

Active Server Page - ASP.NET

Concepțe de bază

Dinamicitatea paginilor web

- **paginile statice** se mai numesc și **pagini HTML**, căci nu conțin alt tip de cod decât cel scris în HTML; **conținutul unei pagini nu variază în timp sau de la un utilizator la altul**; toți văd la fel, fără informații specifice contextului rulării; **conținutul este fixat și 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 afișarea orei exacte, sau pentru adaptarea conținutului 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 execuției**, în funcție de context și de resursele disponibile pe calculatorul de pe care se lansează cererea; de obicei aceste pagini au și un nivel înalt de interactivitate. Pentru a realiza dinamicitatea, trebuie scris cod sursă care se rulează la momentul lansării unei cereri și care **prin rulare stabilăște conținutul ce va fi afișat** în pagină; trebuie avut însă în vedere în ce limbaj este scris acest cod, cine înțelege și poate prelucra acest cod.

Dinamicitatea client-side: codul sursă HTML este mixat cu **instrucțiuni 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 capacitatele 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 știm ce se află instalat pe mașini și cunoaștem gradul de securitate ce trebuie asigurat.

Dinamicitatea server-side: codul sursă HTML este mixat cu **instrucțiuni de compunere a paginii**, scrise **în diverse limbiage de programare și 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 distanță, de către un client. Pagina circulă tot ca stream HTML; mai precis, în modul de prelucrare server, către client în rețea **circulă doar rezultatul prelucrării** paginii, nu și instrucțiunile 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 performanța accesului. Este mult mai puternic, pe server putându-se instala în siguranță, toate tipurile de software de creație pagini web.

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

Web server – este un software care:

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

Tehnologia CGI – Common Gateway Interface

Avantaje:

- acces la resursele sistemului de operare;
- viteza bună de execuție, dacă aplicația 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 aplicațiile accesibile din Internet este denumit pe scurt **ASP.NET (Active Server Page)** și are următoarele caracteristici principale:

- se bazează pe formulare Web (Web Forms) și 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 și tratate în orice browser;
- asigură accesul la date prin mecanismul **ADO.NET**;
- permite **caching** pentru date și 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 aplicații; 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* și *Passport authentication*;
- tratează fișiere XML, inclusiv pentru reconfigurări, **fără a necesita restartarea server-ului**;
- oferă extensibilitate prin crearea și 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 conținând lista site-urilor din evidența 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 și interdicții pe diverse servicii ale IIS.
- Adăugare director virtual:
Default Web Site / buton dreapta mouse / New Virtual Directory; aplicațiile create cu Visual Studio beneficiază de **propriul director virtual**, creat odată cu aplicația; cele create cu Visual Studio pot rula și 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 aplicației.

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 rețea 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 aceeași conexiune. O adresă IP poate avea și pseudonim:

- în registrul DNS, pe serverul de rețea;
- 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** și se indică adresa de IP înregistrată mai sus, plus drepturile de acces.
- La încercarea de conectare de la distanță:
 - fie se afișează directoarele din site (opțiunea **Directory Browsing** activă);
 - fie se predă controlul unui document implicit (opțiunea **Default Document** activată).

Cele două opțiuni 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 informațiilor într-o pagină web

O pagină Web poate conține:

- cod **html**, recunoscut de orice browser;
- cod într-un **limbaj de programare** recunoscut de către serverul care ține pagina Web;
- cod **script** într-un limbaj de tip script, recunoscut de majoritatea browserelor (JavaScript, VB Script fiind cele mai răspândite limbi de tip script).

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

- **secțiunea de precizare a codului**

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

- **secțiunea 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 și gruparea întregului cod în fișier separat, tehnică denumită "code behind"; la începutul paginii se indică numele fișierului conținând 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 secvența de mai jos, într-un fișier **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.WriteLine( System.DateTime.Now.ToString() ); %>  
</html>
```

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

Dacă nu dispunem de serverul de web IIS instalat, putem starta serverul de web de sub Visual Studio rulând o aplicație oarecare; cât timp serverul de dezvoltare e activ, modificăm în linia de adresă din browser doar numele fișierului ce conține pagina (spre ex. înlocuim **default.aspx**, cu **DataOra.aspx**, cu condiția ca **fișierul să fi fost plasat în același director cu aplicația care a startat serverul**, deoarece serverul vede directorul aplicației ca pe un director virtual).

Observație.

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

Pentru fișierul inițial, ce conține codul de prelucrare, extensia fișierului trebuie să fie **aspx**, pentru a anunța serverul cui s-o paseze pentru prelucrare, deoarece acest tip de fișier conține nu forma prelucrată a paginii, ci instrucțiunile de prelucrare !

Exemplu **DataOraJS.html** Dinamicitate client-side

Cu un editor de text se introduce secvența de mai jos, într-un fișier 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 același 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 și îl execută, apoi afișează rezultatul. **Ora afișată 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 afișa 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 funcții, iar în interiorul paginii se pune codul de vizualizare, care apelează această funcție și afișează părțile rezultate prin rularea funcției.

```

<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 funcția (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 părțile deja calculate, browser-ul doar afișându-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 referință la un obiect HTML de tip `Response`, ce conține răspunsul dat de server la solicitarea unei pagini de către un client. Obiectul `Response` recunoaște metodele `Response.Write()` și `Response.Output.Write()` cu care putem adăuga linii în răspunsul returnat clientului; `Response.Output.Write()` permite chiar și scrierea de **text formatat**, pe care un browser știe să-l interpreze.

O altă variantă ar fi să punem funcția într-un fișier separat, pe care să-l cităm în clauza `src="fis.cs"`, lăsând în pagină **doar partea de apel și 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 și 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#"` și `</script>` **se include în această clasă**, iar ceea ce se specifică între `<% și %>` **poate fi imaginat ca o parte dintr-o funcție `Render()`** a clasei **derivată** din clasa `Page`.

