi (spre exemplu `ButtonType` se alege `Button`, ca mod de apariţie), ce să scrie pe butoanele respective etc.



- obiectul **SqlDataSource** are printre proprietăţi şi UpdateQuery, ce conţine o comandă de UPDATE, care poate fi de tip text sau procedură catalogată; în cazul nostru, textul comenzii citează câmpurile şi parametrii asociaţi (puşi implicit, prin mecanismul de legare date şi denumiţi după numele câmpurilor, plus un prefix):

UPDATE produse SET denum =@denum, pret = @pret WHERE (codp = @codp)

Deşi analizînd codul sursă din fişierul aspx constatăm că parametrii se numesc la fel şi pentru comanda UPDATE şi pentru INSERT sau DELETE, ei sunt ţinuţi în colecţii diferite şi deci nu pot fi confundaţi.

```
<DeleteParameters>
            <asp:Parameter Name="codp" Type="String" />
</DeleteParameters>
<UpdateParameters>
            <asp:Parameter Name="denum" Type="String" />
            <asp:Parameter Name="pret" Type="Double" />
            <asp:Parameter Name="codp" Type="String" />
</UpdateParameters>
<InsertParameters>
            <asp:Parameter Name="codp" Type="String" />
            <asp:Parameter Name="denum" Type="String" />
            <asp:Parameter Name="pret" Type="Double" />
</InsertParameters>
```

Gridul fiind legat la o sursă de date, DataBinding-ul funcţionează în ambele sensuri, astfel încât parametrii aparţinând sursei de date vor fi încărcaţi automat cu valorile modificate din grid.

- vor fi declanşate acum, înainte de actualizarea propriu-zisă, şi evenimentele asociate acţiunilor de **modificare** şi **ştergere**, astfel încât li se pot asocia funcţii de tratare: **`dg_RowUpdating`**, **`dg_RowCancelingEdit`**.

  Uzual se pun aici diverse validări specifice; dacă se preia în această funcţie chiar sarcina actualizării (spre exemplu, folosind obiecte SqlCommand şi scriind în bază cu cmd.ExecuteNonQuery, se va inhiba actualizarea **şi** cu comanda UPDATE-ul de pe sursa de date, punând în **`dg_RowUpdating`** instrucţiunea e.Cancel = true.

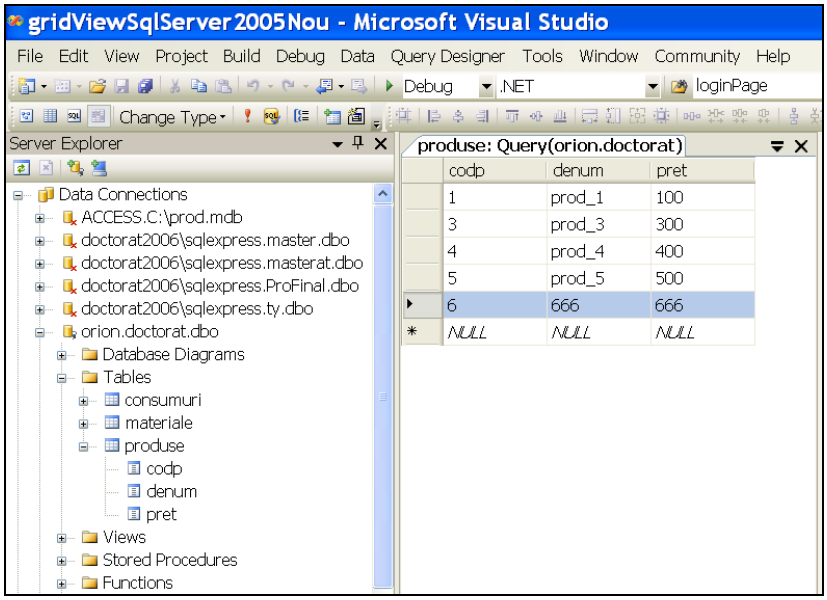
Se pot face diverse combinaţii; de exemplu pentru funcţia de **inserare** se pot încărca unul câte unul parametrii cu valori (nu funcţionează Binding automat căci nu avem linie nouă în grid) şi apoi se lansează metoda `ExecuteNonQuery` a obiectului de `InsertCommand`, aparţinând `SqlDataSource`; în felul acesta nu mai este nevoie să declarăm alt obiect SqlCommand, să-l echipăm cu parametri, conexiune etc.

Ştergerea datelor din grid

- în grid, cu **smart tag / Edit Columns** se adaugă cu **Add** o nouă coloană de tip Command Field, mai precis un câmp „**Delete**”;
- cu săgeată sus-jos se poziţionează câmpul nou introdus, ca ultimă coloană din grid, iar din fereastra din dreapta se stabilesc alte proprietăţi ale acestuia;
- SqlDataSource are printre proprietăţi şi DeleteQuery, ce conţine o comandă DELETE; o putem edita direct sau o putem construi cu QueryBuilder, încât să arate astfel:

```
DELETE FROM produse WHERE (codp = @codp)
```

- la rulare, prin apăsarea butonului Delete, este ştearsă înregistrarea curent selectată în grid; pentru testarea ştergerii, **pot fi adăugate noi înregistrări fictive, folosind Server Explorer**, selectând conexiunea către orion.doctorat.dbo, tabela produse şi beneficiem de un gen de „client SQL” cu care putem opera direct modificări pe baza de date.



DataKeyNames precizează din ce câmpuri va prelua valoarea în dicţionarul Keys
în dicţionarele NewValues şi OldValues valorile se pun

O pagină poate conţine extrem de multe controale; ele pot genera o mulţime de evenimente, care generează la rândul lor cereri multiple către server, mărind exagerat traficul client-server. În plus, odată cu argumentele evenimentului trebuie să circule şi alte informaţii din pagină (spre exemplu starea controalelor) pentru a se putea reconstitui pagina la momentul primirii răspunsului de la server.

Pentru diminuarea frecvenţei traficului şi volumului de date transferate se recomandă:

- evitarea populării în exces a paginii, cu controale;

- setarea proprietăţii **`EnableViewState`** a controalelor pe `false`, când nu este nevoie de păstrarea stării lor între două configurări ale paginii.

Adăugarea datelor în grid

Proprietatea *DataSourceMode* a sursei de date precizează dacă ea lucrează cu *DataSet* sau *DataReader*, drept container de date. În principiu, dacă sursa de date are proprietatea DataSourceMode pe valoarea *DataSet*, putem recompune dataSet-ul sau tabela de date cu care lucrează *SqlDataSource*, după modelul oferit de funcţia de mai jos:

```
protected void btnPullDataSet_Click(object sender, EventArgs e)
    {
        string conexiunea=
            "Data Source=orion;Initial Catalog=doctorat;User ID=sa;Password=as";
        string frazaSelect="select * from produse";
        SqlDataSource SqlDataSource2= new SqlDataSource(conexiunea,frazaSelect);
        DataSourceSelectArguments args = new DataSourceSelectArguments();
        DataView dataView1 = (DataView)SqlDataSource2.Select(args);
        DataTable dataTabl1 = dataView1.ToTable(); ds.Tables.Add(dataTabl1);
      //    dg.DataSourceID = null;
        dg.DataSource = ds; dg.DataBind();
    }
```

Funcţia de inserare nu este implementată direct prin controlul GridView, astfel încât vom folosi *SqlDataSource* şi vom aborda programatic problema, declarând şi instanţiind obiecte de tip *SqlConection*, şi *SqlCommand*. Vom adăuga namespace-urile care definesc clasele aferente acestor controale

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

Adăugăm un buton, pe al cărui eveniment de click legăm funcţia de adăugare a unei noi înregistrări:

```
public void btnAdd_Click(object sender, System.EventArgs e)
    {
        DataSourceSelectArguments args = new DataSourceSelectArguments();
        DataView dataView1 = (DataView)SqlDataSourceOrion.Select(args);
        DataTable dataTabl1 = dataView1.ToTable();
        DataRow linNoua;          linNoua = dataTabl1.NewRow();
        int cateRecords = dataTabl1.Rows.Count, maxCod = 0, crtCod;
        for (int i = 0; i < cateRecords; i++)
        {
            crtCod = Convert.ToInt32(dataTabl1.Rows[i][0]);
            if (crtCod > maxCod) maxCod = crtCod;
        }
        maxCod++; linNoua["codp"] = "" + maxCod;
        // cod in secventa, dupa cel mai mare existent
        linNoua["denum"] = "pr_" + maxCod; linNoua["pret"] = maxCod * 100;

        // aducere la zi si in baza de date
        SqlConnection con = new
SqlConnection(SqlDataSourceOrion.ConnectionString);
            con.Open();
        SqlCommand cdaInsert = new SqlCommand(
            "INSERT INTO produse(codp, denum, pret) " +
             " VALUES (@codp, @denum, @pret)", con);

        cdaInsert.Parameters.Add(new System.Data.SqlClient.SqlParameter(
            "@codp", System.Data.SqlDbType.NVarChar, 50, "codp"));
        cdaInsert.Parameters.Add(new System.Data.SqlClient.SqlParameter(
            "@denum", System.Data.SqlDbType.NVarChar, 50, "denum"));
```

```
        cdaInsert.Parameters.Add(new System.Data.SqlClient.SqlParameter(
                    "@pret", System.Data.SqlDbType.Float, 8, "pret"));
       // da.InsertCommand = cdaInsert;
        cdaInsert.Parameters["@codp"].Value = linNoua["codp"];
        cdaInsert.Parameters["@denum"].Value = linNoua["denum"];
        cdaInsert.Parameters["@pret"].Value = linNoua["pret"];

        if (cdaInsert.ExecuteNonQuery() > 0)
            { mes.Text = ("Inserare cu succes "); dg.DataBind(); }
        else mes.Text = ("Esec la inserare ");
    }


}
```

Se observă că am folosit doar principiu de extragere a dataSet-ului sursei de date, nu întreaga funcţie, oprindu-ne doar la nivel de tabelă, pentru a-i adăuga o linie nouă.

