TDA-LABORATOR 5

Master Page, Transfer intre Pagini, Rutare

Master Page

Master Page este o pagina cu conținut partajat de mai multe Web Form. O Master Page acționează ca un fel de șablon în care putem introduce conținut specific reprezentat de un Web Form. Master Page este utilizata pentru a crea o apariție comună pentru două sau mai multe Formulare Web. Nu este bine sa utilizam Master Page pentru a defini aplicația generală sau logica de afaceri deoarece de multe ori va trebui să schimbam un Formular Web, astfel încât să se bazeze pe o altă Master Page ceia ce va implica, dublarea sau mutarea funcționalitatii paginii Master pe care se bazează, iar aceasta operatiune este obositoare și predispusa la erori. Pentru a ilustra utilizarea paginilor Master am creat un proiect gol ASP.NET si am adăugat o pagină Master numită Basic.Master. Puteți vedea conținutul acestui fișier, astfel cum a fost creat de Visual Studio, în Exemplul 1.

Exemplul 1**.** The contents of the Basic.Master master page

```
<%@ Master Language = "C#" AutoEventWireup = "true" CodeBehind = "Basic.master.cs"
        Inherits = "WebForms.Basic" %>
<!DOCTYPE html>
<html xmlns = "http://www.w3.org/1999/xhtml">
<head runat = "server">
        <title > </title>
        <asp:ContentPlaceHolder ID = "head" runat = "server">
        </asp:ContentPlaceHolder>
</head>
<body>
<form id = "form1" runat = "server">
        <div>
        <asp:ContentPlaceHolder ID = "ContentPlaceHolder1" runat = "server">
        </asp:ContentPlaceHolder>
        </div>
</form>
</body>
</html>
```

Configurarea Master Page. Paginile Master sunt marcate de directiva **Master**, care acceptă un subset de atribute pe care le suportă si directiva Page.

AutoEventWireup= "true" | “false”- Când este true, ASP.NET va apela automat metode din clasa codebehind ca răspuns la evenimentele din pagină.

CodeBehind - Specifică numele fișierului care conține clasa asociată cu pagina. Acest atribut este folosit numai pentru Web application projects. Pentru Web site projects este utilizat atributul CodeFile.

EnableViewState - Determină dacă ASP.NET Framework va folosi starea de vizualizare pentru a păstra starea controalelor. Valoarea implicită este adevărată și explicăm starea de vizualizare din capitolul 18 și o revizuim în profunzime în partea a 3-a.

Inherits- Definește o clasă pe care o moșteneste. Aceasta trebuie sa fie o clasă derivată din clasa Page. Valoarea acestui atribut este utilizată pentru a selecta clasa definită în fișierul specificat de atributul CodeBehind.

Language- Specifică limbajul .NET utilizat. Doar un singur limbaj poate fi folosită pe o pagină.

MasterPageFile- Specifică fișierul care va fi folosit ca **MasterPage**. Este folosit pentru a cuibări pagini principale.

Aceste atribute funcționează în același mod cu echivalentele directivei lor Page. Excepție este atributul MasterPageFile.

Master Page Placeholders. O pagina Master conține toate elementele partajate care dorim să apara în Formularele Web. Părțile specifice din pagină pe care dorim să le includem sunt reprezentate de controale ContentPlaceHolder.

Când Visual Studio creează un nou fișier Master Page, generează două controale ContentPlaceHolder - unul în secțiunea head și unul in sectiunea body. In Exemplul 2, puteți vedea cum am editat fișierul Basic.Master pentru a adapta conținutul pe care dorim să îl afișăm.

Putem avea mai multe pagini Master intr-o aplicatie, ceea ce ne permite să aplicam diferite pagini Master pe diferite părți ale aplicației.

Exemplul 2. Editing the contents of the Basic.Master page

```
<%@ Master Language = "C#" AutoEventWireup = "true"
        CodeBehind = "Basic.master.cs" Inherits = "WebForms.Basic" %>
<!DOCTYPE html>
<html xmlns = "http://www.w3.org/1999/xhtml">
<head runat = "server">
        <title > </title>
</head>
<body>

  <form id="form1" runat="server">
  <div>
        This is a list of <asp:ContentPlaceHolder ID = "ListType" runat = "server"/> :
  <ul>
        <asp:ContentPlaceHolder ID = "ListEntries" runat = "server" />
  </ul>
  </div>
  </form>
</body>
```

```
</html>
```

Am eliminat controalele implicite **ContentPlaceHolder** și am adăugat două noi controale. Vom folosi un **ContentPlaceHolder** al cărui ID este **ListType** pentru a afișa un singur cuvânt și unul cu ID-ul **ListEntries** pentru a afișa un set de elemente li.

Aplicarea paginii de Master. Cel mai simplu mod de a utiliza o pagină Master este de a crea un nou formular Web utilizând elementul șablon Master Page. Visual Studio vă permite să alegeți pagina Master pe care doriți să o utilizați și generează un formular Web care conține doar continutul necesar pentru a popula controalele ContentPlaceHolder. În Exemplul 3, puteți vedea conținutul unui formular web pe care l-am creat numit Colors.aspx.

Exemplul 3. The contents of the Colors.aspx Web Form

```
<%@ Page Title = "" Language = "C#" MasterPageFile = "~/Basic.Master"
AutoEventWireup = "true" CodeBehind = "Colors.aspx.cs" Inherits = "WebForms.Colors" %>
<asp:Content ID = "Content1" ContentPlaceHolderID = "ListType" runat = "server">
</asp:Content>
<asp:Content ID = "Content2" ContentPlaceHolderID = "ListEntries" runat = "server">
</asp:Content>
```

Formularul Web conține o directivă Page care specifică pagina principală și o pereche de controale **Content**. Controalele **Content** sunt corespunzatoare controalelor ContentPlaceHolder din pagina principală, iar elementele HTML și codul pe care le introducem într-un control **Content** vor fi folosite pentru a înlocui ContentPlaceHolder atunci când este procesat formularul Web. Atributul ContentPlaceHolderID vă indică ce control ContentPlaceHolder corespunde fiecărui control **Content**.

Pentru acest exemplu, am definit o metodă simplă în fișierul din spatele codului Colors.aspx.cs, pe care o puteți vedea în Exemplul 4.

Exemplul 4. Defining a method in the Colors.aspx.cs code-behind file

```
using System.Collections.Generic;
namespace WebForms {
        public partial class Colors : System.Web.UI.Page {
                public string[] GetColors() {
                        return new string[] {"Red", "Blue", "Green", "Orange"};
                }
        }
}
```

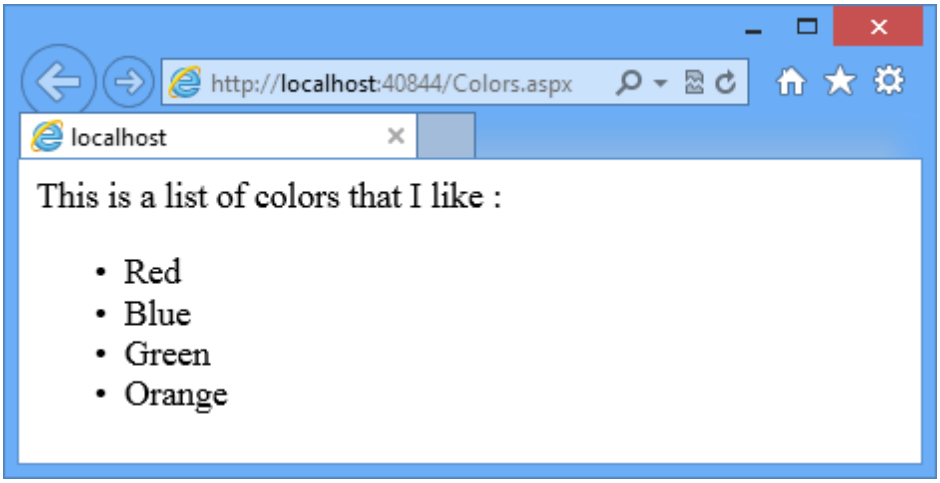
Metoda **GetColors** returnează un șir de articole pe care dorim să le afișăm în listă. În Exemplul 5, puteți vedea cum folosim această metodă pentru a completa Formularul web Colors.aspx, astfel încât să populăm controalele de conținut.