Reiese că între `<script>` și `</script>` putem include funcții, pe când între `<% și %>` nu dăm decât elemente specifice redării paginii Web, neputând defini alte funcții deoarece ne aflăm deja în interiorul unei funcții (adică în funcția `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+="  
" + 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 fișierul `C:\prod.mdb` conținând o bază de date Access, cu tabela produse și câmpurile: `codp`, `denum` și `pret`.

Pregătirea paginii se face tot pe server; o problemă care apare aici este cum va călători informația 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 și de funcția `Render()`, dar își pierde conținutul între două cereri succesive. Putem apela la un control de tip `TextBox`, care va transporta informația în proprietatea sa numită `Text`, folosită drept container; acesta este mecanismul pentru care s-a optat în exemplul de mai sus, rolul de container avându-l `TextBox`-ul `mesaje`.

La încărcarea paginii în server, se lansează evenimentul `Page_Load`, care apelează implicit funcția de tratare cu același nume, moment în care se accesează baza de date și se adaugă informația, 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 aplicații Web folosind Visual Studio

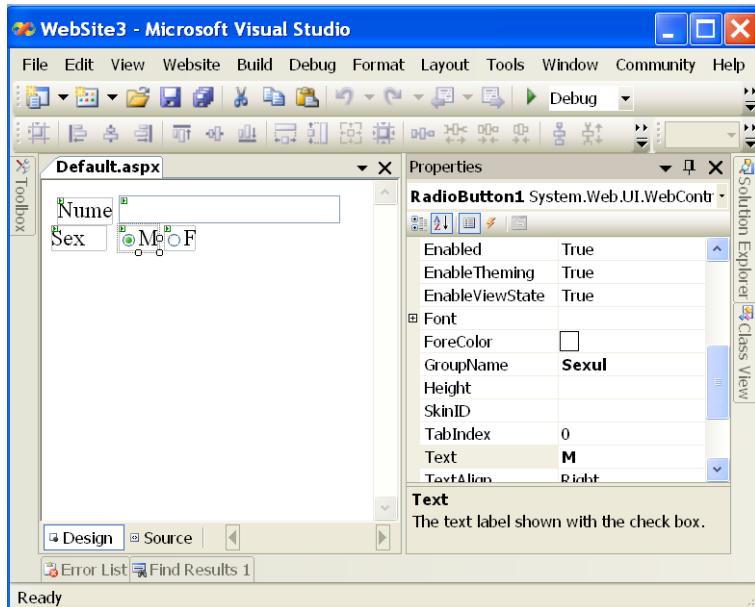
Sub .NET, **File / New / Web Sites / ASP.NET Web Site** și se alege un nume de aplicație și ca localizare se alege **File System**; pentru a se putea lucra rapid, cel puțin pentru faza de realizare și depanare a aplicației este preferabil această localizare.

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

- **File System** – dacă se experimentează pe un site creat provizoriu într-un director local sau de pe un alt calculator din rețea expus ca partajabil (shared). Se poate crea și un director nou, folosind iconul *Create New Folder* sau pur și 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** și folosește un port, ales adhoc, nu pe cel folosit de **IIS**; avem avantajul că putem muta directorul, căci el nu e luat în evidență IIS (metabaza IIS) ca director virtual și 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 iconul din colțul dreapta-sus *Create New Web Application* se poate crea chiar acum un nou director virtual.
- **FTP Site** - când lucrăm pe un site la distanță (sub alt IIS, decât cel local) și-l accesăm prin FTP, furnizând informațiile de conectare (*FTP site, port, director, user, password*).
- **Remote Web Server** - când lucrăm pe un site la distanță și-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* și *password*.

Spre deosebire de Windows Forms, aplicațiile 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; controlurile 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 declanșează 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 și mediul când rulam sub mediul integrat (extrasă din bara de adresă) și va funcționa 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 aplicația (`C:\Inetpub\wwwroot\prima`) observăm mai multe fișiere:

- **xxx.aspx** – conține **codul de vizualizare** a paginii Web (implicit se numește **default.aspx**);
- **xxx.aspx.cs** – conține **codul sursă C#** pentru declarațiile controalelor și funcțiilor 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); conține informațiile și codul de tratare a evenimentelor de la nivel de aplicație.
- **web.config** – conține detaliile de configurare a site-ului web și care sunt preluate cu eventuale modificări la nivel de aplicație.

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

În scopul prevenirii conflictelor de apartenență 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 aparține unor namespace-uri diferite.

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

Fazele de lucru cu o pagină Web

1. **Inițializarea paginii** – este rezultatul unei cereri de la un browser; instantiază controalele din pagină și le inițializează cu valori indicate prin **Properties**; dacă suntem pe postback (adică pagina nu e la prima solicitare) valorile curente ale controalelor sunt extrase din **ViewState** și folosite la inițializare. În această fază se declanșează și evenimentul **PageInit**, dar e puțin folosit, căci se declanșează **înainte** de instanțierea și încărcarea controalelor cu valori utile.
2. **Inițializarea pe baza codului indicat de utilizator** – declanșează evenimentul **PageLoad** și deci va executa codul scris de programator în funcția de tratare PageLoad. Denumirea de PageLoad este aleasă pentru a sugera că ușual, pe server se încarcă și pagini care au mai fost vizualizate în browser și se întorc în server cu completări. În funcția de tratare PageLoad, pe ramura `!Page.IsPostBack` se pot pune valori de inițializare ale controalelor; pe cealaltă ramură valorile nu trebuie puse căci se preiau automat (vezi faza anterioară) din ViewState. Dacă inițializăm controlul prin cod, atunci 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** oricărora evenimente de utilizator.
4. **Tratarea evenimentelor de utilizator** – este o fază derulată după ce pagina e complet încărcată în server, inițializată și validată. În principiu, evenimentele de utilizator ar putea fi împărțite 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 selecție 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 forțează notificarea imediată a severului despre producerea acestui tip de eveniment.
5. **Legarea datelor** – este o fază care se derulează în două trepte:
 - **insert, delete și 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 funcțiile de tratare evenimente nu beneficiază de cele mai recente date, aduse în controale.