Dacă se cere *Refresh* pe pagină, aceasta este retrimisă în server; primim un avertisment în acest sens de la browser şi într-adevăr constatăm că va insera câte o linie la fiecare *Refresh* !!

Alte detalii şi alte modalităţi de lucru cu un GridView puteţi găsi consultând exemplul de proiect complet sub VS 2008, ce va fi prezentat într-un capitol viitor.

Vizualizarea grafică a datelor folosind un control de tip chart

Pentru varietate se prezintă două controale de tip chart, diferite:

- un control open source, disponibil pe Internet, ca sursă şi ca biblioteci DLL, **ZedGraph** (**http://www.codeproject.com/KB/graphics/zedgraph.aspx**)
- unul furnizat drept componentă Web de către Office (Office Web Component 11.0, control similar chart-ului din Excel); trebuie să dispunem de controalele aduse de către MS Office; dacă nu, controalele pot fi descărcate de pe site-ul Microsoft.

În cazul folosirii controlului ZedGraph în aplicaţia care conţine un GridView şi o sursă de date, paşii de lucru ar putea fi următorii:

- In **Solution Explorer** cu buton dreapta mouse se cere **Add Reference** şi se localizează cu **Browse** şi se leagă la proiect cele două fişiere ZedGraph.dll şi ZedGraph.Web.dll, descărcate de pe Internet şi care conţin clasele cu care lucrează controlul web.
- Se construieşte **în directorul aplicaţiei un subdirector de lucru** numit ***ZedGraphImages***, în care controlul îşi compune imaginile grafice.
- Se adaugă o pagină nouă pe care o vom numi **WebFormZedGraph (Website / Add New Item** şi alegem **WebForms)** şi pe ea (fişierul WebFormZedGraph.aspx.cs) se pune codul din funcţiile **`Page_Load`** (care leagă funcţia de tratare a evenimentului `RenderGraph`) şi **`OnRenderGraph`** (care face reprezentarea grafică propriu-zisă). După cum se observă din cod, datele se preiau dintr-un DataSet adus din Cache, unde a fost pus de către pagina principală a aplicaţiei.

```
protected void Page_Load(object sender, EventArgs e)
 {
   this.ZedGraphWeb1.RenderGraph +=
           new ZedGraph.Web.ZedGraphWebControlEventHandler(this.OnRenderGraph);
 }

private void OnRenderGraph(ZedGraph.Web.ZedGraphWeb z,
                System.Drawing.Graphics g, ZedGraph.MasterPane masterPane)
  {

      DataSet ds = (DataSet)Cache["ProdCache"];
      GraphPane myPane = masterPane[0];
      myPane.Title.Text = "";
      myPane.XAxis.Title.Text = "Produs"; myPane.YAxis.Title.Text = "Pret";
      Color[] colors =
          {
            Color.Red, Color.Yellow, Color.Green, Color.Blue,
            Color.Purple,Color.Pink,Color.Plum,Color.Silver, Color.Salmon
          };

      if (Request.QueryString["tip"] != null)
      {
          List<string> listaX = new List<string>();
          PointPairList list = new PointPairList();
          int i = 0;
          foreach (DataRow r in ds.Tables[0].Rows)
          {
              listaX.Add(r[1].ToString()); // denumire produs
              list.Add(0, (double)r[2], i++); // pret
          }

          switch (Request.QueryString["tip"])
          {
              case "Bare":
                  {
                      BarItem myCurve = myPane.AddBar(
```

```
                                        "Curve 2", list, Color.Blue);
                    myCurve.Bar.Fill = new Fill(colors);
                    myCurve.Bar.Fill.Type = FillType.GradientByZ;
                    myCurve.Bar.Fill.RangeMin = 0;
                    myCurve.Bar.Fill.RangeMax = list.Count;
                    myPane.XAxis.Type = AxisType.Text;
                    myPane.XAxis.Scale.TextLabels = listaX.ToArray();
                    break;
                }

            case "Bare3D": break;
            case "Pie3D": break;
            case "Linie":
                {
                    LineItem curve = myPane.AddCurve(
                          "Spline", list, Color.Red, SymbolType.Diamond);
                    curve.Line.IsSmooth = true;
                    curve.Line.SmoothTension = 0.5F;
                    curve.Line.Width = 2;
                    curve.Symbol.Fill = new Fill(Color.White);
                    curve.Symbol.Size = 10;

                    myPane.XAxis.Scale.TextLabels = listaX.ToArray();
                    myPane.XAxis.Type = AxisType.Text;
                    break;
                }
            case "Pie":
                {
                    i = 0;
                    foreach (DataRow r in ds.Tables[0].Rows)
                    {
                        PieItem segment1 = myPane.AddPieSlice((
                          double)r[2], colors[(i++) % colors.Length],
                          Color.White, 45f, (i % 2 == 0) ? 0.2 : 0,
                                                  r[1].ToString());
                    }
                   break;
                }
        }

    }
}
```

După cum se observă, funcţia `OnRenderGraph` este legată în `Page_Load` la delegatul evenimentului `RenderGraph`, aparţinând controlului.

```
this.ZedGraphWeb1.RenderGraph +=
            new ZedGraph.Web.ZedGraphWebControlEventHandler(this.OnRenderGraph);
```

- Cu mouse dreapta, **Choose Items** pe ToolBox, cu Browse, se aduce controlul ZedGraph şi în trusa cu instrumente grafice
- prin dragare se include controlul în pagina WebFormZedGraph;
- se adaugă la începutul paginii WebFormZedGraph şi using ZedGraph, respectiv using System.Collections.**Generic** dacă nu există, căci se lucrează cu astfel de colecţii.
- Pot fi stabilite diverse proprietăţi ale controlului folosind fereastra **Properties**; ca să păstrăm cât mai multe valori implicite, se poate draga controlul direct în **codul html** şi se adaugă tot aici doar proprietăţile `Height = "500" Width = "500"`

- Se adaugă un buton pentru **revenirea în pagina principală** a aplicaţiei, după vizionarea graficului; evenimentul `click` al butonului va avea drept funcţie de tratare:

```
    protected void btnBack_Click(object sender, EventArgs e)
        {
            Response.Redirect("default.aspx");
        }
```

- Pentru variaţie se pune pe prima pagină şi un combobox (**`DropDownList`**) din care se alege tipul graficului; pentru aceasta colecţia Items a comboBox-ului va fi populată cu denumiri de tipuri posibile de grafice: `Bare, Bare3D, Pie, Pie3D, Linie` etc. Controlul combo trebuie să aibă proprietatea `AutoPostBack = true`, pentru a anunţa serverul despre evenimentul de schimbare a tipului de grafic, obligându-l astfel să retraseze graficul la fiecare reîncărcare a paginii principale. Pe evenimentul de schimbare index selecţie în `DropDownList1`, se extrage setul de date din sursa de date şi se pune în `Cache`, după care se transferă controlul către pagina `webFormZed.aspx`, împreună cu tipul graficului ales:

```
protected void DropDownList1_SelectedIndexChanged
                                        (object sender, System.EventArgs e)
    {
        DataSourceSelectArguments args = new DataSourceSelectArguments();
        DataView dataView1 = (DataView)SqlDataSourceLocal.Select(args);
        DataTable dataTabl1 = dataView1.ToTable();
        DataSet ds = new DataSet(); ds.Tables.Add(dataTabl1);
        Cache["ProdCache"] = ds;
        Response.Redirect("webFormZed.aspx?tip=" +
                                this.DropDownList1.SelectedItem.Text);
    }
```

*
* *

In cazul folosirii controlului chart din Office, paşii de parcurs sunt următorii:

- In Solution Explorer cu buton dreapta mouse se cere **Add Reference** şi se alege din pagina COM, **Microsoft Office Web Component 11.0.** Dacă avem o aplicaţie mai veche care adusese deja o copie a referinţei, o putem selecta pe aceasta, cu *Browse*.

- Depinzând de versiune (vezi **Properties** când ţinem selectată noua referinţă) adăugăm şi `using`-ul corespunzător bibliotecii adusă ca referinţă:
  `using Microsoft.Office.Owc11;` sau `using OWC11;`
  sau altceva, în funcţie de controlul de chart folosit.
- Se adaugă o pagină nouă (**Website / Add New Item** şi alegem **WebForms**) şi pe ea se pune codul din **WebFormGrafic.aspx.cs** al aplicaţiei webChart. Datele se preiau dintr-un DataSet adus din Cache sau **se pun ca string,** cifrele fiind separate cu virgulă.