Exemplul 5. Completing the Colors.aspx Web Form

```
<%@ Page Title = "" Language = "C#" MasterPageFile = "~/Basic.Master"
```

```
        AutoEventWireup = "true" CodeBehind = "Colors.aspx.cs" Inherits =
        "WebForms.Colors" %>
<asp:Content ID = "Content1" ContentPlaceHolderID = "ListType" runat = "server">
        colors that I like
</asp:Content>
<asp:Content ID = "Content2" ContentPlaceHolderID = "ListEntries" runat = "server">
        <asp:Repeater SelectMethod = "GetColors" ItemType = "System.String"
                                                                runat = "server">
                <ItemTemplate>
                <li > <%#: Item % > </li>
                </ItemTemplate>
        </asp:Repeater>
</asp:Content>
```

Am populat primul control de conținut cu un șir simplu și am folosit un control asp:**Repeater** pentru a genera un set de articole li în cel de-al doilea control de **Content**. Când este procesat Formularul Web Colors.aspx, ASP.NET Framework va produce HTML preluat din pagina Master și amestecat cu fragmentele de conținut din Formularul Web. Puteți vedea rezultatul solicitării fișierului Colors.aspx înfigura de mai jos.



Transferul controlului intre Pagini

În aceasta sectiune, vom aborda tehnicile ASP.NET pentru schimbarea fluxului normal al unei cereri. Există diverse situatii in care trebuie sa direcționam fluxul cererii către o altă pagină. Aceste tehnici schimbă modul în care este tratată cererea.

Pentru a ilustra aceste tehnici, am creat o noua aplicatie web ASP.NET goala numita RequestControl. Am creat, de asemenea, un formular web numit Default.aspx, al cărui conținut îl puteți vedea în Exemplul 6.

Exemplul 6. The contents of the Default.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeBehind="Default.aspx.cs" Inherits="RequestControl.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
        <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
        <h3>This is Default.aspx</h3>
        <div>
                <input type="radio" name="choice"
                                        value="redirect302" checked="checked"/>Redirect
        </div>
        <div>
                <input type="radio" name="choice" value="redirect301"/>Redirect Permanent
        </div>

        <div>
                <input type="radio" name="choice" value="transferpage" />Transfer Page
        </div>
        <p><button type="submit">Submit</button></p>
    </div>
   </form>
</body>
</html>
```

Acest formular conține o serie de butoane radio care descriu diferite moduri de transfer și un buton care trimite formularul înapoi pe server. Am adăugat apoi un al doilea formular Web numit SecondPage.aspx, al cărui conținut îl puteți vedea în ***Exemplul 7***. Acest formular Web conține doar un mesaj simplu.

Exemplul 7. The contents of the SecondPage.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeBehind="SecondPage.aspx.cs" Inherits="RequestControl.SecondPage" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
        <title></title>
</head>
<body>
        <form id="form1" runat="server">
        <div>
```

```
            This is SecondPage.aspx
        </div>
        </form>
</body>
</html>
```

De asemenea, am adăugat un handler numit CurrentTimeHandler.ashx folosind Generic Handler item template. Puteți vedea conținutul fișierului CurrentTimeHandler.ashx.cs în Exemplul 8.

Exemplul 8. The contents of the CurrentTimeHandler.ashx.cs

```
using System;
using System.Web;
namespace RequestControl {
        public class CurrentTimeHandler : IHttpHandler {
                public void ProcessRequest(HttpContext context) {
                        context.Response.ContentType = "text/plain";
                        context.Response.Write(string.Format("The time is: {0}",
                                                DateTime.Now.ToShortTimeString()));
                }
                public bool IsReusable {
                get {
                        return false;
                }
                }
        }
}
```

Acest Handler generează un mesaj care conține ora curentă.

Redirectarea adreselor URL. Clasa HttpResponse oferă câteva metode și proprietăți care acceptă redirecționarea adreselor URL.

IsRequestBeingRedirected -Returnează true atunci când se redirecționează solicitarea. Această proprietate este utilă numai atunci când setați al doilea parametru pentru metoda Redirect și RedirectPermanent la false, astfel încât gestionarea cererii să nu fie încheiată imediat.

Redirect (URL) si Redirect (URL, end)- Trimite un răspuns cu un cod de stare 302, direcționând clientul către adresa URL specificată. Al doilea argument este o vatiabila booleana care, dacă este true, încheie imediat procesul de gestionare a cererii apelând la HttpApplication.CompleteRequest. Prima versiune este echivalentă cu setarea celui de-al doilea parametru la true.

RedirectLocation - Folosit pentru a seta URL-ul țintă atunci când efectuați redirecționări manual.

RedirectPermanent(URL) si RedirectPermanent (url, end) –Ca si metoda Redirect, dar răspunsul este trimis cu un cod de stare 301.

RedirectToRoute (name) -Trimite un răspuns cu un cod de stare 302 către o adresă URL generată dintr-o rută.

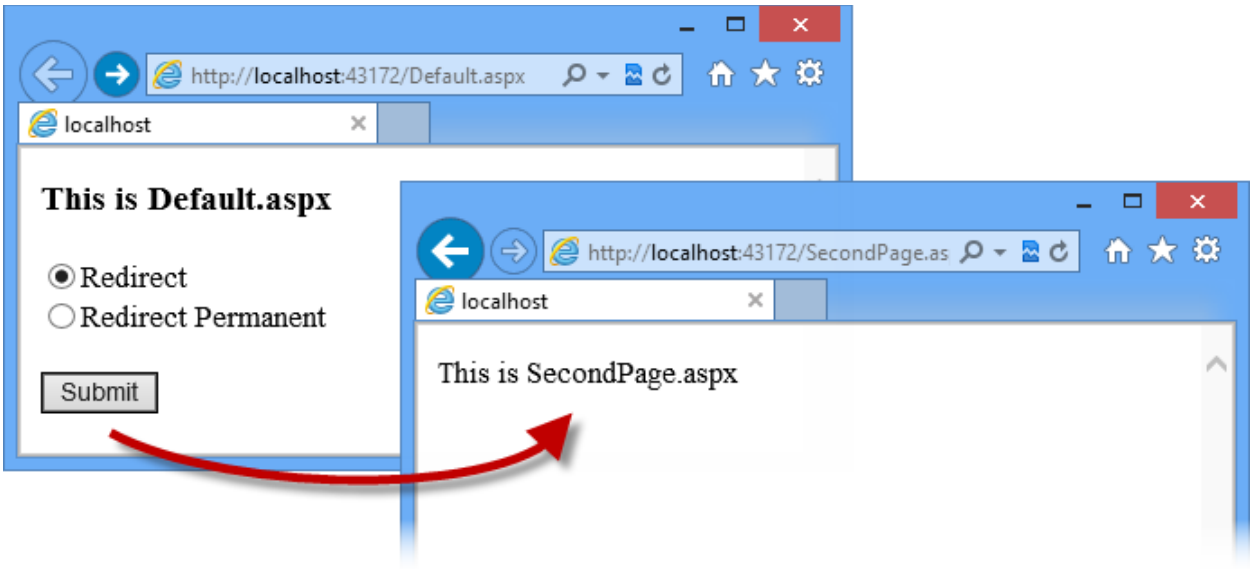
RedirectToRoutePermanent (name) - Trimite un răspuns cu un cod de stare 301 către o adresă URL generată dintr-o rută.

