საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ირაკლი ბულია,
ეკატერინე თურძია

Web-აპლიკაციების აგება ASP.NET & C# პაკეტების საფუძველზე

(ლაბორატორიული პრაქტიკუმის
დამხმარე სახელმძღვანელო)

დამტკიცებულია:
სტუ-ს სარედაქციო-
საგამომცემლო
საბჭოს მიერ

თბილისი
2009

უაკ 681.3.06

ლაბორატორიული პრაქტიკუმის მეთოდურ სახელმძღვანელოში გადმოცემულია ინტერნეტისთვის Web-აპლიკაციების ასაგებად ASP.NET ინსტრუმენტის სერვის-ფუნქციების გამოყენების და მართვის საკითხები C#.NET ინტეგრალური პროგრამული პაკეტით, აგრეთვე ADO.NET მონაცემთა ბაზის დრაივერით.

განკუთვნილია მართვის საინფორმაციო სისტემების (Management Information Systems) სპეციალობის მაგისტრანტებისა და გამოყენებითი ინფორმატიკის სპეციალისტებისათვის ვებ-დეველოპინგის სფეროში.

რეცენზენტი: ასოც. პროფ. გ. ღვინეფაძე

პროფ. გ. სურგულაძის რედაქციით

ISBN 978-9941-14-365-6

სარჩევი

შესავალი

ASP.NET (Active Server Pages – აქტიური სერვერული გვერდები) – არის NET-პლატფორმის ნაწილი და ტექნოლოგია, რომელიც დინამიკურად ქმნის დოკუმენტებს Web-სერვერზე, როცა ისინი მოითხოვება HTTP-ს საშუალებით.

ASP.NET ტექნოლოგია ანალოგიურია PHP, ColdFusion და სხვ. ტექნოლოგიების, მაგრამ მათ შორის მნიშვნელოვანი განსხვავებაცაა. ASP.NET, როგორც მისი დასახელება გვიჩვენებს, შეიქმნა სპეციალურად NET პლატფორმასთან სრული ინტეგრაციის მიზნით, რომლის ნაწილიც ითვალისწინებს C# ენის მხარდაჭერას.

როგორც ცნობილია, Web-გვერდების დასაპროგრამებლად გამოიყენება ისეთი სცენარების ენები, როგორიცაა VBScript ან JScript. ეს სკრიპტული ენები მუშაობდა, მაგრამ ხშირად გარკვეულ პრობლემებს უქმნიდა დაპროგრამების „ნამდვილი" ენების პროგრამისტებს სხვადასხვა ადმინისტრატორული დავალებათა შესრულებისას, რაც საბოლოო ჯამში აისახებოდა სისტემის მწარმოებლურობის დაქვეითებაში.

მაღალგანვითარებული ენების გამოყენების შემთხვევაში, მაგალითად, შესაძლებელია მუშაობის პროცესის უზრუნველყოფა სრული სერვერული ობიექტური მოდელით. ASP.NET ახორციელებს მიმართვას გვერდის ყველა მმართველ ელემენტთან, როგორც ზოგადად გარემოს ობიექტებთან. სერვერის მხარესაც კი ხორციელდება წვდომა .NET–ის ყველა საჭირო კლასთან.

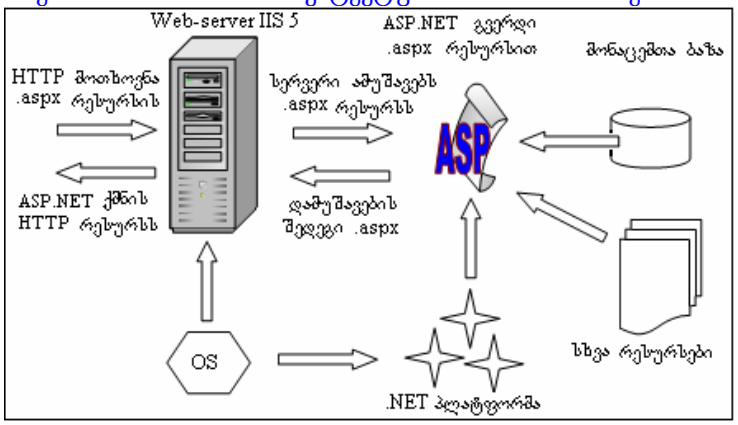
გვერდების მართვის ელემენტები ფუნქციონალურია და ფაქტობრივად, შესაძლებელია ყველაფრის გაკეთება, რასაც Windows-ის ფორმის კლასებთან ვაკეთებდით, რაც უფრო მოქნილს ხდის სისტემას. ამის გამო ASP.NET-ის გვერდებს, რომლებიც ქმნის HTML შედგენილობას, ხშირად უწოდებენ Web-ფორმებს.