Legarea datelor se face automat la fiecare postback; dacă se dorește scrierea de cod ce folosește 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 și **înainte** de redarea paginii ca HTML. Ușual 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 și pagina este redată ca HTML și pornește spre client. În acest moment se declanșează evenimentul **Page.Unload** și nicio modificare asupra paginii nu mai este posibilă. *Garbage collector*-ul eliberează toate instanțele de obiecte ce nu mai sunt referite și se declanșează evenimentul **Page.Disposed**, care încheie ciclul de viață al paginii în server.

Pentru așezarea mai flexibilă 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 secțiunea HTML, nu cel din secțiunea Standard, care ar necesita resurse ASP.NET, 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 și eliberare resurse pe server.

Similar stau lucrurile și pentru **Request** și **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 același 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) conține funcții DLL ce se încarcă dinamic, în același proces cu serverul.

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

- **filtre** – încărcate la inițializare server și folosite la tratarea evenimentelor la nivel de server;
- **extensiile** - încărcate la prima solicitare și apoi partajate între aplicații.

Dezavantaj: la eșuare, afectează tot serverul, făcând parte din același proces.

1. IIS primește cererea și după extensia de fișier vede ce DLL din ISAPI deservește cererea. Dacă extensia este **.aspx**, adică ASP.NET, atunci invocă funcția adecvată din **aspnet_isapi.dll** pentru tratarea cererii (în consola IIS, taburile **MIME**, respectiv **Handler Mapings** pentru a vedea **associerile dintre extensiile de fișiere și 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 fișierul **.aspx** obținând un **assembly**, crează un **application domain** și instruiește mașina virtuală CLR (Common Language Runtime) să execute **assembly-ul** în contextul **application domain** creat
4. **assembly** folosește clase din FCL (Framework Class Library) pentru a rezolva cererea și generează un **răspuns**;
5. **worker process** preia răspunsul generat, îl împachetează și-l transmite procesului din **aspnet_isapi.dll**, care îl transmite la rândul lui serverului IIS pentru a fi înaintat clientului, la distanță.

Uzual, structura unei pagini include trei secțiuni distincte:

- **secțiunea de directive**, prin care se stabilesc condițiile 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.
- **secțiunea de cod**, introdusă prin tag-ul `<script>` și care precizează codul executabil folosit la execuția unor comenzi din pagină (uzual precompilat, nu neapărat script, cum s-ar putea înțelege din delimitator);
- **secțiunea de machetare, Page layout**, reprezentând scheletul paginii și 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 spațiu; **nu trebuie separat cu spațiu 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" >` și `</form>` putem pune cod C#, declarat ca script:

Eventualele erori de compilare sunt raportate doar când se rulează pe aceeași mașină (`localhost`), dar se poate cere raportarea erorilor și când se rulează pe un server aflat la distanță; în acest scop în fișierul de configurare `web.config`, se pune în rubrica adecvată codul următor:

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

Rezultatul compilării este un DLL și este pus în directorul `c:\winnt\Microsoft.Net\Framework\...\\Temporary ASP.NET Files`. (precum și în directorul aplicației `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 fișierul `Default.aspx.cs` se definește 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 și 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 funcție de selecția dintr-un control `DropDownList`, la apăsarea pe buton să se efectueze Adunare, Scădere sau nimic (None).

Să se modifice programul astfel încât să se renunțe la buton, operația declanșându-se automat la schimbarea selecției din `DropDownList`.

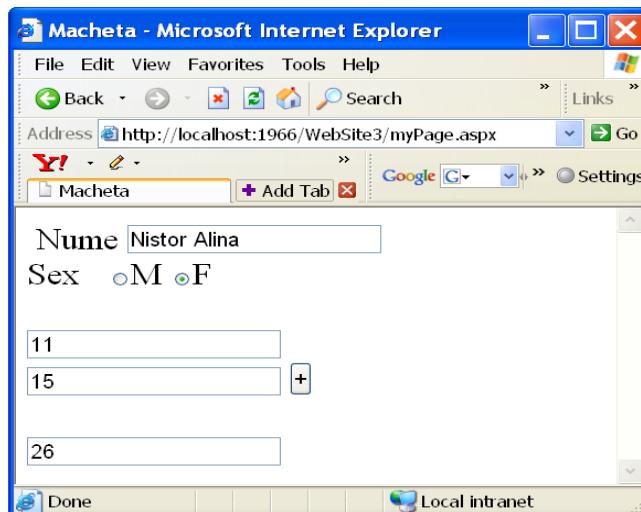
- 2.** Aplicatie cu o **macheta de introducere date**, care la Save face mutarea datelor într-un textBox multiline (sau in ListBox).
- 3.** Pentru familiarizarea cu mediul .NET se va elabora o aplicație simplă, exemplificând lucrul cu controale de tipul: **Button**, **TextBox**, **RadioButton**, **DropDownList**, **AddRotator**, **Calendar**.

TextBox; RadioButton; Button

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

Se adaugă **apoi** trei textBox-uri cu identificatorii **a**, **b** și **r** și un buton inscriptonat cu „+”, care la apăsare (evenimentul Click) declanșează adunarea a două numere și afișarea rezultatului. Funcția 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** și se populează adăugând câteva linii în colecția (proprietatea) **Items**;
- și se fixează proprietatea **AutoPostBack** pe **true**, pentru a anunța serverul de orice schimbare de selectie;
- se tratează evenimentul **SelectedIndexChanged** prin funcția:

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

- funcția preia valoarea selectată și o pune într-un TextBox numit **tbTip**.