Efectuăm redirecționarea în fișierul de codul din fisierul Default.aspx.cs , așa cum se poate vedea în Exemplul 9.

Exemplul 9*.* Performing redirections in the Default.aspx file

```
using System;
namespace RequestControl {
        public partial class Default : System.Web.UI.Page {
            protected void Page_Load(object sender, EventArgs e) {
                if (IsPostBack) {
                        switch (Request.Form["choice"]) {
                            case "redirect302":
                                Response.Redirect("/SecondPage.aspx", false);
                                break;
                            case "redirect301":
                                Response.RedirectPermanent("/CurrentTimeHandler.ashx");
                                break;
                        }
                }
            }
        }
}
```

Folosim proprietatea IsPostBack pentru a vedea dacă avem de-a face cu o solicitare postBack. Apelăm apoi la metoda Redirect sau RedirectPermanent pentru a redirecționa cererea fie spre formularul Web SecondPage.aspx, fie spre handler-ul generic CurrentTimeHandler. Daca pornim aplicația, ar trebui sa arate ca mai jos.



Efectuarea manuală a redirecționarilor. Metodele Redirect și RedirectPermanent sunt modalități convenabile de a efectua redirectari, dar putem folosi proprietățile definite de clasa HttpResponse pentru a efectua indicații manual. În Exemplul 10, puteți vedea cum am actualizat clasa de cod din fișierul Default.aspx.cs pentru a efectua o redirecționare manuală.

Exemplul 10. Performing manual redirection in the Default.aspx.cs file

```
using System;
namespace RequestControl {
    public partial class Default : System.Web.UI.Page {
        protected void Page_Load(object sender, EventArgs e) {
            if (IsPostBack) {
                switch (Request.Form["choice"]) {
                    case "redirect302":
                        Response.Redirect("/SecondPage.aspx", false);
                        break;
                    case "redirect301":
                        //Response.RedirectPermanent("/CurrentTimeHandler.ashx");
                        Response.RedirectLocation = "/CurrentTimeHandler.ashx";
                        Response.StatusCode = 301;
                        Context.ApplicationInstance.CompleteRequest();
                        break;
                }
            }
        }
    }
}
```

Nu există nici un avantaj daca tratam redirecționarile in acest fel. Folosim proprietatea RedirectLocation pentru a specifica adresa URL la care dorim să redirecționăm browserul și utilizăm proprietatea StatusCode pentru a seta tipul de redirecționare. Nu dorim să fie procesată în continuare cererea după ce am stabilit valorile proprietății, de aceea apelăm la metoda HttpApplication.CompleteRequest pentru a încheia procesul de gestionare a cereri.

Gestionarea selectiei și execuției unui Handler. O modalitate alternativă de a gestiona fluxul de cereri este de a controla selecția și execuția handler-elor. Aceste tehnici se bazează pe membrii definiți de clasa HttpContext și HttpServerUtility. Am sintetizat, mai jos, metodele și proprietățile pe care le putem folosi.

Metode si proprietati HttpContext:

CurrentHandler- Returnează handler-ul către care a fost transferată cererea.

Handler- Returnează handler-ul selectat inițial pentru a genera un răspuns pentru cerere.

PreviousHandler - Returnează handler-ul de la care a fost transferată cererea.

RemapHandler (handler) - Previne procesul de selecție a handler-ului standard. Această metodă trebuie apelată înainte de declanșarea evenimentului MapRequestHandler.

Metode si proprietati HttpServerUtility:

Transfer (path) - Transferă solicitarea către handler pentru calea specificată. Formularul și datele șirului de interogare sunt transmise noului instrument de gestionare.

Transfer (path, preserve) - Transferă solicitarea către handler pentru calea specificată. Formularul și datele șirului de interogare sunt transmise noului handler dacă argumentul **preserve** este true.

Transfer (handler, preserve) - Transferă cererea la obiectul de gestionare specificat. Formularul și datele șirului de interogare sunt transmise noului handler dacă argumentul **preserve** este true.

Execute (path) - Generează un răspuns de la handler pentru calea specificată fără a încheia secvența normală de manipulare. Formularul și datele șirului de interogare din cereri sunt transmise gestionarului.

Execute (path, preserve) - Generează un răspuns de la handler pentru calea specificată fără a încheia secvența normală de manipulare. Formularul și datele de șir de interogare din solicitări sunt transmise gestionarului dacă argumentul **preserve** este true.

Execute (path, writer) - Generează un răspuns de la handler pentru calea specificată fără a încheia secvența normală de manipulare. Răspunsul este scris în obiectul TextWriter specificat. Formularul și datele șirului de interogare din cereri sunt transmise gestionarului.

Execute (path, writer, preserve) - Generează un răspuns de la handler pentru calea specificată fără a încheia secvența normală de manipulare. Răspunsul este scris în obiectul TextWriter specificat. Formularul și datele din șirul de interogare din solicitări sunt transmise gestionarului dacă argumentul preserve este true.

Execută (handler, writer, preserve) - Generează un răspuns de la handler-ul specificat fără a încheia secvența de manipulare normală. Răspunsul este scris în obiectul TextWriter specificat. Formularul și datele de șir de interogare din solicitări sunt transmise gestionarului dacă argumentul preserve este true.

Transferul unei solicitări. Putem Efectua un transfer intre pagini apelând metoda HttpServerUtility.Transfer. Putem trece printr-un obiect IHttpHandler sau pe o cale de fișier - dacă folosim o cale de fișier, atunci se utilizează tehnica implicită de selectare a handler-ului. În Exemplul 11, puteți vedea cum am folosit metoda Transfer în fișierul de cod Default.aspx.cs. Obținem un obiect HttpServerUtility prin proprietatea Page.Server.

Exemplul 11**.** Transferring a request in the Default.aspx.cs code-behind file

```
using System;
namespace RequestControl {
    public partial class Default : System.Web.UI.Page {
        protected void Page_Load(object sender, EventArgs e) {
        if (IsPostBack) {
            switch (Request.Form["choice"]) {
                case "redirect302":
                        Response.Redirect("/SecondPage.aspx", false);
                        break;

                case "redirect301":
                        //Response.RedirectPermanent("/CurrentTimeHandler.ashx");
                        Response.RedirectLocation = "/CurrentTimeHandler.ashx";
                        Response.StatusCode = 301;
                        Context.ApplicationInstance.CompleteRequest();
                        break;
                case "transferpage":
                        Server.Transfer("/SecondPage.aspx");
                        break;
                }
            }
        }
    }
```

Puteți obține informații despre manipulatoarele utilizate pentru procesarea unei solicitări folosind proprietățile Handler, CurrentHandler și PreviousHandler, toate definite de clasa HttpContext. Proprietatea Handler returnează obiectul IHttpHandler care a fost selectat initial.