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using OWC11;
public partial class webFormGrafic : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        DataSet ds = (DataSet)Cache["ProdCache"];
```

```
        //Obiect ChartSpace container de chart-uri
        ChartSpace objCSpace = new ChartSpaceClass();
        //Add un chart si-i stabileste tipul
        ChChart objChart = objCSpace.Charts.Add(0);
        if (Request.QueryString["tip"] != null)
        {
            switch (Request.QueryString["tip"])
            {
                case "Bare": objChart.Type =
ChartChartTypeEnum.chChartTypeBarStacked; break;
                case "Bare3D": objChart.Type =
ChartChartTypeEnum.chChartTypeBar3D; break;
                case "Linie": objChart.Type =
ChartChartTypeEnum.chChartTypeSmoothLine; break;
                case "Pie3D": objChart.Type =
ChartChartTypeEnum.chChartTypePie3D; break;

            }

        }
        else
            objChart.Type = ChartChartTypeEnum.chChartTypePie3D;

        // Titlu si legenda
        objChart.HasTitle = true;
        objChart.Title.Caption = "Nivelul preturilor";
        objChart.HasLegend = false;
        //objChart.Legend.Border.DashStyle = ChartLineDashStyleEnum.chLineDash;
        //objChart.Legend.Position =
ChartLegendPositionEnum.chLegendPositionRight;
        //Populare chart cu date din dataSet
        string strCategory = "";
        string strValue = "";
        foreach (DataRow r in ds.Tables[0].Rows)
            {
                strCategory += r[1].ToString() + ","; // denumire produs
                strValue += r[2].ToString() + ",";    // pret
            }

        if (strCategory != "") strCategory =
strCategory.Remove(strCategory.Length - 1, 1);
        if (strValue != "") strValue = strValue.Remove(strValue.Length - 1, 1);

         //Adauga o serie de date la colectia de serii
        objChart.SeriesCollection.Add(0);
        //leaga  denumirile de pe Ox si valorile de pe Oy
        objChart.SeriesCollection[0].SetData(ChartDimensionsEnum.chDimCategories,
            (int)ChartSpecialDataSourcesEnum.chDataLiteral, strCategory);
        objChart.SeriesCollection[0].SetData(ChartDimensionsEnum.chDimValues,
            (int)ChartSpecialDataSourcesEnum.chDataLiteral, strValue);

        // "prepara" pagina introducand gif-ul dat de chart
        Response.ContentType = "image/gif";
        Response.BinaryWrite((byte[])objCSpace.GetPicture("gif", 500, 400));
        Response.End();


    }
}
```

- pagina principală va conţine un obiect **Image** adus din **ToolBox,** care va afişa chart-ul furnizat de **WebFormGrafic,** deoarece are proprietatea `ImageURL = WebFormGrafic`, pusă prin cod**,** folosind o cale relativă la directorul aplicaţiei.

Graficul e furnizat deja "preparat", doar îl extragem ca gif din spaţiul chart-ului; putem avea mai multe chart-uri în acelaşi spaţiu şi le putem activa pe rând, câte unul.

- Pentru variaţie se pune pe prima pagină şi un combobox (**DropDownList**) din care se alege tipul graficului; pentru aceasta colecţia Items a comboBox-ului va fi populată cu denumiri de tipuri posibile de grafice: `Bare, Bare3D, Pie, Pie3D, Linie` etc. Controlul combo trebuie să aibă proprietatea `AutoPostBack = true`, pentru a anunţa serverul despre evenimentul de schimbare a tipului de grafic, obligându-l astfel să retraseze graficul la reîncărcarea paginii principale. Pe evenimentul de schimbare index selecţie în `DropDownList1`, se precizează şi prin cod sursă că imaginea va prelua ca sursă graficul din pagina `WebFormGrafic.aspx` indicând în plus prin *QueryString* şi tipul de grafic ales:

```
protected void DropDownList1_SelectedIndexChanged
             (object sender, System.EventArgs e)
 {
     DataSourceSelectArguments args = new DataSourceSelectArguments();
     DataView dataView1 = (DataView)SqlDataSourceOrion.Select(args);
     DataTable dataTabl1 = dataView1.ToTable();
     DataSet ds = new DataSet(); ds.Tables.Add(dataTabl1);
     Cache["ProdCache"] = ds;
     this.Image1.ImageUrl =
        "WebFormGrafic.aspx?tip=" + this.DropDownList1.SelectedItem.Text;
 }
```

La adăugarea referinţei în proiect, se poate bifa opţiunea **`Copy to local`**, prin care controlul este copiat în directorul aplicaţiei, astfel încât să potă fi testată aplicaţia şi pe o maşină care nu are controalele din Office 2003 instalate.

*

* *

Proceduri stocate

- **Exploatarea bazei de date folosind obiecte de tip Command**
- **Lucru cu procedurile stocate**

Exploatarea bazei de date folosind obiecte de tip Command

Am observat până acum că obiecte precum `DataAdapter`, respectiv `SqlDataSource` au comenzi pe care le ţin ca proprietăţi ce sunt de fapt referinţe către obiecte de tip XxxCommand (**SqlCommand, OleDbCommand, OdbcCommand etc.**).

Obiectele de tip XxxCommand oferă în fond unica modalitate de acces la date, numai că ele:

- pot echipa un **DataAdapter** (sau DataSource) pe care îl ajută, după caz, să umple un DataSet cu înregistrări (metoda **Fill** a adaptorului) sau să opereze modificările, ştergerile sau inserările în baza de date (metoda **Update** a adaptorului);
- pot fi folosite independent, caz în care pot genera la execuţie un **DataReader**, container de date, care furnizează înregistrare cu înregistrare, *read only;*

Obiectele de tip Command sunt purtătoare ale unor comenzi textuale de tip SELECT, INSERT, DELETE, UPDATE din limbajul SQL sau ale unor nume de proceduri, pe care le pot lansa în execuţie.

Comenzile de acces direct la baza de date sunt de trei tipuri :

- **`ExecuteReader`**, care întoarce o colecţie de tupluri de date, accesibile apoi linie cu linie;
- **`ExecuteScalar`**, care returnează o singură valoare, de tip generic `object` ;
- **`ExecuteNonQuery`**, care execută actualizările asupra bazei de date şi returnează un `int`, indicând numărul de înregistrări afectate prin modificări, ştergeri sau inserări în baza de date.

Obiectele claselor de tip Reader (**`SqlDataReader`**, **`OleDbDataReader`**) pot fi generate de către obiectele de tip Command prin apelul **`ExecuteReader`** şi furnizează un flux de date, read-only, disponibilizând câte o înregistrare la fiecare apel al metodei **`Read()`**, spre deosebire de metoda `Fill()` a unui adaptor, care furnizează o colecţie de înregistrări:

```
OleDbDataReader dr = myCmd.ExecuteReader();
```

> `DataReader`-ul pointează aprioric pe BOF şi e nevoie de o citire prealabilă pentru a dispune de prima înregistrare.

Parcurgerea înregistrărilor cu DataReader este simplă şi se aseamănă cu citirea câte unei înregistrări dintr-un fişier. Localizarea unui câmp în înregistrarea din DataReader se face fie prin indexare, fie cunoscând numele coloanei din tabelă.

```
textBox1.Text = dr["pret"];    sau    textBox1.Text = dr[2];
```

O aplicaţie care utilizează DataReader presupune parcurgerea următorilor paşi:

- se crează o aplicaţie nouă, în care prin suprascrierea fişierelor `webForm1.aspx.cs`, `webForm1.aspx` se aduce codul sursă prezentat mai jos;
- se ataşează delegaţii pentru cele trei butoane, căci prin copiere nu s-au adus şi legăturile;
- se face o copie pentru fişierul **prod.mdb**, căci va fi alterat prin *ExecuteNonQuery;*
- se rulează şi comentează fiecare linie sursă, constatând efectele rulării.
- se poate completa aplicaţia cu un `GridView`, pentru vizualizarea simultană a tuturor tuplurilor selectate;
- pe un buton **`puneInGrid`** se adaugă codul de la funcţia **btnExecuteReader**, iar în loc de parcurgere cu `while` a **dataReader** -ului, se pune cod de legare a dataReader-ului ca sursă de

date pentru gridView (gridul va trebui să aibă ID-ul sursei pe nul, căci va folosi `DataSource` în loc de `DataSourceID`: `GridView1.DataSourceID = null;`)

```
//while (dr.Read())
//{
//    textBox1.Text+="\r\n"+dr["codp"]+ " " + dr["denum"]+ " " + dr["pret"];
//}
GridView1.DataSource = dr; GridView1.DataBind(); dr.Close(); con.Close();
```

Legarea dataReader-ului ca DataSource pentru un grid funcţionează numai pentru controlul `GridView` din **Web Forms**, nu şi pentru cel din Windows Forms.

Comparaţii între cele două modalităţi de folosire a obiectelor de tip xxxCommand:

Echipând un dataAdapter, obiectele de tip Command:	**Folosite individual, obiectele de tip Command:**
1. permit lucru vizual cu dataAdapter; 2. dataAdapter-ul închide/deschide implicit conexiunea; 3. acces mai lent la baza de date; 4. disponibilizează tot setul de înregistrări.	1. permit doar programare soft; 2. trebuie închisă /deschisă conexiunea explicit; 3. acces mai rapid la baza de date; 4. disponibilizează înregistrare cu înregistrare, prin intermediul unui obiect dataReader.

Observaţie. Se poate folosi metoda **ExecuteReader()** şi în locul celorlalte metode (*ExecuteNonQuery, ExecuteScalar*), căci metoda se va adapta după cerinţă.

În esenţă, obiectul de tip XxxCommand primeşte din construcţie cele două informaţii de care avea nevoie şi adaptorul: stringul SQL de executat şi conexiunea:

```
OleDbCommand myCmd      = new OleDbCommand(strSql,con);
```

Ce se întâmplă însă dacă părţi din comandă sunt variabile şi nu se cunosc la momentul construirii obiectului xxxCommand, ci se schimbă de la o execuţie la alta ?

Se poate rezolva această problemă în două moduri:

- prin **concatenare de string**-uri**, la momentul execuţiei**, înainte de crearea obiectului xxxCommand (căci comanda în sine este statică);
- prin încărcare de **parametri** şi folosirea lor la momentul execuţiei comenzii în baza de date.

Obiectele de tip XxxCommand au o colecţie de parametri, obiecte de tip `Parameter`, ce sunt caracterizate prin **nume**, **tip, direcţie** şi **valoare**; aceşti parametri circulă impreună cu comanda şi conţin valori în reprezentare internă; trebuie să coincidă cu tipul celor din baza de date, când sunt folosiţi la stocare valori în BD, dar pot fi şi de lat tip dacă sunt folosiţi în alt scop (spre exemplu, parametru `@PretMin` poate fi şi de tip *int*, deoarece el este folosit doar în comparaţii, chiar dacă preţul de referinţă este stocat în baza de date ca *double*.