AddRotator

Fișierul **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 afișeze 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 fișierele imagine citate în `reclama.xml`.

Calendar

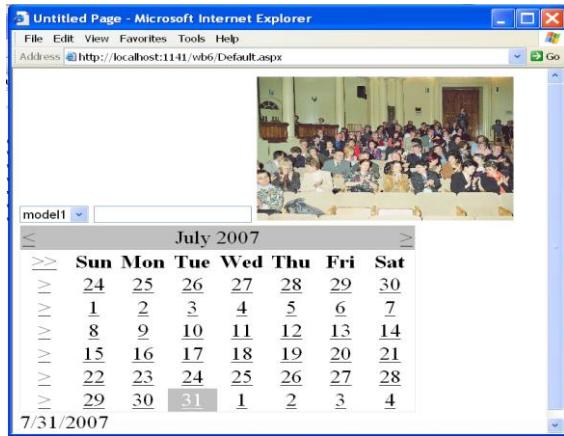
- Se draghează un control **Calendar** și i se fixează proprietatea **SelectionMode** pe valoarea **DayWeekMonth**;
- i se tratează evenimentul `SelectionChange` cu funcția:

```

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;
}

```

Funcția preia datele selectate din calendar și 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 și un **control** având o mulțime de proprietăți și evenimente pe care le recunoaște. Este și **container** de controale, colecția `Controls` furnizând acces la controalele de pe pagină.

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

- **Init** – generat la crearea unei instanțe de tip Page; se poate folosi și pentru a ataşa dinamic handlere altor evenimente recunoscute de pagină și ce pot fi declanșate ulterior.
- **Load** – la terminarea inițializării și încărcarea obiectului Page în memoria serverului; se poate folosi și pentru a ataşa cod de inițializare controalelor de pe pagină.
- **PreRender** – declanșat înainte de redarea vizuală a paginii pe ecran; este folosit pentru eventuale retușuri 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 aplicațiilor să se adapteze dinamic la schimbările contextuale din mediul de rulare

<%...%> este un bloc ce ține de redarea interfeței (eventuale evaluări) și se pune în interiorul unei metode; deci nu poate conține definirea unei alte metode !

Tratarea unui eveniment se poate face într-o din modalitățile:

1. adăugând un **handler** la delegatul disponibil pentru acel eveniment;
2. sau **supraîncărcând metoda** (`OnXXXX()`) moștenită 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 atașarea / detașarea **dinamică** a **mai multor metode** de tratare a același eveniment;
- aceeași funcție de tratare se poate aplica și altor evenimente fără a o rescrie în diverse clase.

În .NET există clasa care descrie un delegat generic, numit **EventHandler**; instantierea clasei produce un obiect delegat și presupune furnizarea prototipului funcțiilor de tratare ce le va conține.

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

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

și arată că acest tip de delegat poate ține referința oricărei funcții ce primește un obiect generic și un bloc de parametri cu argumentele specifice unui eveniment generic și returnează `void`.

Când adăugăm unui eveniment o funcție de tratare, se instanțiază un nou delegat, care prin constructor primește și referința pe care o va transporta:

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

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

Mecanismul `AutoEventWireup="true"`

În aplicațiile Web mai există un mecanism de atașare a unor funcții de tratarea evenimentelor unei pagini Web numit **mecanism AutoEventWireup**. Este cel de-al treilea mecanism de atașare cod executabil unor evenimente, alături de supraîncărcarea funcțiilor moștenite din clasa de bază, respectiv mecanismul delegării.

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

Presupunem că se creează o aplicație Web numită `AprindereAutomata`. În fereastra de vizualizare se alege formatul HTML și se modifică codul sursă pentru a arăta astfel:

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

Atenție 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 funcționează de sine stătător, fără a necesita construire de delegați de evenimente și atașare de funcții de tratare. Prețul plătit este că funcțiile de tratare 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` și `AutoEventWireup="true"` deoarece unele din funcțiile de tratare pot fi apelate de două ori; explicația constă în faptul că în cadrul mecanismului `AutoEventWireup` ASP.NET leagă automat la

delegații corespunzători, metodele cu numele impus, iar când programatorul atașează metodele cu + le leagă a doua oară!

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

Mecanismul **code-behind** separă codul necesar descrierii interfeței utilizator (user interface) de codul ce detaliază algoritmii de prelucrare (business logic). Prima categorie se pune în fișierul de tip **.aspx**, cea de-a doua în fișierul **.cs**; astfel este posibil și lucru în echipă, designerii ocupându-se de proiectarea vizuală a paginii, iar programatorii de prelucrări.

Cele două fișiere sunt legate prin atributul `Codebehind` din directiva `Page`; acesta precizează fișierul cu cod sursă. Linkeditarea apare astfel ca fiind inițiată din pagina de utilizator.

Directiva `Page` precizează atribute necesare compilării și vizualizării paginii Web.

Src - citează numele fișierului sursă ce conține clasa code-behind care este compilată dinamic la solicitarea unei pagini Web. Când folosim Visual Studio .NET nu este nevoie de acest atribut doarece clasa code-behind este precompilată.

Inherits – furnizează numele clasei code-behind ce conține codul paginii ASPX și din care clasa generată dinamic va moșteni prin derivare.

Desi **src** furnizează aceeași informație 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 însotește pagina Web. Pentru că mediul Visual Studio separă și el codul de vizualizare de cel de prelucrare, are totuși nevoie la momentul proiectării aplicației de numele fișierului ce ține codul sursă C#, pentru a-l compila din vreme; acesta e indicat prin `Codebehind`. Așadar, `Codebehind` este un atribut înțeles și folosit numai de VS (nu și de ASP.NET la execuția paginii) pentru localizarea fișierului sursă ce conține definirea claselor folosite în pagină.