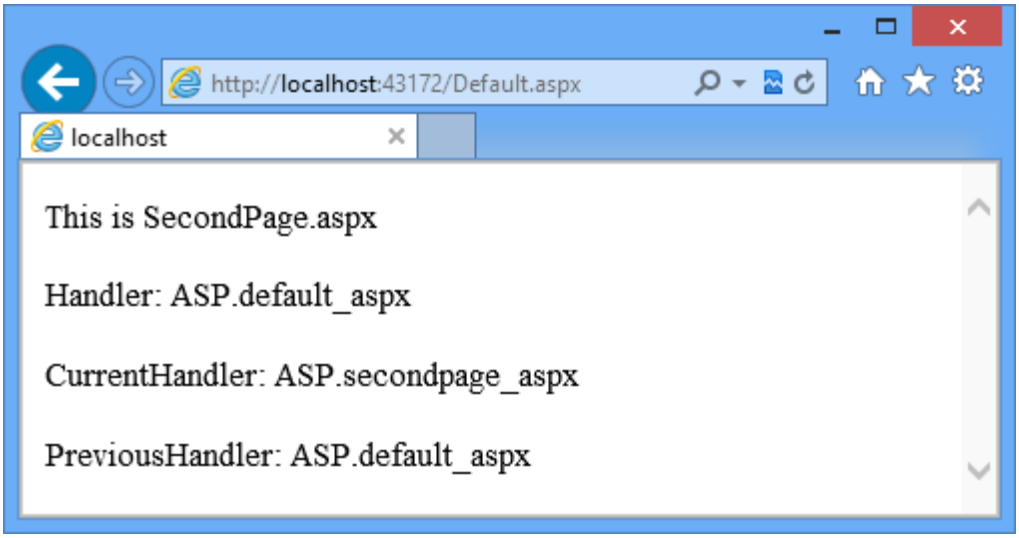
CurrentHandler returnează obiectul IHttpHandler la care a fost transferată cererea, iar PreviousHandler returnează IHttpHandler din care a fost transferată cererea.

Pentru a demonstra utilizarea acestor proprietăți, am adăugat câteva elemente de cod la Formularul web SecondPage.aspx, așa cum se arată în Exemplul 12.

Exemplul 12. Adding elements and code nuggets to the SecondPage.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeBehind="SecondPage.aspx.cs" Inherits="RequestControl.SecondPage" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
        <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        This is SecondPage.aspx
        <p>Handler: <%: Context.Handler %> </p>
        <p>CurrentHandler: <%: Context.CurrentHandler %> </p>
        <p>PreviousHandler: <%: Context.PreviousHandler %> </p>
    </div>
    </form>
</body>
</html>
```

Pentru a vedea efectul acestor completări, porniți aplicația, selectati radio butonul Transfer și faceți clic pe butonul Submit. Clasa de cod din spatele formularului web Default.aspx va efectua un transfer către SecondPage.aspx și va produce rezultatele prezentate în figura de mai jos.



Observați că adresa URL afișată de browser în figură este Default.aspx, chiar dacă conținutul este generat de SecondPage.aspx. Spre deosebire de o redirecționare HTTP, un transfer are loc în întregime pe server și nu este evident pentru browser (sau pentru utilizator).

Gestionarea informatiilor de stare

Am vazut modul în care ASP.NET gestionează cererile. Vom vedea in continuare care sunt mecanismele de gestionare a informatiilor de stare, care permite sistemului ASP.NET să creeze o aplicație cu stare dintr-o serie de solicitări HTTP fără stare, oferindu-ne capacitatea de a asocia împreună cererile conexe și de a stoca și regăsi datele de care avem nevoie pentru a crea continuitate pentru utilizator.

Pentru a ilustra aceste facilitate, am creat un nou proiect Visual Studio numit State folosind șablonul ASP.NET Empty Web Application și am adăugat un Formular web numit Default.aspx (fără a utiliza o pagină master). Puteți vedea conținutul fișierului Default.aspx în Exemplul 13.

Exemplul 13. The contents of the Default.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
        Inherits="State.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
        <title></title>
</head>
<body>
        <form id="form1" runat="server">
        <div>This page has been displayed <%: GetCounter() %> time(s).</div>
        </form>
</body>
</html>
```

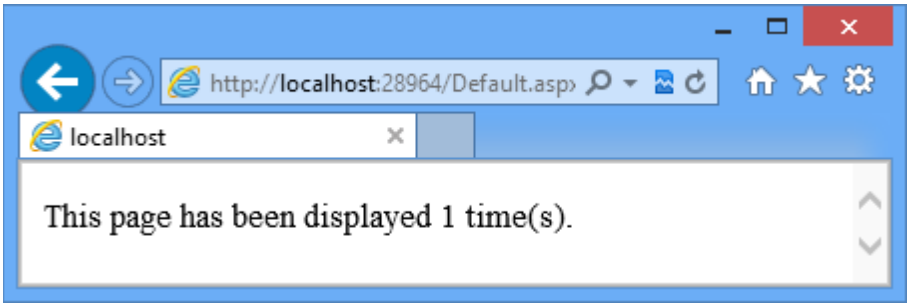
Acesta este un formular Web simplu care conține cod care apelează la metoda GetCounter din clasa cod-behind, definita în Exemplul 14.

Exemplul 14. The contents of the Default.aspx.cs file

```
using System;
namespace State {
    public partial class Default : System.Web.UI.Page {
        private int counter = 0;
        protected int GetCounter() {
                return ++counter;
        }
    }
}
```

Clasa code-behind definește un câmp int numit **counter**, care este incrementat de fiecare dată când se apelează metoda GetCounter. Ideea este să contorizam numărul de accesari ale paginii Default.aspx, dar, după cum veți afla, există o problemă în acest cod care tine direct de modul în care funcționează Web Forms.

Dacă pornim aplicația vom vedea un contor simplu care nu se incrementeaza, așa cum se vede în figura de mai jos.



Deoarece la fiecare solicitare, obținem o nouă instanță a handlerului nostru Web Form, obținem și o nouă variabilă contor a cărei valoare inițială este setată atunci când clasa este instanțiată și care este modificată doar o dată înainte ca obiectul să fie distrus.

Este destul de ușor să evitați problema cu starea pierdută și să stocați datele de stare în aplicația dvs. folosind funcțiile de gestionare a datelor de stare pe care le oferă ASP.NET Framework. Funcția specifică pe care ar trebui să o utilizați depinde de sfera dorită pentru datele dvs.: puteți stoca date care sunt partajate de toate cererile din aplicație (date despre aplicație), partajate de toate solicitările făcute de același utilizator (date de profil), partajate de toate cererile dintr-o singură sesiune (date de sesiune), partajate între două cereri simple (View State) sau partajate între cererile făcute de o fereastră a browserului (cookie-uri).

Informatiile de stare ale aplicatiei se fac la diverse nivele si anume, la nivel de aplicatie, la nivel de sesiune, la nivel de server sau la nivel de pagina. Aceste informatii sunt pastrate in obiecte specifice situate pe server, in pagina, in memoria browser-ului sau chiar pe calculatorul clientului.