```
private void pretPeste200_Click(object sender, System.EventArgs e)
{
      string strSql =
            "SELECT codp, denum, pret FROM produse where pret > @PretMin";
      OleDbConnection con =
            new OleDbConnection
            (@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\prod.mdb");
      try {  con.Open(); }
      catch(Exception ex) { mesaje.Text="Eroare open conexiune"+ex.Message;}
```

```
        OleDbCommand myCmd      = new OleDbCommand(strSql,con);
        OleDbParameter param1;
        param1 = myCmd.Parameters.Add
              (new OleDbParameter("@PretMin", OleDbType.Integer) );
        param1.Direction = ParameterDirection.Input;
        myCmd.Parameters["@PretMin"].Value=200;
        OleDbDataReader dr = myCmd.ExecuteReader();

        dg.DataSource = dr; dg.DataBind();
        dr.Close(); con.Close();
}
```

Observaţie. Comanda poate face astfel referiri mai sofisticate la acel parametru, identificat după nume; spre exemplu, putem să transferăm ca parametru de intrare un **BLOB** – Binary Large Object ce ar putea conţine o imagine !

Se observă că nu se folosesc obiecte DataAdapter sau DataSet, ci doar obiecte XxxCommand şi DataReader.

Pentru exemplificarea unei comenzi `ExecuteScalar()` am presupus că dorim să afişăm numărul de produse distincte, aflate în tabela produse. În acest scop am pus pe un buton din macheta aplicaţiei, următoarea funcţie :

```
private void btnExecScalar_Click
                        (object sender, System.EventArgs e)
{
 OleDbConnection conn = null;
 OleDbCommand myCommand = null;
 try
        {
         conn = new OleDbConnection(
         @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\prod.mdb");
         conn.Open();
         myCommand = new OleDbCommand
                        ( "SELECT COUNT(*) FROM produse", conn);
         int nrProd = (int)myCommand.ExecuteScalar();
         textBox1.Text="Numar produse comercializate: "+ nrProd;
        }
 catch(Exception excpt)
        {
             mesaje.Text=
                        "Esec pe ExecuteScalar: "+excpt.Message;
        }
}
```

Un apel `ExecuteNonQuery()` a fost exemplificat presupunând că se doreşte modificarea denumirii produselor care încep cu « p », prin adăugarea prefixului « txt ». Funcţia care realizează acest lucru tratează evenimentul `Click` al unui buton inscripţionat sugestiv:

```
private void btnExecNonQuery_Click
                        (object sender, System.EventArgs e)
{
        OleDbConnection conn = null;
        OleDbCommand command = null;

        string ConnString =
              "Provider=Microsoft.Jet.OLEDB.4.0;"+
                              "Data Source=C:\\prod.mdb";
        string cmd =
              "UPDATE produse SET denum = 'txt'&denum "+
                                    "where denum like 'p%'";
        try
```

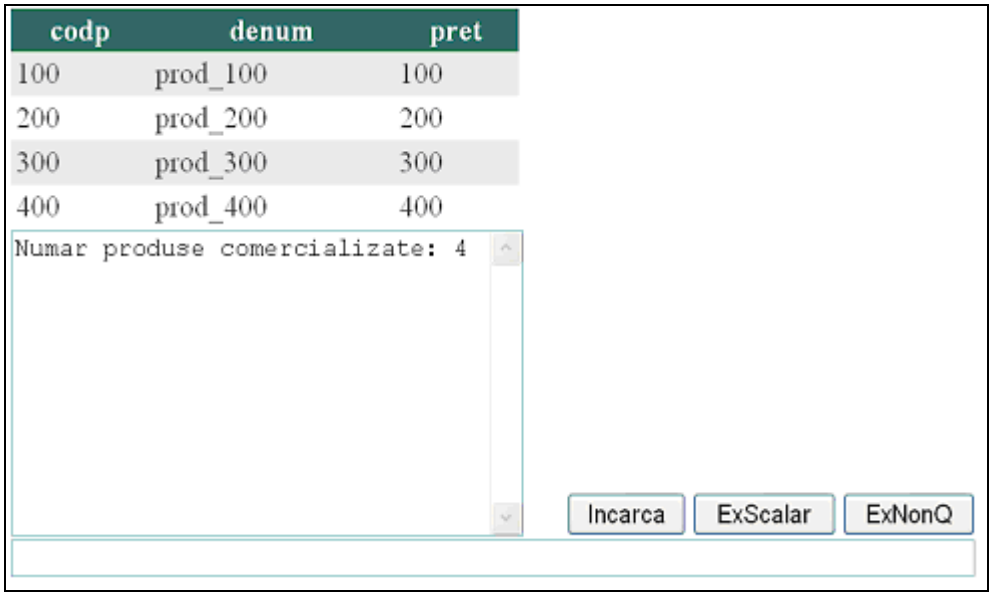
```
        {
         conn = new OleDbConnection(ConnString);  conn.Open();
         command = new OleDbCommand(cmd, conn);
         int nrRec = command.ExecuteNonQuery();
         textBox1.Text="Numar inregistrari afectate: "+ nrRec;
        }
        catch(Exception excpt)
        {
              mesaje.Text=
                 "Esec pe ExecuteNonQuery: "+ excpt.Message;
        }
        finally
        {
         if (conn.State == ConnectionState.Open) conn.Close();
        }
 }
```

Pentru rulare sub **Visual Studio**, cu bază de date sub **SQL Server**, codul sursă este următorul :

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.Sql;
using System.Data.SqlTypes;
using System.Data.SqlClient;

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

   }
   protected void btnIncarca_Click(object sender, EventArgs e)
   {
      string strSql = "SELECT codp, denum, pret FROM produse";
      SqlConnection con =
         new SqlConnection("Data Source=DOCTORAT2006\\SQLEXPRESS;Initial
Catalog=masterat;Integrated Security=True");
      textBox1.Text = "Lista produselor disponibile";
      try { con.Open(); }
      catch (Exception ex) { mesaje.Text = "Eroare open conexiune" + ex.Message; }

      SqlCommand myCmd = new SqlCommand(strSql, con);

      SqlDataReader dr = myCmd.ExecuteReader();

      //while (dr.Read())
      //{
      //    textBox1.Text += "\r\n" + dr["codp"] + " " +
          // dr["denum"] + " " + dr["pret"];
      //}

      GridView1.DataSource = dr; GridView1.DataBind();
      dr.Close(); con.Close();
   }
   protected void btnExecScalar_Click (object sender, System.EventArgs e)
   {
      SqlConnection conn = null;
```

```
        SqlCommand myCommand = null;
        try
        {
            conn = new SqlConnection(
            @"Data Source=DOCTORAT2006\SQLEXPRESS;Initial `Catalog=masterat;Integrated
Security=True");
            conn.Open();
            myCommand = new SqlCommand ("SELECT COUNT(*) FROM produse", conn);
            int nrProd = (int)myCommand.ExecuteScalar();
            textBox1.Text = "Numar produse comercializate: " + nrProd;
        }
        catch (Exception excpt)
        {
            mesaje.Text="Esec pe ExecuteScalar: " + excpt.Message;
        }
    }
    protected void btnExecNonQuery_Click (object sender, System.EventArgs e)
    {
        SqlConnection conn = null;
        SqlCommand command = null;

        string ConnString = "Data Source=DOCTORAT2006\\SQLEXPRESS;Initial Catalog=masterat;Integrated
Security=True";
        string cmd =     "UPDATE produse SET denum = 'txt'+denum " +     "where denum like 'p%'";
        try
        {
            conn = new SqlConnection(ConnString); conn.Open();
            command = new SqlCommand(cmd, conn);
            int nrRec = command.ExecuteNonQuery();
            textBox1.Text = "Numar inregistrari afectate: " + nrRec;
        }
        catch (Exception excpt)
        {
            mesaje.Text="Esec pe ExecuteNonQuery: " + excpt.Message;
        }
        finally
        {
            if (conn.State == ConnectionState.Open) conn.Close();
        }
    }


}
```

Pagina web cu rezultatele rulării arată astfel :



Lucru cu procedurile stocate

Procedurile stocate sunt cereri frecvent folosite asupra unei baze de date, grupate sub un nume de apel. Ele se concretizează sub forma unor porţiuni de cod, compilabile separat, existente pe server.

Avantajele folosirii procedurilor stocate**:**

- procedurile sunt deja compilate şi **deci nu mai conţin erori de compilare,** fiind direct lansabile în execuţie; fiind cunoscute de sever beneficiază şi de utilizarea **statisticilor interne** pentru optimizarea accesului la datele unei tabele;
- aflându-se pe server, procedurile stocate nu au nevoie să fie compuse şi retransmise prin reţea de fiecare dată când sunt invocate; astfel **accesul este mai rapid** deoarece se transmit doar parametrii de apel şi numele procedurii;
- au un **grad de securitate mai ridicat**, deoarece nu este vizibil codul sursă (a se vedea SQL injection ca element de insecuritate); controlăm mai bine accesul, dând dreptul de a executa proceduri stocate, dar nu şi instrucţiuni individuale.

Observaţii.

- A se vedea ce a pus în baza de date (**Queries / Design / View SQL code** pentru Microsoft Access, respectiv **Stored procedures** pentru SQL Server).
- Parametrii trebuie să aibă acelaşi tip cu ce s-a declarat la crearea procedurilor, iar dacă ei se asociază unor câmpuri din baza de date trebuie să corespundă ca tip cu aceştia sau să fie convertiţi la momentul folosirii (vezi `webBdProc1`, cu pret *`float`* comparat cu param *`int`*).
- Concordanţa la tipurile asociate cu *`DateTime`* este delicată existând multe formate de dată calendaristică, diferind ca lungime şi ca moment de referinţă ales.