Asocierea src și Inherits sugerează o pagină fără compilare prealabilă.

Asocierea Codebehind și 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 beneficiarului există altă versiune de .NET, clasa folosește versiunea cu care a fost ea creată.

Meniu **Project / Show All Files** permite vizualizarea în Solution Explorer a tuturor fișierelor folosite în cadrul unui proiect, altfel implicit sunt vizibile doar o parte dintre ele.

Fișierul de configurare **web.config** conține toate clauzele de configurare a proiectului, inclusiv `debug = "true"`, necesară când aplicația nu poate fi rulată în mod depanare, căci ea a fost compilată fără opțiuni de depanare. Punând-o pe true, ne asigurăm că recompilarea se face cu adăugarea tuturor facilităților de depanare.

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

Tipologia controalelor dintr-o aplicație ASP.NET

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

Web Server Controls

- orientate obiect; built-in și 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 și mai sus, la containerul controalelor.

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 atașează controlului un mic script ce face ca evenimentul să fie transmis serverului cu ocazia altui eveniment, uzual Click de buton.

Image ``

`ImageUrl` indică locația URL în care rezidă imaginea de afișat

CheckBox și RadioButton - `<input type="checkbox">` și `<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">` `ImageClickEventArgs` încapsulează informații 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 anunțate imediat serverului;
- **Non-postback**, care nu sunt anunțate imediat serverului, pentru a nu mări excesiv traficul în rețea.

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 dețin o proprietate booleană **AutoPostBack**, cu valori implicate, prin care se specifică dacă evenimentele generate de acestea sunt **postback** sau **non-postback**. După valoarea implicită a acestei proprietăți putem distinge:

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

Modificând valoarea proprietății **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	Categorie	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>
	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 conțin 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

Controlul GridView

A. Lucru cu GridView. Aplicații

- 1. BD Oracle**
- 2. SqlServer**
- 3. BD FoxPro**
- 4. BD Access**

B. Paginarea datelor din grid

C. Modificarea / stergerea datelor din grid

D. Sortarea datelor din grid

E. Inserarea datelor în grid

Lucru cu DataGrid. Aplicații

- 1. BD Oracle : aplicația oracleTest**
- 2. SqlServer : aplicația SqlTest**
- 3. BD FoxPro: aplicația foxProAdmit**
- 4. BD Access : aplicația gridButoane**

Paginarea datelor din grid

Properties **Allow Paging** sau smart tag
(**Enable Paging** bifat)

PagerSettings / Mode: Numeric

Page Style / Page Size: 4

```
private void dg_PageIndexChanged (object source,  
    System.Web.UI.WebControls.DataGridPageChangedEventArgs e)  
{  
    dg.CurrentPageIndex = e.NewPageIndex;  
    this.LeagaGrid();  
}
```

Modificarea datelor din DataGrid

	Cod	Denumire	Pret
Edit	abc	abc	0
Edit	abc	abc	0.1
Edit	abc	abc	0.2
	1	2	

SqlDataSource - SqlDa

GridView Tasks

- Auto Format...
- Choose Data Source:
- Configure Data Source
- Refresh Schema
- Edit Columns...
- Add New Column...
- Enable Paging
- Enable Sorting
- Enable Selection
- Edit Templates

Fields

Available fields:

- HyperLinkField
- ImageField
- ButtonField
- CommandField
 - Edit, Update, Cancel
 - Select
 - Delete
- TemplateField

Add

Selected fields:

- Edit, Update, Ca...
- Cod
- Denumire
- Pret

▲ ▼ X

CommandField properties:

ButtonType	Button
CancelImageUrl	
CancelText	Cancel
DeleteImageUrl	
DeleteText	Delete
EditImageUrl	
EditText	Edit
FooterText	
HeaderImageUrl	
HeaderText	
InsertImageUrl	
InsertText	Insert
NewImageUrl	
NewText	New

Modificarea datelor din DataGrid

Inserare Înregistrări în DataGrid

```
private void btnAdd_Click(object sender, System.EventArgs e)
{
    DataRow linNoua; dataSet1=(DataSet) Cache["ProdCache"];
    linNoua = dataSet1.Tables["produse"].NewRow();
    int cod=Convert.ToInt32(
        dataSet1.Tables[0].Rows[dataSet1.Tables[0].Rows.Count-1][0]);
    cod++;           linNoua["codp"] = ""+cod;
    linNoua["denum"] = "";      linNoua["pret"] = 0;
    dataSet1.Tables["produse"].Rows.Add(linNoua);
    Cache["ProdCache"]=dataSet1; dg.DataSource = dataSet1;
    dg.DataBind();
}
```

Sortarea datelor din grid

- cu clauza **ORDER BY** în fraza **SELECT**
- cu obiectul **DataView** sortat și legat ca **DataSource** pentru grid
- cu:
 - **Properties / AllowSorting: true**
 - Tratare eveniment **SortCommand**

```
private void dg_SortCommand( object source,
    System.Web.UI.WebControls.DataGridSortCommandEventArgs e)
{
    dataSet1=(DataSet) Cache["MatCache"];
    DataView sortat = new DataView(dataSet1.Tables["materiale"]);
        // sortat.Sort="pret ASC";
    sortat.Sort=e.SortExpression;
    dg.DataSource = sortat; dg.DataBind();
    tbMesaje.Text+="Sortare dupa "+e.SortExpression+
                    ":" sortat.Count+" inreg.";
}
```