Stocarea datelor de stare la nivel de aplicație. Când stocați datele aplicației, acestea sunt disponibile în întreaga aplicație indiferent de cererea procesată. În Exemplul 15, puteți vedea cum am actualizat codul Default.aspx.cs din spatele fișierului pentru a utiliza proprietatea Application pentru salvarea de date.

Exemplul 15**.** Updating the code-behind class to use application data

```
using System;
namespace State {
    public partial class Default : System.Web.UI.Page {
        protected int GetCounter() {
                Application.Lock();
```

```
            int result = (int)(Application["counter"] ?? 0);
            Application["counter"] = ++result;
            Application.UnLock();
            return result;
        }
    }
}
```

Accesăm funcția de date a aplicației se realizeaza prin proprietatea Application, care este moștenită de la clasa de bază System.Web.UI.Page sau de la proprietatea cu același nume a obiectului HttpContext. Proprietatea Application returnează un obiect System.Web.HttpApplicationState, pe care îl putem folosi pentru a stoca și recupera date.

Datele pe care le stocăm prin proprietatea Aplicației sunt păstrate în memorie și se pierd atunci când aplicația este oprită sau repornită. Obiectul HttpApplicationState stochează perechi cheie / valoare. Puteți stoca și recupera valorile de date utilizând un indexator în stil matricial, astfel:

```
...
Application["counter"] = ++result;
...
```

Când preluați o proprietate de date a aplicației, veți primi un obiect pe care trebuie să-l convertiți într-un tip pe care îl puteți utiliza, astfel:

```
...
int result = (int)(Application["counter"] ?? 0);
...
```

Clasa HttpApplicationState definește câțiva membri suplimentari care pot fi utili atunci când lucrați cu valorile datelor.

AllKeys - returnează un șir de șiruri care conține toate valorile cheie.

Count - returnează numărul de articole de date ale aplicației.

Clear() - Elimină toate elementele de date din starea aplicației.

Lock() - Serializează accesul la datele aplicației.

Remove(key) - Sterge elementul cu cheia specificată din starea aplicației.

UnLock() - Deblochează datele aplicației, astfel încât să poată fi efectuate actualizări simultane.

Stocarea datelor de stare la nivel de sesiune. Când facem o prima solicitare pentru Framework-ul ASP.NET, acesta creează o nouă sesiune și adaugă un cookie la răspuns.

Orice solicitare ulterioara pe care o facem in aceiasi sesiune conține acest cookie, permițând crearea de continuitate într-un set de solicitări HTTP fără stare. Datele sesiunii sunt accesibile tuturor solicitărilor într-o sesiune.

Datele sesiunii sunt accesate prin proprietatea Session mostenita din clasa de bază Page (sau din proprietatea HttpContext.Session). Proprietatea Session returnează un obiect System.Web.SessionState.HttpSessionState in care putem memora sau accesa valorile datelor de sesiune.

Exemplul 16. Ilustreaza modul de utilizare a datelor sesiunii.

Exemplul 16*.* Updating the Default.aspx Web Form to display profile and session data

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
        Inherits="State.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>

    <style>
        .nameContainer { margin: 10px 0;}
        input { margin-right: 10px; }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        This page has been displayed
        <%: GetSessionCounter() %> times(s) for this session.
    </div>
    <div class="nameContainer">
        <input id="requestedUser" value="Joe" runat="server" />
        <button type="submit">Submit</button>
    </div>
    </form>
</body>
</html>
```

Puteți vedea cum am implementat metoda **GetSessionCounter()** în fișierul Default.aspx.cs din Exemplul 16.

Exemplul 16*.* Updating the code-behind class to display profile and session data

```
using System;
using System.Web.Profile;
namespace State {
    public partial class Default : System.Web.UI.Page {
```

```
        protected void Page_Load(object sender, EventArgs e) {
        }
        protected int GetSessionCounter() {
                int counter = (int)(Session["counter"] ?? 0);
                Session["counter"] = ++counter;
                return counter;
        }
    }
}
```

Metoda GetSessionCounter, folosește proprietatea Session pentru a accesa obiectul HttpSessionState pentru a citi și scrie un contor.Puteți vedea cum funcționează funcția de stare a sesiunii pornind aplicația. Toate sesiunile au propriile lor date de sesiune.

Pe lângă faptul că ne permite să regăsim și să stocăm valorile stării sesiunii după nume, obiectul HttpSessionState returnat de proprietatea Session definește unii membri utili care pot ușura lucrul cu datele sesiunii.

Count- Returnează numărul de date ale sesiunii curente.

IsCookieLess- Cererile sunt asociate cu sesiuni, fie prin adăugarea unui cookie la cerere, fie prin adăugarea de informații la adresa URL a solicitării. Această proprietate returnează true atunci când este utilizată opțiunea URL de solicitare.

IsNewSession- Returnează true dacă aceasta este prima solicitare pentru o sesiune.

IsReadOnly- Returnează true dacă datele sesiunii sunt numai de citire, ceea ce se întâmplă atunci când atributul EnableSessionState din Directiva Web Form Page este setat pe ReadOnly.
Keys - returnează o colecție de chei pentru toate elementele de date ale stării sesiunii.

Mode - Returnează detalii despre modul în care datele de sesiune sunt stocate folosind o enumerare System.Web.SessionState.SessionStateMode.

SessionID - returnează ID-ul unic pentru sesiunea curentă.

Abandon() - Încheie sesiunea curentă. Orice solicitări suplimentare vor duce la crearea unei noi sesiuni.

Clear() - Elimină toate elementele de date din starea sesiunii curente.

Utilizarea View State. View data vă permite să stocați date între solicitările pentru același formular Web adăugând elemente HTML ascunse la răspunsul trimis în browser. View State poate fi utilă, dar trebuie utilizată cu atenție, deoarece adaugă date care sunt trimise clientului ca parte a răspunsului - și apoi trimise din nou la server, ca parte a următoarei solicitări.

În Exemplul 17, puteți vedea modificările pe care le-am făcut în fișierul Default.aspx Web Form pentru a ilustra utilitatea View State.

Exemplul 17. Simplifying the Default.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
        Inherits="State.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <style>
        button { margin: 10px 0; }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            This page has been displayed <%: GetCounter() %> time(s).
        </div>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

Stocarea se face folosind un element de input ascuns, ceea ce înseamnă că funcționează numai atunci când utilizatorul trimite un formular către server și când Formularul Web conține un element de formular HTML programabil (ceea ce înseamnă că atributul runat este setat pe server).

În Exemplul 18, puteți vedea modul în care am modificat fișierul din spatele codului Default.aspx.cs pentru a utiliza View State.

Exemplul 18. Using view state in the Default.aspx.cs code-behind file