Pentru început vom crea şi folosi o **procedură fără parametri**; pentru simplitate presupunem că avem funcţii distincte care crează, respectiv apelează procedura, la apăsarea câte unui buton inscripţionat corespunzător. Funcţia de creare încearcă mai întâi să şteargă o eventuală versiune mai veche a aceleeaşi proceduri, pentru a ne permite îmbunătăţirea progresivă a procedurii.

```
protected void btnCreareProc_Click(object sender, System.EventArgs e)
    {
        string strCreare = "CREATE PROCEDURE ProdScumpe AS  "
            + "Select * From produse where pret > 615 Order By denum";
        SqlConnection con =
```

```
            new SqlConnection(
            "Data Source=orion;Initial Catalog=doctorat;User ID=sa;Password=as;"
            );
        con.Open();
        SqlCommand myCmd = new SqlCommand();
        myCmd.CommandType = CommandType.Text; myCmd.Connection = con;
        myCmd.CommandText = "DROP PROCEDURE ProdScumpe";
        try { myCmd.ExecuteNonQuery(); }
        catch { TextBox1.Text = "Procedura de sters nu exista!!"; }

        myCmd.CommandText = strCreare;
        try
        {
            myCmd.ExecuteNonQuery();
            TextBox1.Text = "Procedura creata cu succes!!";
        }
        catch { TextBox1.Text = "Procedura nu a fost creata!!"; }
        con.Close();
    }
```

Se pot folosi diverse modalităţi de **a apela o procedură stocată**; cel mai simplu mod este ca într-un obiect de tip Command, să înlocuim fraza SELECT cu numele procedurii stocate:

```
protected void btnApelProc_Click(object sender, System.EventArgs e)
    {
        SqlConnection con =
            new SqlConnection(
            "Data Source=orion;Initial Catalog=doctorat;User ID=sa;Password=as;"
            );

        con.Open();

        SqlCommand myCmd = new SqlCommand("ProdScumpe", con);
        myCmd.CommandType = CommandType.StoredProcedure;
        SqlDataReader dr = myCmd.ExecuteReader();
        TextBox1.Text = "Lista produselor scumpe";
        while (dr.Read())
        {
            TextBox1.Text += "\r\n" + dr["codp"] + " " +
                dr["denum"] + " " + dr["pret"];
        }
        dr.Close(); con.Close();
    }
```

Dacă procedura ar lucra cu parametri de intrare, spre exemplu numai produsele cu preţul peste **o valoare, dată ca text într-un TextBox al formei**, procedura s-ar schimba, deoarece comanda trebuie să declare acum şi parametri de intrare / ieşire; în acest caz este nevoie să lucrăm şi cu obiecte de tip **`Parameter`**.

Aşa cum spuneam, obiectele Command dispun de o colecţie numită **`Parameters`**, în care prin metoda **`Add`**, adăugăm parametrii pregătiţi anterior sau furnizăm metodei informaţia necesară pentru a construi ea aceşti parametri:

```
param1 = cmdMea.Parameters.Add(new SqlParameter("@PretMin", SqlDbType.Int));
param1.Direction = ParameterDirection.Input;
cmdMea.Parameters["@PretMin"].Value = Convert.ToInt32(txtPret.Text);
```

Metoda **`Add`** a colecţiei `Parameters` are mai multe versiuni supraîncărcate, în funcţie de numărul de argumente folosite la construirea unui obiect de tip parametru. Se observă că, în afara faptului că metoda a construit şi a adăugat un parametru de tip `int` la o comandă, ea returnează şi referinţa parametrului, astfel încât acesta să poată fi ulterior echipat şi cu alte proprietăţi, spre exemplu direcţia:

```
param1.Direction = ParameterDirection.Input;
param2.Direction = ParameterDirection.Output;
```

Un parametru mai este accesibil şi prin metoda de **indexare** a colecţiei, care localizează un parametru după **poziţie** sau după **numele** său; construcţia de mai jos stochează drept valoare pentru parametru de intrare în procedură, preţul rezultat prin conversia în `int` a textului introdus de utilizator la rulare, în câmpul **`txtPret`**:

```
cmdMea.Parameters["@PretMin"].Value =
                Convert.ToInt32(txtPret.Text);
```

sau

```
cmdMea.Parameters[0].Value =
                Convert.ToInt32(txtPret.Text);
```

Reamintim că în ADO.NET parametrii trebuie **să se numească la fel** cu cei definiţi în procedura stocată şi să aibă **acelaşi tip** şi **aceeaşi lungime**.

Direcţia unui parametru poate fi:

- **`Input`** – parametru de intrare în procedură;
- **`Output`** - parametru de ieşire, returnat de procedură şi care nu poate conţine date de intrare pentru procedură;
- **`InputOutput`** - parametru folosit în acelaşi timp şi pentru intrare şi pentru ieşire, în comunicarea cu procedura;
- **`ReturnValue`** - parametru de ieşire, returnat de subprogramele de tip *function*.

Pot exista mai mulţi parametri de tip **`Input`**, **`Output`**, **`InputOutput`**, dar doar un singur parametru de tip **`ReturnValue`.**

Funcţiile de creare, respectiv folosire a unei **proceduri stocate cu un parametru de intrare**, arată acum astfel:

```
protected void btnCreareProc_Click(object sender, System.EventArgs e)
    {
        string strCreare = "CREATE PROCEDURE ProdPestePret(@pretMin INT) AS    " +
            "SELECT * FROM produse WHERE pret > @pretMin ";
        SqlConnection con =
            new SqlConnection(
            "Data Source=orion;Initial Catalog=doctorat;User ID=sa;Password=as;"
            );
        con.Open();

        SqlCommand cmdMea = new SqlCommand();
        cmdMea.CommandType = CommandType.Text; cmdMea.Connection = con;

        // daca exista, o sterge pentru a crea o noua versiune
        cmdMea.CommandText = "DROP PROCEDURE ProdPestePret";
        try { cmdMea.ExecuteNonQuery(); }
        catch (Exception excpt)
        {
            TextBox1.Text = "Stergere esuata: " + excpt.Message;
        }

        // refolosire comanda pentru a crea noua procedura
        cmdMea.CommandText = strCreare;
        cmdMea.CommandType = CommandType.Text;
```

```
        try
        {
            cmdMea.ExecuteNonQuery();
            TextBox1.Text = "Creare cu succes!!";
        }
        catch (Exception excpt)
        {
            TextBox1.Text = "Creare esuata: " + excpt.Message;
        }
        con.Close();
    }

    protected void btnTestareProc_Click(object sender, System.EventArgs e)
    {
        SqlConnection con =
            new SqlConnection(
            "Data Source=orion;Initial Catalog=doctorat;User ID=sa;Password=as;"
            );

        con.Open();

        SqlCommand cmdMea = new SqlCommand("ProdPestePret", con);
        cmdMea.CommandType = CommandType.StoredProcedure;

        SqlParameter param1;

        param1 = cmdMea.Parameters.Add(
                  new SqlParameter("@PretMin", SqlDbType.Int));
        param1.Direction = ParameterDirection.Input;

        cmdMea.Parameters["@PretMin"].Value = Convert.ToInt32(txtPret.Text);

        SqlDataReader dr = null;
        try
        {
            dr = cmdMea.ExecuteReader();
            TextBox1.Text = "Lista produselor peste "
                        + txtPret.Text + " EUR \r\n\r\n";
            while (dr.Read())
            {
                TextBox1.Text += "\r\n" + dr["codp"] + "   " +
                    dr["denum"] + "     " + dr["pret"] + " EUR";
            }
            dr.Close();
        }
        catch (Exception excpt)
        {
            TextBox1.Text = excpt.Message;
        }
        con.Close();
    }
```

De reţinut că **parametrii de ieşire sunt accesibili doar după închiderea DataReader**-ului.

Exerciţiu. Transformaţi procedura din aplicaţia de mai sus, astfel încât să returneze şi câte produse sunt în baza de date, cu preţul cuprins între două valori date.

Rezolvare. Se declară un alt treilea parametru, de data aceasta de tip `Output` şi se foloseşte ca în textul sursă de mai jos.

```
protected void btnCreareProc_Click(object sender, System.EventArgs e)
  {
```

```
        SqlConnection myCon =
            new SqlConnection(
            "Data Source=orion;Initial Catalog=doctorat;User ID=sa;Password=as;"
            );


        SqlCommand myCmd = new SqlCommand();
        myCmd.CommandType = CommandType.Text; myCmd.Connection = myCon;

        // daca procedura exista, o sterge pentru a crea o noua versiune
        myCon.Open();
        myCmd.CommandText = "DROP PROCEDURE ProdIntre2Preturi";

        try { myCmd.ExecuteNonQuery(); tbMesaje.Text = "Creare procedura Ok"; }

        catch (Exception excpt)
        {
            tbMesaje.Text = "Esec la stergere procedura: " + excpt.Message;
        }

        finally { myCon.Close(); }

        // refolosire comanda pentru inregistrare procedura
        string strCreare = "CREATE PROCEDURE ProdIntre2Preturi(@pretMin FLOAT , @pretMax FLOAT,
@CateProd INT OUTPUT )  AS " +
            "SELECT * FROM produse WHERE pret > @pretMin AND pret < @pretMax " +
            "SELECT  @CateProd = COUNT (*)FROM produse WHERE pret > @pretMin AND pret < @pretMax
";

        myCmd.CommandText = strCreare;
        myCmd.CommandType = CommandType.Text;
        myCon.Open();
        try { myCmd.ExecuteNonQuery(); tbMesaje.Text = "\r\nCreare procedura OK "; }
        catch (Exception excpt)
        {
            tbMesaje.Text = "Exceptie scriere procedura " + excpt.Message;
        }
        myCon.Close();
    }

    protected void btnApelProc_Click(object sender, System.EventArgs e)
    {
        SqlConnection MyCon =
            new SqlConnection(
            "Data Source=orion;Initial Catalog=doctorat;User ID=sa;Password=as;"
            );

        if (MyCon.State == ConnectionState.Closed) MyCon.Open();
        SqlCommand myCmd = new SqlCommand("ProdIntre2Preturi", MyCon);

        myCmd.CommandType = CommandType.StoredProcedure;

        SqlParameter param1, param2, param3;

        param1 = myCmd.Parameters.Add(
                            new SqlParameter("@PretMin", SqlDbType.Float));
        param1.Direction = ParameterDirection.Input;
        myCmd.Parameters["@PretMin"].Value = double.Parse(tbMin.Text);

        param2 = myCmd.Parameters.Add(
                            new SqlParameter("@PretMax", SqlDbType.Float));
        param2.Direction = ParameterDirection.Input;
        myCmd.Parameters["@PretMax"].Value = double.Parse(tbMax.Text); ;
```

```
    param3 = new SqlParameter("@CateProd", SqlDbType.Int);
    param3.Direction = ParameterDirection.Output;
    myCmd.Parameters.Add(param3);
    SqlDataReader dr = null;

    try
    {
       dr = myCmd.ExecuteReader();
       while (dr.Read())
       {
          tbMesaje.Text += "\r\n" + dr["codp"] + " " +
             dr["denum"] + " costa  " + dr["pret"];
       }
    }
    catch (Exception excpt)
    {
       tbMesaje.Text = "Exceptie la executie procedura" + excpt.Message;
    }
    finally
    {
       if (dr != null) dr.Close();
       MyCon.Close();
    }

    tbCateProd.Text = myCmd.Parameters["@CateProd"].Value.ToString();

}
```