DataSet mixt

■ SELECT, DELETE, UPDATE aadevate

```
private void LeagaGrid()
{
    strConex="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\prod.mdb";
    oleDbDataAdapter1=new OleDbDataAdapter(
        "SELECT produse.denum, materiale.denm," +
        "consumuri.codp,consumuri.codm, consumuri.consum " +
        "FROM produse, materiale, consumuri " +
        "WHERE produse.codp = consumuri.codp AND " +
        "consumuri.codm = materiale.codm " +
        "ORDER BY produse.denum ", strConex );
    dataSetMixt = new DataSet();
    oleDbDataAdapter1.Fill(dataSetMixt,"mixta");
    DataGrid1.DataSource = dataSetMixt; DataGrid1.DataBind();
}
```

A. Securitatea accesului la BD

- 1. Authentication**
- 2. Authorization**
- 3. Impersonation**

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 și 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 aplicația care conține un GridView și o sursă de date, pașii de lucru ar putea fi următorii:

- În **Solution Explorer** cu buton dreapta mouse se cere **Add Reference** și se localizează cu **Browse** și se leagă la proiect cele două fișiere **ZedGraph.dll** și **ZedGraph.Web.dll**, descărcate de pe Internet și care conțin clasele cu care lucrează controlul web.
- Se construiește **în directorul aplicației un subdirector de lucru** numit **ZedGraphImages**, în care controlul își compune imaginile grafice.
- Se adaugă o pagină nouă pe care o vom numi **WebFormZedGraph** (**Website / Add New Item** și alegem **WebForms**) și pe ea (fișierul WebFormZedGraph.aspx.cs) se pune codul din funcțiile **Page_Load** (care leagă funcția de tratare a evenimentului **RenderGraph**) și **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 aplicăiei.

```
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 "Bar":
            {
                BarItem myCurve = myPane.AddBar(
```



```

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

```

- Pentru variație se pune pe prima pagină și un combobox (**DropDownList**) din care se alege tipul graficului; pentru aceasta colecția 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 anunța 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 selecție în `DropDownList1`, se extrage setul de date din sursa de date și 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, pașii de parcurs sunt următorii:

- In Solution Explorer cu buton dreapta mouse se cere **Add Reference** și se alege din pagina **COM, Microsoft Office Web Component 11.0**. Dacă avem o aplicație mai veche care adusese deja o copie a referinței, o putem selecta pe aceasta, cu *Browse*.
- Depinzând de versiune (**vezi Properties când ținem selectată noua referință**) adăugăm și `using`-ul corespunzător bibliotecii adusă ca referință:
`using Microsoft.Office.Owc11;` sau `using OWC11;`
 sau altceva, în funcție de controlul de chart folosit.
- Se adaugă o pagină nouă (**Website / Add New Item** și alegem **WebForms**) și pe ea se pune codul din `WebFormGrafic.aspx.cs` al aplicației 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"];
    //Object 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 conține un obiect **Image** adus din **ToolBox**, care va afișa chart-ul furnizat de **WebFormGrafic**, deoarece are proprietatea **ImageUrl** = **WebFormGrafic**, pusă prin cod, folosind o cale relativă la directorul aplicației.

Graficul e furnizat deja "preparat", doar îl extragem ca gif din spațiul chart-ului; putem avea mai multe chart-uri în același spațiu și le putem activa pe rând, câte unul.

- Pentru variație se pune **pe prima pagină** și un combobox (**DropDownList**) din care se alege tipul graficului; pentru aceasta colecția **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 anunța serverul despre evenimentul de schimbare a tipului de grafic, obligându-l astfel să retraseze graficul la reîncărcarea paginii principale. Pe evenimentul de schimbare index selecție în **DropDownList1**, se precizează și prin cod sursă că imaginea va prelua ca sursă graficul din pagina **WebFormGrafic.aspx** indicând în plus prin **QueryString** și 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 referinței în proiect, se poate bifa opțiunea **Copy to local**, prin care controlul este copiat în directorul aplicației, astfel încât să potă fi testată aplicația și pe o mașină 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 țin ca proprietăți ce sunt de fapt referințe 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, ștergerile sau inserările în baza de date (metoda `Update` a adaptorului);
- pot fi folosite independent, caz în care pot genera la execuție 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 execuție.

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

- `ExecuteReader`, care întoarce o colecție 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 și returnează un `int`, indicând numărul de înregistrări afectate prin modificări, ștergeri 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` și 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 colecție de înregistrări:

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

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

Parcurgerea înregistrărilor cu `DataReader` este simplă și se asemănă cu citirea câte unei înregistrări dintr-un fișier. 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 aplicație care utilizează `DataReader` presupune parcurgerea următorilor pași:

- se crează o aplicație nouă, în care prin suprascrierea fișierelor `webForm1.aspx.cs`, `webForm1.aspx` se aduce codul sursă prezentat mai jos;
- se atașează delegații pentru cele trei butoane, căci prin copiere nu s-au adus și legăturile;
- se face o copie pentru fișierul `prod.mdb`, căci va fi alterat prin `ExecuteNonQuery`;
- se rulează și comentează fiecare linie sursă, constatănd efectele rulării.
- se poate completa aplicația cu un `GridView`, pentru vizualizarea simultană a tuturor tuplurilor selectate;