```
using System;
namespace State {
    public partial class Default : System.Web.UI.Page {
        private int counter;
        protected void Page_Load(object src, EventArgs e) {
            counter = (int)(ViewState["counter"] ?? 0);
            ViewState["counter"] = ++counter;
        }

        protected int GetCounter() {
            return counter;
        }
    }
}
```

Structura clasei din spatele formularului Web este diferită de celelalte exemple din acest laborator, deoarece View State trebuie utilizat într-un mod particular. Framework-ul ASP.NET compilează formularele web în clase care utilizează fiecare control, element programabil și cod pe rând pentru a produce HTML. Problema cu care ne confruntăm este că datele de stare din View State sunt adăugate la răspunsul HTML imediat ce este atins elementul Web Form, adica inaintea procesarii evenimentelor client. Aceasta înseamnă că trebuie să ne asigurăm că am citit și actualizat valorile View State înainte de începerea redării HTML. O modalitate de a face acest lucru este să tratam aceste date pe evenimentul Load. Deoarece datele din View State sunt gestionate în metoda Page_Load, metoda noastră GetCounter trebuie doar să returneze valoarea contorului.

Accesăm funcția utilizeaza proprietatea ViewState care este moștenit de la clasa de bază Page și care returnează un obiect System.Web.UI.StateBag. Ca și în cazul celorlalte tipuri de management de stare, citim și scriem valorile datelor utilizând un indexator în stil cheie/valoare. Clasa StateBag definește unii membri suplimentari care pot fi utili atunci când lucrați cu date View State.

Count - Returnează numărul de articole de date de stare vizualizate.

Keys - Returnează o colecție de nume ale elementelor de date ale stării de vizualizare.

Values - Returnează o colecție de obiecte StateItem, fiecare reprezentând un element de date de stare de vizualizare.

Clear() - Elimină toate elementele de date din starea de vizualizare.

Remove(name) - Șterge elementul de date cu numele specificat din starea de vizualizare.

IsItemDirty (name) – Returnează true dacă elementul de date cu numele specificat a fost modificat.

Utilizarea cookie-urilor. Putem stoca date de stare folosind cookie-uri. În Exemplul 19, puteți vedea cum am actualizat fișierul Default.aspx.cs pentru a stoca contorul nostru de afișare folosind un cookie.

Exemplul 19*.* Using a cookie to store state data

```
using System;
using System.Web;
namespace State {
    public partial class Default : System.Web.UI.Page {
        private int counter;
        protected void Page_Load(object src, EventArgs e) {
                HttpCookie incomingCookie = Request.Cookies["counter"];
                counter = incomingCookie == null ? 0 : int.Parse(incomingCookie.Value);
                counter++;
                Response.Cookies.Add(new HttpCookie("counter", counter.ToString()));
```

```
        }
        protected int GetCounter() {
                return counter;
        }
    }
}
```

Accesăm cookie-urile pe care browserul le-a trimis o dată cu solicitarea folosind proprietatea Request.Cookies. Această proprietate returnează un obiect HttpCookieCollection care acceptă un indexator în stil matrice care ne permite să regăsim cookie-urile după nume - numele folosit pentru cookie-ul nostru este counter.

Clasa HttpCookieCollection definește urmatorii membrii.

Add(cookie) - Adaugă un nou cookie în colecție.

Clear() - Elimină toate cookie-urile.

CopyTo(array) - Copiază cookie-urile într-un tablou HttpCookie.

Count- Returnează numărul de cookie-uri din colecție.

Keys - returnează o colecție cu numele cookie-urilor.

Remove(name) Elimină cookie-ul cu numele specificat din colecție.

Domain - Obține și stabilește domeniul cu care este asociat cookie-ul.

Expires - Obține sau stabilește timpul de expirare pentru cookie.

HttpOnly - Obține sau setează dacă cookie-ul este accesibil printr-un apel JavaScript Ajax.

Name - Obține sau setează numele cookie-ului.

Secure - Obține sau setează dacă cookie-ul trebuie trimis doar prin conexiuni SSL.

Shareable - Obține sau setează dacă valoarea cookie-ului ar trebui să fie memorată în cache.

Value - Obține sau setează valoarea cookie-ului.

Mai intai verificăm dacă există un cookie numit counter asociat cu solicitarea. Dacă există, parsăm proprietatea Value:

```
...
HttpCookie incomingCookie = Request.Cookies["counter"];
counter = incomingCookie == null ? 0 : int.Parse(incomingCookie.Value);
...
```

Pentru a seta o valoare pentru un cookie, trebuie să accesăm HttpCookieCollection returnat de proprietatea Response.Cookies și să creăm un nou obiect HttpCookie pe care il adăugăm apoi la colecție cu metoda Add, astfel:

```
...
Response.Cookies.Add(new HttpCookie("counter", counter.ToString()));
...
```

Când vom crea un cookie ca parte a solicitării, vom vedea valoarea pe care o setăm în următoarea solicitare. Pentru contorul nostru, aceasta înseamnă că trebuie să avem mereu grijă să setam o nouă valoare, altfel vom primi cereri cu valori expirate.

Rutarea

În continuare, prezentăm funcția de rutare URL, care permite unei aplicații să suporte căi virtuale care nu sunt direct legate de fișierele din proiect. Vom vedea cum puteți defini, aplica, extinde și restricționa rutele. Pentru a ilustra aceste facilitati, am creat un nou proiect numit Rutare folosind șablonul proiectului de aplicație web ASP.NET gol. Pentru exemple, avem nevoie de câteva formulare web pe care le putem accesa cu adresele URL. Incepem prin crearea unui formular web numit Default.aspx, care este prezentat în Exemplul 20.

Exemplul 20. The contents of the Default.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeBehind="Default.aspx.cs" Inherits="Routing.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
        <title></title>
</head>
<body>
        <p>This is Default.aspx</p>
</body>
</html>
```

Trebuie să putem spune ce Formular Web a fost folosit pentru a genera un răspuns, deci marcajul conține numele fișierului. Următorul nostru pas este să creăm un folder numit Store, in care am adăugat un Formular Web numit Cart.aspx.

Conținutul acestui fișier este afișat în Exemplul 21.

Exemplul 21. The contents of the Store/Cart.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeBehind="Cart.aspx.cs" Inherits="Routing.Store.Cart" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
        <title></title>
</head>
<body>
        <p>This is /Store/Cart.aspx</p>
</body>
</html>
```

Nu am efectuat nicio modificare a modului în care căile sunt gestionate de aplicație, astfel încât pentru a accesa aceste Formularele Web, trebuie să solicităm adresele /Default.aspx și /Store/Cart.aspx.

Pregătirea aplicației pentru rutare. Convenția pentru utilizarea rutării este crearea unui folder App_Start și utilizarea acestuia pentru a crea un fișier de clasă numit RouteConfig.cs, care conține o metodă care configurează sistemul de rutare. Apoi apelăm la această metodă din clasa globala a aplicației Global.

Am adăugat un folder App_Start la proiect și am creat un fișier clasă numit RouteConfig.cs în el. Conținutul fișierului clasei este prezentat în Exemplul 22.

Exemplul 22*.* The contents of the App_Start/RouteConfig.cs class file

```
using System.Web.Routing;
//namespace Routing.App_Start {
namespace Routing {
        public class RouteConfig {
                public static void RegisterRoutes(RouteCollection routes) {
                }
        }
}
```

Metoda RegisterRoutes va conține instrucțiunile care configurează rutarea pentru aplicația noastră. Argumentul metodei este un obiect RouteCollection, care este definit în spațiul de nume System.Web.Routing.

Definim căile virtuale de care avem nevoie in aplicației adăugând rute la obiectul RouteCollection.

Trebuie să configurăm rutarea înainte ca aplicația să primească orice solicitare, ceea ce înseamnă că vom face acest lucru in metoda Application_Start a clasei globale a aplicatiei. Am adăugat o clasă globală la aplicație la proiect și am actualizat fișierul de clasă Global.asax.cs pentru a apela metoda RegisterRoutes, așa cum se arată în Ecemplul 23.

Ecemplul 23*.* The contents of the Global.asax.cs code-behind file