Gestiunea stării în aplicaţiile web

Conexiunile prin Internet sunt **"stateless connections",** adică după conectarea unui client, serverul furnizează informaţia cerută după care "uită" totul despre acel utilizator. În situaţia în care se completează nişte valori într-un formular şi se solicită serverului să le prelucreze şi să ne dea un răspuns, pe care să-l vedem **împreună cu valorile introduse de noi,** este nevoie de un mecanism prin care o parte din informaţiile din pagină să fie restaurate în forma la care se aflau când pagina a plecat spre server. Problema restaurării paginii apare deci, ori de câte ori se lucrează mai mult timp cu aceeaşi pagină, care efectuează un "du-te – vino" spre server, de câte ori este nevoie.

Starea se defineşte ca mulţime a valorilor conţinute în **controalele şi variabilele** paginilor, a informaţiilor corespunzătoare utilizatorilor sau aplicaţiei în ansamblu.

"**stateless connections**" presupune însă că la fiecare reîncărcare, **pagina să se reconstruiască de la zero**, folosind valorile **implicite** ale controalelor şi variabilelor, lucru care nu e de dorit dacă vrem să vedem pagina cu ultimile valori introduse de noi.

ASP.NET permite gestiunea acestei stări atât **de către mediu** ASP, cât şi **de către programator**, divizând informaţia în trei categorii distincte, în funcţie de nivelul de referinţă - **pagină, utilizator, aplicaţie** - şi implicit, de containerul în care se ţin:

- **`View state`** – având drept container un obiect de clasă `StateBag`, instanţiat automat de sistem;
- **`Session state –`** container fiind obiectul `Session`;
- **`Application state –`** pentru care containerul este obiectul `Application`.

View state – gestiunea stării la nivel de pagină

View State reprezintă **starea paginii** şi controalelor sale; ea **se ţine automat** de către ASP.NET Framework pentru paginile şi controalele care au setată proprietatea **`EnableViewState`** pe `true`. La transmiterea paginii către server, informaţia de stare este încărcată în ***state bag***, nişte câmpuri ascunse în pagină, care codifică această stare şi circulă odată cu pagina, această tehnică fiind **recunoscută de orice browser**. La reîncărcarea paginii din server (adică la **postback**), informaţia de stare este rescrisă în controale, utilizatorul văzând pagina completată şi cu valorile pe care le-a introdus anterior.

Pentru creşterea performanţei aplicaţiei, programatorul poate pune proprietatea `EnableViewState` pe `false` **pentru întreaga pagină** sau **pentru controalele din pagină** a căror stare nu interesează. Spre exemplu, dacă ne putem permite la fiecare încărcare de pagină să citim din nou, direct din baza de date, informaţiile prezentate într-un GridView (**`dg`**), putem declara (sau pune vizual în fereastra **Properties**):

```
dg.EnableViewState = false;
```

Dacă informaţia de stare a întregii **pagini** nu necesită salvare şi restaurare la fiecare transmitere către / dinspre server, putem solicita acest lucru chiar prin directiva de pagină:

```
<%@ Page Language="C#"  EnableViewState="false" %>
```

Dacă dorim inhibarea salvării **`ViewState`** la nivelul întregii **aplicaţii** (pentru toate paginile acesteia), punem proprietatea `EnableViewState` pe `false` în secţiunea **`<pages>`** din fişierele de configurare *machine.config* sau *web.config*.

Cum informaţia din **`ViewState`** circulă de fiecare dată cu pagina către şi dinspre server, nu orice informaţie trebuie salvată la acest nivel. În general, **nu se stochează** la nivel de **`ViewState`** **informaţia de volum mare,** sau informaţia care trebuie să aibă un **grad înalt de securitate**.

În plus, doar pentru informaţia de tip `string`, `int`, `boolean`, `Arrays`, `ArrayLists` şi `HashTable` a fost optimizat mecanismul de transfer; pentru celelalte tipuri acest transfer poate deveni foarte neperformant.

Informaţiile din **variabile** nu sunt puse automat în `ViewState`, programatorul trebuind să scrie cod pentru adăugarea lor explicită în `state bag`; acest container este un obiect de clasă **`StateBag`** şi poate fi gândit ca un dicţionar ce stochează asociaţii de tip **atribut - valoare**. Atributul este un `string` ce permite regăsirea valorii, iar valoarea este interpretată ca un obiect generic, putând fi de toate tipurile.

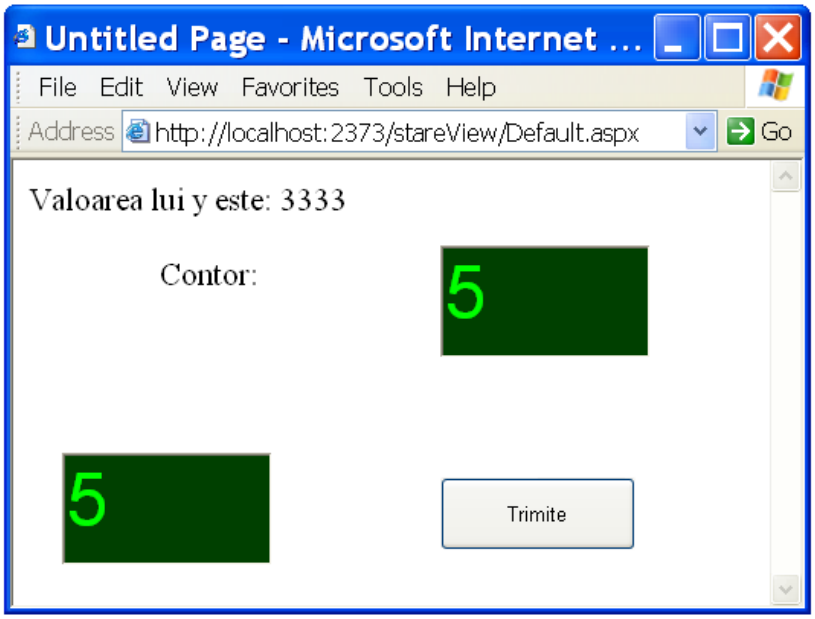
Un astfel de dicţionar deţine atât funcţii pentru inserare de perechi **atribut - valoare**, cât şi funcţii pentru extragere sau modificare.

Exemplu **`stareView`**. La nivel de pagină, să se ţină evidenţa numărului de reîncărcări ale paginii, pe întreaga durată a sesiunii. Două controale de tip `TextBox` vor transporta către server numărul accesului: unul automat (**`tbNrAcces`**) cu proprietatea **`EnableViewState`** pe `true` și unul controlat prin program (**`tbNrAcces1`**), deoarece are proprietatea **`EnableViewState`** pe `false`.

În clasa principală a aplicaţiei se declară variabila **`int contor`**; ea este iniţializată în funcţia de tratare `OnInit()`, moment în care este stocată și ca text, pentru afișare în **`tbNrAcc1`**:

```
override protected void OnInit(EventArgs e)
    {
            contor=0; ViewState.Add("kContor", contor);
            tbNrAcc1.Text="0";
            base.OnInit(e);
    }
```

ocazie cu care este şi stocată în **`ViewState`** cu cheia de identificare **`kContor`**.



Pe textBox-ul **`tbNrAcc1`** adăugat vizual pe formă şi etichetat corepunzător, cu proprietatea **`EnableViewState`** pe `false`, la fiecare **încărcare de pagină**, extragem din `ViewState` valoarea contorului, o incrementăm şi afişăm noua valoare, ca text.

Pentru controlul **`tbNrAcc`** preluăm textul care circulă implicit prin mecanismul `ViewState`, îl convertim în număr pentru a-l putea incrementa și-l repunem ca text, în control.