- pe un buton **puneInGrid** se adaugă codul de la funcția **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+="\n"+dr["codp"]+ " " + dr["denum"]+ " " + dr["pret"];
//}
GridView1.DataSource = dr; GridView1.DataBind(); dr.Close(); con.Close();
```

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

Comparății între cele două modalități de folosire a obiectelor de tip **xxxCommand**:

Echipând un **dataAdapter**, 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.

Folosite individual, obiectele de tip **Command**:

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**.

Observație. Se poate folosi metoda **ExecuteReader()** și în locul celorlalte metode (**ExecuteNonQuery**, **ExecuteScalar**), căci metoda se va adapta după cerință.

În esență, obiectul de tip **XxxCommand** primește din construcție cele două informații de care avea nevoie și adaptorul: stringul SQL de executat și conexiunea:

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

Ce se întâmplă însă dacă părți din comandă sunt variabile și nu se cunosc la momentul construirii obiectului **xxxCommand**, ci se schimbă de la o execuție la alta ?

Se poate rezolva această problemă în două moduri:

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

Obiectele de tip **XxxCommand** au o colecție de parametri, obiecte de tip **Parameter**, ce sunt caracterizate prin **nume, tip, direcție și valoare**; acești parametri circulă împreună cu comanda și conțin valori în reprezentare internă; trebuie să coincidă cu tipul celor din baza de date, când sunt folosiți la stocare valori în BD, dar pot fi și de lat tip dacă sunt folosiți în alt scop (spre exemplu, parametru **@PretMin** poate fi și de tip **int**, deoarece el este folosit doar în comparații, chiar dacă prețul de referință 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();
}

```

Observație. 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 conține o imagine !

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

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

```

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 dorește modificarea denumirii produselor care încep cu « p », prin adăugarea prefixului « txt ». Funcția care realizează acest lucru tratează evenimentul Click al unui buton inscripționat 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 expt)
{
    mesaje.Text=
        "Esec pe ExecuteNonQuery: "+ expt.Message;
}
finally
{
    if (conn.State == ConnectionState.Open) conn.Close();
}
}

```

Pentru rulare sub **Visual Studio 2008**, cu bază de date sub **SQL Server Express**, 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.SqlClient;
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 :

codp	denum	pret
100	prod_100	100
200	prod_200	200
300	prod_300	300
400	prod_400	400
Numar produse comercializate: 4		
<input type="button" value="Incarca"/> <input type="button" value="ExScalar"/> <input type="button" value="ExNonQ"/>		

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 porțiuni de cod, compilabile separat, existente pe server.

Avantajele folosirii procedurilor stocate:

- procedurile sunt deja compilate și **deci nu mai conțin erori de compilare**, fiind direct lansabile în execuție; fiind cunoscute de sever beneficiază și de utilizarea **statisticilor interne** pentru optimizarea accesului la datele unei tabele;
- aflându-se pe server, procedurile stocate nu au nevoie să fie compuse și retransmise prin rețea de fiecare dată când sunt invocate; astfel **accesul este mai rapid** deoarece se transmit doar parametrii de apel și 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 și instrucțiuni individuale.

Observații.

- 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 același 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 aceștia sau să fie convertiți la momentul folosirii (vezi `webBdProc1`, cu pret `float` comparat cu param `int`).
- Concordanța la tipurile asociate cu `DateTime` este delicată existând multe formate de data calendaristică, diferind ca lungime și ca moment de referință ales.

Pentru început vom crea și folosi o **procedură fără parametri**; pentru simplitate presupunem că avem funcții distincte care crează, respectiv apeleză procedura, la apăsarea către unui buton inscripționat corespunzător. Funcția de creare încearcă mai întâi să șteargă o eventuală versiune mai veche a aceleiași proceduri, pentru a ne permite îmbunătățirea 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 modalități 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 prețul peste o valoare, dată ca text într-un TextBox al formei, procedura s-ar schimba, deoarece comanda trebuie să declare acum și parametri de intrare / ieșire; în acest caz este nevoie să lucrăm și cu obiecte de tip **Parameter**.

Așa cum spuneam, obiectele Command dispun de o colecție numită **Parameters**, în care prin metoda **Add**, adăugăm parametrii pregătiți anterior sau furnizăm metodei informația necesară pentru a construi ea acești 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 colecției **Parameters** are mai multe versiuni supraîncărcate, în funcție de numărul de argumente folosite la construirea unui obiect de tip parametru. Se observă că, în afara faptului că metoda a construit și a adăugat un parametru de tip **int** la o comandă, ea returnează și

referința parametrului, astfel încât acesta să poată fi ulterior echipat și cu alte proprietăți, spre exemplu direcția:

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

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

sau

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

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

Reamintim că în ADO.NET parametrii trebuie să se numească la fel cu cei definiți în procedura stocată și să aibă **același tip și aceeași lungime**.

Direcția unui parametru poate fi:

- **Input** – parametru de intrare în procedură;
- **Output** - parametru de ieșire, returnat de procedură și care nu poate conține date de intrare pentru procedură;
- **InputOutput** - parametru folosit în același timp și pentru intrare și pentru ieșire, în comunicarea cu procedura;
- **ReturnValue** - parametru de ieșire, returnat de subprogramele de tip *function*.

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

Funcțiile 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 reținut că parametrii de ieșire sunt accesibili doar după închiderea DataReader-ului.

Exercițiu. Transformați procedura din aplicația de mai sus, astfel încât să returneze și câte produse sunt în baza de date, cu prețul cuprins între două valori date.