```
using System;
using System.Web.Routing;
```

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

Metoda noastră **Application_Start** apelează **RouteConfig.RegisterRoutes**, pentru valoarea returnată de proprietatea **RouteTable.Routes**. Clasa **RouteTable** oferă acces la setul de rute care a fost creat pentru aplicație prin proprietatea statica Routes .

Lucrul cu rutele fixe. Cel mai simplu mod de a crea o ruta este de a defini o nouă cale virtuală fixă care va viza un formular Web. Facem acest lucru folosind metoda RouteCollection.MapPageRoute, așa cum se arată în Exemplul 24.

Exemplul 24*.* Creating basic routes in the App_Start/RouteConfig.cs file

```
using System.Web.Routing;
namespace Routing {
        public class RouteConfig {
            public static void RegisterRoutes(RouteCollection routes) {
                routes.MapPageRoute("default", "", "~/Default.aspx");
                 routes.MapPageRoute("cart1", "cart", "~/Store/Cart.aspx");
                 routes.MapPageRoute("cart2", "apps/shopping/finish", "~/Store/Cart.aspx");
                }
        }
}
```

Fiecare apel al metodei MapPageRoute definește o noua ruta și fiecare rută mapează o nouă cale virtuală către un formular Web. Există mai multe versiuni ale metodei MapPageRoute, pe care le-am descris mai jos.

MapPageRoute(name, path, target)

MapPageRoute(name, path, target, checkAccess)

MapPageRoute(name, path, target,checkAccess, defaults)

MapPageRoute(name, path, target,checkAccess, defaults, constraints)

MapPageRoute(name, path, target, checkAccess, defaults,constraints, tokens)

Parametrii posibili ai metodei MapPageRoute au urmatoarea semnificatie:

name : Numele cu care se va manipula ruta

path : Calea virtuala URL

target : Fisierul fizic (Web Form)

checkAccess : dreptul de acces al utilizatorului (true sau false)

defaults : valorile implicite pentru segmentele variabile care nu sunt incluse în adresa URL

constraints : constrângerile care limitează segmentele variabile URL

tokens : specifică tokeni de date care sunt utilizate pentru de rutare

Declarațiile din exemplul 24 adaugă suport pentru trei noi căi virtuale care sunt dirijate către cele două Formulare Web ale noastre, așa cum este rezumat mai jos.

path	**target**
/ (the root URL)	**/Default.aspx**
/cart	**/Store/Cart.aspx**
/apps/shopping/finish	**/Store/Cart.aspx**

Aceste rute vor corespunde numai cererilor pentru adrese URL care se potrivesc cu traseele pe care le-am definit exact și, din acest motiv, sunt cunoscute sub numele de rute fixe.

De exemplu o ruta definite astfel:

routes.MapPageRoute("cart2","apps/shopping/finish",…)

se potriveste **cu folderol:** /apps/shopping/finish

pentru ca are trei segmente, commune , cu folderol : /apps/shopping

nu se potriveste pentru ca are prea puține segmente, cu folderol:/apps/shopping/finish/cart

nu se potriveste pentru ca are prea multe segmente iar cu folderul : /app/shopping/checkout

nu se potriveste pentru ca desi are număr corect de segmente, unul nu se potrivește.

Informații despre Rute. Adesea este util să obținem informații despre modul în care sistemul de rutare a procesat o solicitarea. Accesăm informațiile de rutare prin proprietatea RouteData a clasei Page. Proprietatea RouteData a clasei Page returnează obiectul System.Web.Routing.RouteData, care oferă aceste informații.

Proprietățile clasei RouteData:

DataTokens- Returnează o colecție de perechi cheie/valoare asociate rutei.

Route- Returnează un obiect **RouteBase** care poate fi utilizat pentru a obține informații despre ruta aplicată la solicitare.

RouteHandler-Returnează implementarea IRouteHandler care s-a potrivit cu solicitarea către un IHttpHandler.

Values- Returnează un set de valori ale parametrilor pentru rută.

Route- Returnează un obiect **RouteBase** care poate fi utilizat pentru a obține informații despre ruta aplicată la solicitare.

Exemplul 25. Utilizarea informatiilor din rute

```
using System.Web.Routing;

namespace Routing.Store {
    public partial class Cart : System.Web.UI.Page {
        protected string GetURLFromRoute() {
        Route myRoute = RouteData.Route as Route;
                if (myRoute != null) {
                        return myRoute.Url;
                } else {
                        return "Unknown RouteBase";
                }
        }
    }
}
```

Daca adăugam in fișierul /Store/Cart.aspx o metoda numită GetURLFromRoute:

Exemplul 26. Fișierul /Store/Cart.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeBehind="Cart.aspx.cs" Inherits="Routing.Store.Cart" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
        <title></title>
</head>
```

<body>

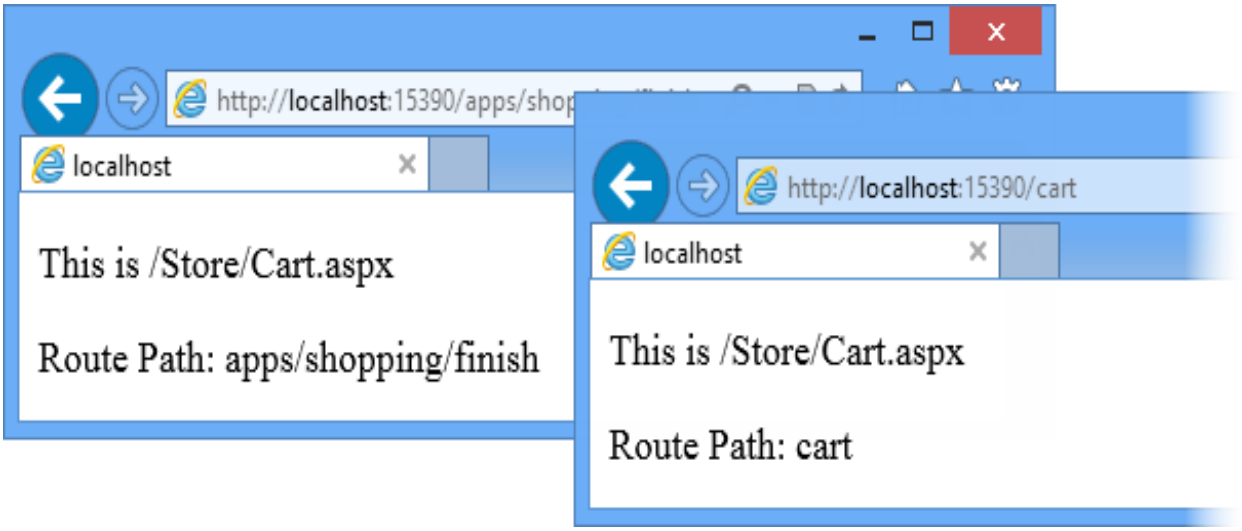
** This is /Store/Cart.aspx **

** Route Path: <%: GetURLFromRoute() %> **

</body>

</html>

Rezulta ceva de tipul de mai jos:



Clasa Route definește cateva proprietăți:

Constraints- Returneaza sau setează constrângerile utilizate pentru a limita intervalul de adrese URL.