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (IsPostBack)
        {
            contor = (int)ViewState["kContor"]; contor++;
            tbNrAcc1.Text = contor.ToString();
```

```
            int aux = Convert.ToInt32(tbNrAcc.Text);
            aux++; tbNrAcc.Text = aux.ToString();
        }
        Response.Write("Valoarea lui y este: " + (string)Application["y"]);
    }
```

De remarcat cast-ul spre `int` aplicat obiectului returnat de indexarea elementului din `ViewState`. Ori de câte ori apăsăm butonul `btnTrimite`, noua valoare este restocată în `ViewState`, pentru a călători împreună cu pagina:

```
private void btnTrimite_Click(object sender, System.EventArgs e)
{
      ViewState["kContor"] = contor;
}
```

De menţionat că textBox-ul `tbNrAcc1` are proprietatea `EnableViewState` pe `false`, deşi numărul de accesări, fiind vorba de o informaţie mică, ar fi putut fi lăsat să circule implicit cu controlul, ca în cazul lui `tbNrAcc`, nu neapărat pusă explicit ca variabilă, în **`ViewState`**.
Ca format, numărul de accesări circulă ca `string` într-un caz și ca `int` în celălalt.

Gestiunea stării la nivel de sesiune

În afara obiectului **state bag,** instanţiat şi gestionat automat de sistem, mai există încă două obiecte care ţin informaţia de stare:

- **Session** - care ţine informaţia specifică fiecărui utilizator în parte, pe durata unei sesiuni de lucru, când navighează între două pagini diferite.
- **Application** - care ţine informaţia comună mai multor utilizatori ai aceleeaşi aplicaţii Web.

Alături de acestea, în programarea aplicaţiilor web, ne vom întâlni frecvent cu încă două obiecte:

- `Request` - obiect pentru acces la cererea în curs; el circulă către server;
- `Response` - obiect pentru expedierea unui răspuns (`HttpResponse`) înapoi către client.

O sesiune se defineşte printr-o secvenţă de cereri lansate de la acelaşi browser client şi derulate astfel încât **intervalul de timp dintre două cereri să nu depăşească intervalul de timeout** definit în fişierul de configurare **web.config** (implicit 20 min); depăşirea acestui interval încheie automat sesiunea curentă de lucru.

Managementul **stării sesiunii** rezolvă problema păstrării informaţiei, când navigând într-o aplicaţie web accesăm pagini diferite. Sesiunea poate fi aşadar comparată cu o vizită la un site, cu **răsfoirea mai multor pagini**.
Exemple în care apare necesară păstrarea stării la nivel de sesiune:

- **Coşul de cumpărături** – o listă a tuturor produselor reţinute pe măsura vizitării unor pagini de prezentare a produselor dintr-un magazin virtual.
- **Datele despre vizitatorul unui site**, în scopul personalizării unui acces viitor (*customer profiling*), se ţin într-o bază de date. Pentru colectarea şi actualizarea lor periodică, pe măsura răsfoirii paginilor se acumulează informaţii în **Session** (de unde a venit, ce pagini a vizitat, cât timp a stat într-o pagină, ce link-uri a ales la plecare etc.).

Așadar, pagina și utilizatorul se asociază cu informația de sesiune.

Aplicaţie `stareSesiune2`.

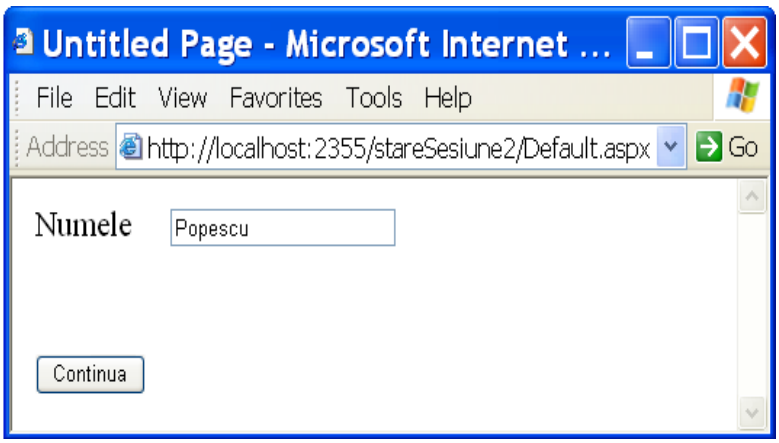
Probabil cel mai simplu exemplu ar fi să construim o aplicaţie web cu două pagini; prima pagină citeşte într-un textBox numele unei persoane, pe care-l va memora apoi în obiectul de stare al sesiunii (la încărcarea paginii în server), sub indexul `"utilizator"` :

```
private void Page_Load(object sender, System.EventArgs e)
{
```

```
        Session["utilizator"] = tbNume.Text;
}
```

Adăugăm încă o pagină web aplicaţiei noastre: **Website / Add New Item / WebForm,** pe care o vom denumi **`pag2.aspx`**.Pe butonul `Continua` din prima pagină declanşăm trecerea în pagina a doua:

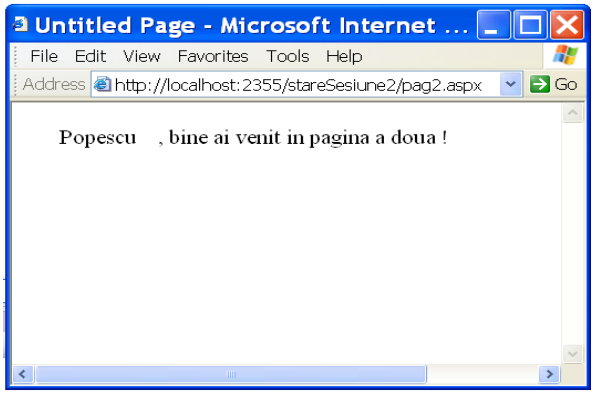
```
private void btnContinua_Click(object sender, System.EventArgs e)
{
  Response.Redirect("pag2.aspx");
}
```



iar la încărcarea paginii următoare vom recupera şi afişa informaţia din obiectul Session (când se declanşează `Page_Load` şi din al cui punct de vedere, server sau client, este văzută ca încărcare ?) : Server!!

Punctul de vedere este cel al serverului, iar succesiunea evenimentelor este: PageInit, PageLoad, **`RequiredFieldValidator`**, **`btn`**Click, TextChanged, **insert**, **delete** şi **update, select,** Page.Unload

```
private void Page_Load(object sender, System.EventArgs e)
 {          // in pagina a doua
      lblSalut.Text = (string)Session["utilizator"]+
                              ", bine ai venit in pagina 2 !" ;
 }
```



Gestiunea stării la nivel de aplicaţie

O aplicaţie Web se defineşte ca fiind mulţimea **paginilor Web**, a **fişierelor de date** şi **cod**, a altor tipuri de **resurse disponibile în directorul virtual** sau în subdirectoarele acestuia. Aşadar, aplicaţia Web se identifică cu directorul virtual, mai precis cu fişierul *global.asax* din acest director; acesta conţine de altfel **codul partajabil între mai mulţi utilizatori** simultani ai aplicaţiei Web.

Fişierul *global.asax* conţine şi codul sursă pentru tratarea evenimentelor ce apar la nivelul aplicaţiei.

Există **un singur fişier** *global.asax* ce utilizează un singur limbaj de programare pentru a preciza codul executabil de tratare a evenimentelor ce apar la nivelul aplicaţiei Web.

La instanţierea unei pagini Web se instanţiează un obiect din clasa `HttpApplication`; el are metode, proprietăţi şi evenimente specifice şi dispune de obiectele ce-i permit să intercepteze cereri HTTP. Principalele referinţe de obiecte pe care `HttpApplication` le deţine sunt către obiectele:

- `Application`, ce memorează informaţia privind starea aplicaţiei;
- `Request`, ce conţine cererea unui client către server;
- `Response`, ce conţine `HttpResponse` dat ca răspuns clientului, la o cerere;
- `Session`, ce memorează informaţia privind starea sesiunii de lucru.

Uneori este nevoie de păstrarea unor informaţii despre toate sesiunile de lucru pe un site, informaţii accesibile în permanenţă **tuturor utilizatorilor**, **din oricare pagină** a site-ului. Ele sunt frecvent folosite de aplicaţie şi se ţin **în memorie, pe server**, cât timp aplicaţia este accesată de vreun utilizator; reiese că orice nouă informaţie adăugată în obiectul `Application`, consumă un plus de memorie. O parte din aceste informaţii se referă la sesiunile de lucru cu o aplicaţie (durata lor) şi sunt legate de evenimente precum începerea sau terminarea unei sesiuni de lucru, lansarea unei cereri etc.

Pentru că mai mulţi utilizatori au acces simultan la acelaşi obiect `Application`, pentru prevenirea blocajelor în lanţ, se impune ca toate operaţiile pe acest obiect să se facă după blocarea lui de către cel care doreşte să opereze pe el:

```
Application.Lock();
{
      // modificări pe obiectul Application
}
Application.UnLock();
```

Aplicaţia `JurnalApp`

Cu **WebSite – Add / New Item / Global Application Class** (care ulterior nu mai apare ca optiune, dacă site-ul are deja fişierul **global.asax**) se crează automat fişierul conţinând **funcţiile de tratare a evenimenteleor apărute la nivel de aplicaţie şi la nivel de sesiune de lucru.**

Din **Solution Explorer / Global.asax** vedem unele dintre evenimentele generale ale aplicaţiei, deja tratate prin funcţii vide. Le vom modifica pentru a semnala în pagina Web, prin mesaje, succesiunea declanşării principalelor evenimente la nivelul aplicaţiei.

Deoarece o parte dintre evenimentele aplicaţiei au ca efect **schimbarea paginii**, **întreruperea unei sesiuni** de lucru etc., pentru a le consemna nu putem totdeauna să afişăm mesaje corespunzătoare în pagină, astfel încât preferăm înscrierea lor într-un jurnal, pe disc. Pentru aceasta construim o funcţie de jurnalizare, care scrie textul mesajului primit, împreună cu ora la care l-a primit:

```
void ScrieJurnal(string mesaj)
{
  System.IO.StreamWriter fisout = // append daca exista
      new System.IO.StreamWriter("C:\\jurnal.txt",true);
  string rand = "\n"+DateTime.Now.ToString(  ) + "   " + mesaj;
  fisout.WriteLine(rand);           fisout.Close(  )
}
```

Dacă sistemul de operare nu permite accesul la fişierul `"C:\\jurnal.txt"` pentru a nu da drepturi pe tot directorul `C:\` se crează mai bine cu notepad-ul fişierul `"C:\\jurnal.txt"` şi se dau drepturi de acces utilizatorului ASP.NET, aşa cum s-a precizat în capitolul despre accesul la baze de date.

Aplicaţia **JurnalApp** foloseşte această funcţie pentru a marca prin mesaje, succesiunea diferitelor evenimente apărute la nivelul aplicaţiei.