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

```
protected void btnCreateProc_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 strCreate = "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 = strCreate;
    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();

}

```

Baze de date pe INTERNET

- A. Server Web - IIS Microsoft**
- B. Visual Studio .NET ASP.NET**
- C. Server de BD - SQL Server**

Paradigma INTERNET

A. Internet Information Service

**B. Mixarea informațiilor în pagina Web:
aplicații practice**

- **dataOra.aspx**
- **functia**
- **bdPage.aspx**

C. ASP.NET și Visual Studio .NET

Internet Information Service

- Internet
- TCP - IP
- Limbaje script: HTML, DHTML, XML;
- Web Servers: **Apache; IIS = inetinfo.exe**
- Instrumente vizuale ASP.NET;
Web Forms; componente .NET; Windows Forms
logica prezentării / logica de business

- Control Panel / Programs /
 - Programs and Features
 - Turn Windows features on/off
 - **Web Management Tools**
 - IIS Manager Console
 - **WWW Service**
 - App Development Features
 - ✓ .NET; ASP.NET; ISAPI Extensions; ISAPI Filters
 - **IIS Hostable Web Core**
- Start / IIS: **IIS Manager**
- C:\Windows\System32\inetsrv
 - InetMgr.Exe
 - InetInfo.exe

- Directorul fizic **c:\inetpub\wwwroot;**
- Virtual Directory (W10)
 - Control Panel / **Administrative tools** (cu Search)
 - **IIS Manager**
 - Tabs: **ASP.NET / FTP / IIS/ Management**
 - din panel dreapta: **View sites**
 - **Default Web Site** / mouse right / **Add Virtual Directory;**
- Internet Server Application Programming Interface
- IIS: Start / Stop / Restart
- Handler Mapings
 - *.**.aspx PageHandlerFactory**
- MIME Types
- Default documents
- Directory browsing

Mixarea informațiilor în pagina Web: aplicații practice

▪ Dinamicitate Client side

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

▪ Dinamicitate Server side

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

Mixarea informațiilor în pagina Web: aplicații practice

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

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

<%
for (double i = 1.0; i<=10.0; i++)
{ Response.Output.Write( "<tr><td>{0}</td><td>" + "{1:f}</td><tr>",
i, Patrate(i));
} %>
```

- Codebehind
- Inherits

Mixarea informațiilor în pagina Web: aplicații practice

```
private void Page_Load(object sender, System.EventArgs e)
{
    string strSql = "SELECT codp, denum, pret FROM produse";

    SqlConnection con = new SqlConnection(
        "data source=10.2.64.73; initial catalog=doctorat;persist security
         info=False; user " + "id=sa;password=as"
    );

    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())
    {
        mesaje.Text+='\r\n'+dr["codp"]+ " " + dr["denum"]+ " " + dr["pret"];
        mesaje.Text+='\r\n';
    }
    dr.Close();            con.Close();
}
```

ASP.NET și Visual Studio .NET

- Crearea aplicației WEB cu **ASP.NET**
- Fișiere componente
- Lansarea din IE:
<http://localhost/prima.aspx>

Flux procesare pagină Web

1. Initializare pagina

- ! isPostBack: valori din *Properties*
- isPostBack: valori din *ViewState*
- **PageInit** – putin folosit (controale neinitializate incă)

2. Initializare pagina prin cod

- **PageLoad**
- **if(isPostBack)**
- **else ...**

Flux procesare pagină Web

3. Validarea prin controale de validare **în server**

- validarea **în browser** s-a facut înainte de plecarea paginii spre server

4. Tratare evenimente de utilizator

- evenimente cu răspuns imediat
 - **AutoPostBack, Click**
- evenimente de modificare
 - **TextChanged, SelectedIndexChanged**

Flux procesare pagină Web

5. Data binding

- **insert, delete și update**
- **select** se execută după Page.PreRender()
- dezavantaj: functiile de tratare nu au valorile cele mai recente !

6. Eliberarea resurselor

- **PageUnload**
- **PageDisposed**
- Aplicația **8_EventsFlow**

Gestiunea stării în aplicațiile WEB

- A. View state (state bag)**
- B. Session state**
- C. Application state**
- D. Implicațiile gestiunii stării în lucru cu baze de date**

A. View state (state bag)

- **Concepție:**
 - "stateless connection"
 - **Starea:** mulțimea de valorile conținute în controalele (**EnableViewState**) și variabilele unei pagini
- **Salvarea implicită a controalelor**
- **Salvarea/recuperarea conținutului variabilelor:**

```
ViewState.Add("kContor", contor);
contor = (int)ViewState["kContor"];
```
- **Aplicația *stareView***

- **EnableViewState :**
 - **false:** reconstruire pagină de la zero
 - **true:** recompunere pagină din starea salvată anterior
- **AutoPostBack : true / false**
 - **postback :** Button, Calendar, DataGrid, DataList, ImageButton, LinkButton, Repeater.
 - **non-postback:** CheckBox, CheckBoxList, DropDownList, ListBox, RadioButtonList, RadioButton, TextBox.

B. Session state

- **Sesiune:** secvență de cereri lansate de la același browser client și derulate la intervale de timp care să nu depășească intervalul de ***timeout*** definit în fișierul de configurare web.config (implicit 20 min);
- **Session** – obiect care ține informația specifică fiecărui utilizator în parte, pe durata unei sesiuni de lucru, când navighează între două pagini diferite.

B. Session state

- O pagină încarcă date și le salvează în obiectul Session
- Altă pagină preia datele din Session și le folosește
- Aplicația ***stareSesiune***

B. Session state

```
private void Page_Load(object sender,  
                      System.EventArgs e)  
{   Session["utilizator"] = tbNume.Text; }
```

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

```
private void Page_Load(object sender,  
                      System.EventArgs e)  
{   lblNume.Text = (string)Session["utilizator"]; }
```

C. Application state

```
int nrSesiune=(int)Application.Contents["nrSesiune"]+1;  
Application.Remove("nrSesiune");  
Application.Add("nrSesiune",nrSesiune);
```

- Contorizare cereri
- Contorizare sesiuni
- Aplicația *jurnalApp*

Implicațiile gestiunii stării în lucru cu baze de date

```
private void incarcaGrid_Click(object sender, EventArgs e)
{
    try
    {
        oleDbTypeAdapter1.Fill(dataSet1);
        DataGrid1.DataSource =this.dataSet1;
        DataGrid1.DataBind();
    }
    catch(Exception exc)
    {
        mesaje.Text=exc.Message;
    }
}
```

- Aplicația *stareGrid*