DataTokens -Returneaza sau setează token-urile de date asociate rutei (sub forma cheie/valoare).

Defaults- Returneaza sau setează valorile implicite pentru segmente variabile.

RouteExistingFiles- Returneaza sau seteaza dacă sistemul de rutare trebuie să gestioneze solicitările care se potrivesc cu fișierele existente.

RouteHandler- Returneaza sau setează implementarea IRouteHandler asociată rutei.

Url- Returneaza sau seteaza calea folosită de rută.

Rute cu segmente variabile. Rutele fixe sunt utile, dar trebuie să creăm multe dacă avem o serie de adrese URL similare care vizează același formular web. Rutele pot contine segmente variabile care se noteaza cu un nume intre acolade ({și}).

De exemplu ruta

routes.MapPageRoute("dall", "{app}/default", "~/Default.aspx");

se potriveste cu:

/billing/default

/accounts/default

/payments/default

dar nu se potriveste cu:

/apps/shopping

/apps/shopping/finish/cart

Daca avem rutele:

routes.MapPageRoute("dall", "{app}/default", "~/Default.aspx");

routes.MapPageRoute("d4", "store/default", "~/Store/Cart.aspx");

definite in aceasta ordine atunci definitia a doua nu va fi folosita niciodata pentru ca este cuprinsa in prima definitie. Potrivirile sunt tratate in ordinea in care sunt definite. Exista mai multe metode pentru rezolvarea acestei problem dar cea mai simpla metoda este ordonarea corespunzatoare a definitiilor. In cazul nostrum:

routes.MapPageRoute("d4", "store/default", "~/Store/Cart.aspx");

routes.MapPageRoute("dall", "{app}/default", "~/Default.aspx");

O alta rezolvare utilizeaza Constrangeri de rutare.

Constrangeri de rutare. Putem limita gama de adrese URL pe care o rută se va potrivi prin aplicarea constrângerilor. Folosim clasa **MapPageRoute** care ne permite să furnizăm un obiect **RouteValueDictionary**, pe care îl folosim pentru a specifica **expresii regulate** care limitează valorile cu un segment variabil. De exemplu:

routes.MapPageRoute("dall", "{app}/default", "~/Default.aspx", false, null,

new RouteValueDictionary{{ "app", "accounts|billing|payments"}});

Aceasta definitie utilizeaza o expresie regulată simplă care se potrivește cu trei valori explicite, **"accounts|billing|payments"** dar putem utiliza orice expresie regulata pentru a limita potrivirile.

Exemplul 27. Utilizarea restrictiilor in definirea rutelor

namespace Routing {

public class RouteConfig {

```
        public static void RegisterRoutes(RouteCollection routes) {
                routes.MapPageRoute("default", "", "~/Default.aspx");
                RouteValueDictionary constraints = new RouteValueDictionary {
                    {"first", "[0-9]*"},{ "second", "[0-9]*"}, { "operation", "plus|minus"}
        };
        routes.MapPageRoute("calc", "calc/{first}/{operation}/{second}",
            "~/Calc.aspx", false, null, constraints);
        }
    }
}
```

Definește o ruta cu trei segmente variabile-first, operation, si second cu restrictiile definite de **constraints** :

first și second se potrivste doar cu valori întregi, **"[0-9]*"**

operation se va potrivi doar cu plus sau minus, **"plus|minus“**

```
routes.MapPageRoute("calc", "calc/{first}/{operation}/{second}",
    "~/Calc.aspx", false, null, constraints);
```

Exemplul 28. Extragerea informatiilor din ruta "calc“:

```
protected void Page_Load(object sender, EventArgs e) {
        int firstNumber = 0, secondNumber = 0;
        string firstString, secondString, operationString;
        if (RouteData.Values.Count > 0) {
                firstString = RouteData.Values["first"].ToString();
                secondString = RouteData.Values["second"].ToString();
                operationString = RouteData.Values["operation"].ToString();
        } else {
                firstString = Request["first"];
                secondString = Request["second"];
```

```
            operationString = Request["operation"];
        }
}
```

Rute de lungime variabila. Putem crea mai multă flexibilitate folosind un numar variabil de segmente , care ne permit să potrivim adresele URL de lungimi arbitrare.

Exemplul 29. Exemplu de ruta cu un numar variabil de segmente

```
using System.Web.Routing;
namespace Routing {
    public class RouteConfig {
        // ...other routes omitted for brevity...
        routes.MapPageRoute("calc3", "calc/{operation}/{*numbers}", "~/Calc.aspx");
    }
}
```

Un segment prefixat de un asterisc (*) este de lungime variabila. Acest lucru permite rutei noastre să corespundă oricărui URL care are două sau mai multe segmente si în care primul segment este calc.

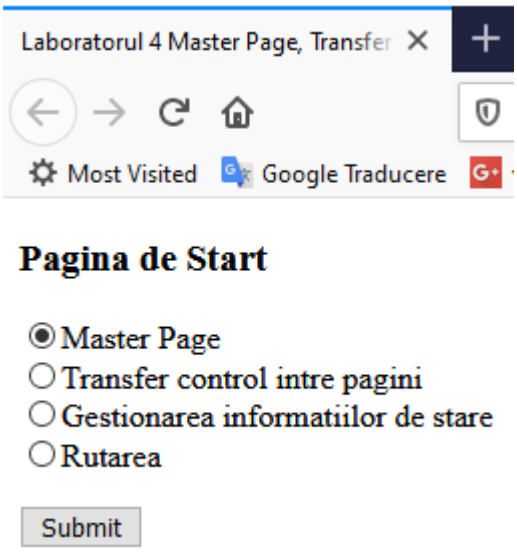
Exemplul 30.Extragerea informatiilor din ruta cu un numar variabil de segmente "calc3“:

```
int firstNumber = 0, secondNumber = 0;
string firstString, secondString, operationString;
if (RouteData.Values.Count > 0) {
    if (RouteData.Values.ContainsKey("numbers")) {
        string[] elems = RouteData.Values["numbers"].ToString().Split('/');
        firstString = elems[0];
        secondString = elems[1];
    }
}
```

Segmentele cu lungime variabilă se vor potrivi cu orice număr de segmente URL. Într-un proiect real, trebuie să acordam atenție validarilor in acest caz.

Exercitii.

1. **Studiati si testati exemplele din textul de mai sus.**
2. **Sa se creeze o noua aplicatie web ASP.NET care sa contina toate exemplele din aceasta lucrare si este structurata astfel:**
   - ✓ **Contine o pagina Web Form de start StartPage.aspx pe care avem un radio buton cu patru check-uri ca mai jos:**



   - ✓ **La selectia Master Page, daca actionam butonului Submit sa contina constructiile din Exemplele 1-5.**
   - ✓ **La selectia Transfer control intre pagini, daca actionam butonului Submit sa contina constructiile din Exemplele 6-12. Paginile Default.aspx si SecondPage.aspx vor fi continut pentru o pagina Master Page comuna Transfer.Master.**
   - ✓ **La selectia Gestionarea informatiilor de stare, daca actionam butonului Submit sa contina constructiile din Exemplele 13-19. Pagina Default.aspx din Exemplele 13-19 va fi redenumita DefaultState.aspx pentru a nu exista ambiguitate.**
   - ✓ **La selectia Rutarea daca actionam butonului Submit sa contina constructiile din Exemplele 20-30.**