```
protected void Application_Start(Object sender, EventArgs e)
```

```
{
      Application.Add("nrSes",0);Application.Add("nrCer",0);
      ScrieJurnal("Aplicatia a fost lansata.");
}

protected void Session_Start(Object sender, EventArgs e)
{
      Application.Lock();
      int nrSesiuni=(int)Application.Contents["nrSes"]+1;
      Application.Set("nrSes",nrSesiuni);
      Response.Write("Sesiunea "+ nrSesiuni+" a inceput la "
            +DateTime.Now.ToLongTimeString()+"<br>");
      Response.Write("Aplicatia are  "+
            Application.AllKeys.Length+" variabile globale <br>");
      Response.Write("  Aplicatia detine perechile: ");
      foreach( string cheie in Application.Contents.AllKeys)
            Response.Write(" <br> "+ cheie +" = "
                  +Application.Contents[cheie].ToString());
      ScrieJurnal("Sesiunea nr. "+Application["nrSes"]+" " +
                  Session.SessionID+ " a fost deschisa");
      Application.UnLock();

}

protected void Application_BeginRequest(Object sender, EventArgs e)
{
      Application.Lock();
      Response.Write("Lansat ev. BeginRequest <br><br>");
      int nrCereri=(int)Application.Contents["nrCer"]+1;
      Application.Set("nrCer",nrCereri);
      ScrieJurnal("A fost primita cererea cu nr. " +nrCereri);
      Application.UnLock();
}


protected void Session_End(Object sender, EventArgs e)
{
      ScrieJurnal("Sfarsitul sesiunii nr. "
            +Application["nrSes"]+" " +Session.SessionID);
      int nrSesiuni=(int)Application["nrSes"]-1;
      Application.Set("nrSes",nrSesiuni);
}

protected void Application_End(Object sender, EventArgs e)
{
      ScrieJurnal("Aplicatia stop");
}
```

Se lansează în paralel o nouă sesiune pe acelaşi server de dezvoltare, observând atent portul prin care comunică cu site-ul şi tastând sub IE în bara de adresă, spre exemplu, **http://localhost:2521/JurnalApp/**). Se cere din când în când, **View / Refresh** din browser şi observăm înregistrarea cererii către server, adică evenimentul `BeginRequest`.

Se poate lucra şi cu serverul de web IIS; în acest sens directorul fizic în care a fost creat site-ul este dus în Inetpub\wwwroot şi este apoi făcut director virtual (sub IIS, **Default website** / director fizic / **Properties** şi se alege **Create??**); se pot apoi deschide câte sesiuni dorim, direct din instanţe diferite ale Internet Explorer-ului.

Consultând apoi fişierul jurnal `C:\\jurnal.txt`, constatăm că la primul apel de pagină web se generează un grup de obiecte, de mai multe tipuri.

La crearea obiectului **`Application`** se declanşează evenimentul `Application_Start`; la crearea câte unui obiect **`Session`** se declanşează şi evenimentul `Session_Start`. Pe *timeout* (depăşirea numărului de minute de inactivitate) se generează automat evenimentul `Session_End`

pentru fiecare obiect `Session`. Când ultimul obiect `Session` se distruge, se declanşează pe lângă `Session_End` şi evenimentul `Application_End`. Trebuie să avem în **web.config** declarat:

```
<sessionState
            mode="InProc"
            timeout="1"
    />
```

Obiectul `Response` nu există când se lansează aplicaţia, astfel încât functia `Application_Start` nu poate afişa informaţii în pagina web, căci pagina primeşte informaţiile prin intermediul obiectului `Response`.

Observăm că e uşor de intuit momentul **începerii** aplicaţiei, respectiv sesiunii (când cerem vizualizarea paginii), dar dificil de întrevăzut momentul **terminării** sesiunii, respectiv aplicaţiei. Aceasta deoarece serverul nu ştie dacă un client deja conectat va mai reveni sau a încheiat deja lucru cu această pagină.

Putem specifica în fişierul de configuraţie `web.config` parametrul `timeout = 1`, în loc de `timeout = 20`, cât era pus automat, astfel încât după un minut de inactivitate sesiunea de lucru să se considere încheiată; putem să marcăm astfel în timp scurt, trecerea prin toate evenimentele declanşate la vizitarea site-ului.

În ce priveşte aplicaţia, probabil cel mai bine este să **forţăm terminarea ei** făcând o modificare în fişierul `global.asax`, caz în care ASP.NET sesizează că s-a schimbat aplicaţia, încheie aplicaţia curentă şi o restartează în noul format.

`global.asax` şi **`paginile aspx`** lucrează pe acelaşi namespace, astfel încât putem folosi în **`webform1.aspx`** variabile sau funcţii definite în **`global.asax`**.

Un alt mod de a menţine valori pe toată ***durata aplicaţiei şi cu vizibilitate la nivelul întregii aplicaţii*** îl reprezintă variabilele de tip *`static`*. Ele se declară în `Global.asax` (cod script necompilat, e tot C# !!) sau `Global.asax.cs` (cod precompilat) şi devin utilizabile în toate paginile aplicaţiei.
Într-un script din `Global.asax` o declaraţie ar putea arăta astfel:

```
<%@ Application Classname="myClass" %>
<script language="c#" runat=server>
public static int cntApplication = 0;
public void Session_Start()
{
      cntApplication++;
}
</script>
```

adică se dă un nume de clasă obiectului ce va conţine variabila, se declară şi iniţializează variabila, ulterior ea putând fi folosită într-un eveniment, chiar la nivel de sesiune.

Pentru lucru cu variabile globale definite în codul ce va fi compilat în prealabil, adică în fişierul `Global.asax.cs` procedeul este tot cel standard, impus de limbajul C#. Spre exemplu, în **`global.asax.cs`**, clasa definită de mediu este numită implicit `Global` şi ca orice clasă, poate conţine definiţii de variabile sau funcţii:

```
public static int x;
```

În WebForms putem adresa aceste variabile sau funcţii, folosind rezoluţia de clasă sau obiectul, sub forma:

```
Global.x = 3;
```

Revenind la fişierul `global.asax`, putem preciza că mai există un alt mod de a declara variabile statice şi anume direct în fişier, în afara blocului de script, caz în care cele două mecanisme *application state* şi *static variables* se confundă. În acest scop, deschidem cu notepad fişierul `global.asax` şi adaugăm în afara blocului de script, secvenţa:

```
<object id="y" class="System.String" scope="Application" runat="server"/>
```

Astfel se adaugă variabila `y`, de tip `string`, la nivel de `Application` (putea fi şi la nivel de `Session`). Variabila nu este recunoscută înainte de această declaraţie şi durează atâta timp cât durează şi aplicaţia (sau sesiunea, dacă scopul este `Session`). Variabila **poate fi calificată** apoi ca făcând parte din dicţionarul aplicaţiei şi poate intra în evaluarea unor expresii.

```
<% Application["y"]="3333"; %>
```

pus în `webform1.aspx` face o iniţializare, iar

```
Response.Write("Valoarea lui y este: "+ (string)Application["y"]);
```

pus în `webform1.aspx.cs` afişează în pagină conţinutul lui `y`.
În blocul `<script runat="server"> </script>` se pot pune handlere, metode sau variabile statice. Aşadar, ca orice cod, declaraţiile de variabile pot fi puse în fişierul citat în `code-behind` sau în blocul script.

Implicaţiile gestiunii stării în lucru cu baze de date

Aplicaţia `stareGrid`

Considerăm o aplicaţie simplă cu un grid, care la apăsarea unui buton `IncarcaGrid` afişează datele dintr-o tabelă a bazei de date; pentru aceasta, construim un proiect **Web Application**, dragăm un obiect **dataAdapter** şi cu mouse dreapta îi cerem **Generate DataSet**; aducem apoi un obiect **dataGrid**, iar pe un buton solicităm umplerea dataSet-ului şi legarea gridului la dataSet pentru a vizualiza informaţiile:

```
private void incarcaGrid_Click(object sender, System.EventArgs e)
{
	try
	{
		oleDbDataAdapter1.Fill(dataSet1);
		DataGrid1.DataSource =this.dataSet1;
		DataGrid1.DataBind();
	}
	catch(Exception exc)
	{		mesaje.Text=exc.Message;	}

}
```

Dacă mai adăugăm un buton `CerePostBack`, care nu face nimic, dar la apăsare forţează un PostBack (adică solicită vizitarea serverului pentru recompunerea paginii curente, cu eventuale modificări dinamice indicate prin prelucrările din funcţia de tratare a evenimentului click pe acest buton), observăm că gridul va continua să afişeze bine datele. Secretul constă în faptul că deşi nu s-a mai apăsat pe butonul `IncarcaGrid` datele se văd căci ele circulă cu controlul **`DataGrid`**, care are setată implicit proprietatea **`EnableViewState`** pe **`true`**. Dacă setăm această proprietate pe **`false`** observăm că datele nu mai apar în grid după apăsarea butonului `CerePostBack`.

O problemă poate fi totuşi formulată în legătură cu acest lucru: se justifică această călătorie a unui volum, uneori foarte mare, de date către server şi înapoi? Dacă nu se justifică, cum am putea totuşi vedea şi beneficia de datele din grid ?

Există două alternative:

1. punem instrucţiunile de încărcare grid direct în **`Page_Load`**, astfel încât datele vor fi preluate din baza de date la fiecare încărcare de pagină (măcar nu mai circulă şi către server, ci doar înapoi către client);
2. **caching**, caz în care datele rezidă pe server, în memoria procesului ce execută cererea sau în cea globală şi nu mai trebuie citite din baza de date, lucru care ar consuma mai mult timp.

Mecanismul de *cache* extinde întrucâtva facilităţile introduse prin `application state`; acesta nu numai că ţine informaţia la nivelul întregii aplicaţii, ci dă, spre exemplu, posibilitatea sesizării momentului în care un obiect din cache a fost modificat şi deci pagina necesită un *refresh*, pentru a beneficia de noua versiune (a se vedea metodele clasei **`Cache`**).