წინამდებარე წიგნში ჩვენი მიზანია უფრო ღრმად გავერკვეთ ASP.NET–ში, ანუ როგორ მუშაობს იგი, რისი გაკეთება შეგვიძლია ჩვენ, სად და როგორ გამოვიყენოთ C#.NET პაკეტი.

I თავი
თეორიული საფუძვლები

1.1. ASP.NET–ის საბაზო არქიტექტურა

ASP.NET გამოიყენებს ინტერნეტის ინფორმაციულ სერვერს (IIS – Internet Information Server) HTTP მოთხოვნებზე საპასუხო შინაარსის მისაწოდებლად. ASP.NET გვერდები მოთავსებულია ფაილებში .aspx გაფართოებით. მისი საბაზო არქიტექტურა იხ. 1.1 ნახაზზე.



ნახ.1.1. საბაზო არქიტექტურა

ASP.NET-ის დამუშავების დროს მისაწვდომია .NET-ის ყველა კლასი, სპეციალური კომპონენტები, შექმნილი C# ან სხვა ენებზე, მონაცემთა ბაზები და ა.შ. ფაქტობრივად, სახეზეა ყველა ის შესაძლებლობა, რომელსაც იყენებს C# დანართის აგებისას. ე.ი., C#-ის გამოყენება ASP.NET-ში ეფექტურს ხდის დანართის შესრულებას.

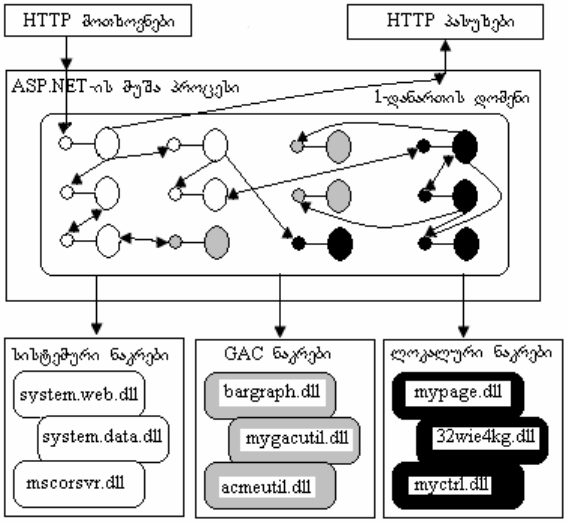
ASP.NET ფაილი შეიძლება შეიცავდეს ნებისმიერ ელემენტს შემდეგი სიიდან:

- ინსტრუქციის დამუშავება სერვერისთვის;
- კოდები ენებზე: C#, VB.NET, Jscript.NET ან სხვ., რომელთა მხარდაჭერა ხდება .NET პლატფორმით;
- შინაარსი ნებისმიერი ფორმით, რომელიც გენერირდება რესურსის სახით HTML–ით;
- ASP.NET -ის ჩადგმული სერვერული მართვის ელემენტები.

ამგვარად, შესაძლებელია ASP.NET ფაილის არსებობა, რომელიც მხოლოდ ერთი სტრიქონისგან შედგება, მაგალითად, Hello: , ყოველგვარი სხვა კოდის ან ინსტრუქციის გარეშე. ამის საფუძველზე იქმნება დასაბრუნებელი HTML გვერდი, რომელშიც მოთავსებულია აღნიშნული ტექსტი.

ამგვარად, ASP.NET-ის ბირთვია .NET-ის კლასების ერთობლიობა, რომელიც ემსახურება HTTP მოთხოვნების დამუშავებას. ზოგიერთი მათგანი ამ კლასებიდან განსაზღვრულია სისტემურ ნაკრებში (Assembly), როგორც .NET-ის საბაზო კლასების ბიბლიოთეკის ნაწილი. ზოგი შეიძლება მოთავსებული იყოს გლობალური კეშის ნაკრებში (GAC – Global Assembly Cache), ზოგიც შეიძლება ჩაიტვირთოს ლოკალური ნაკრებიდან, რომელიც განთავსებულია დანართის ვირტუალურ კატალოგში. ყველა კლასი, რომლებიც ემსახურება HTTP მოთხოვნებს, ჩაიტვირთება დანართის დომენში (დანართის არე – Application area) ASP.NET-ის მუშა პროცესში, და ურთიერთქმედებს ერთმანეთთან, აფორმირებს მოთხოვნებზე პასუხებს.

1.2 ნახაზი ასახავს ASP.NET-ის საბაზო არქიტექტურას დონეების მიხედვით. იგი 1.1 ნახაზის დეტალურად ასახსნელადაა შემოტანილი.



ნახ.1.2. საბაზო არქიტექტურა დონეების მიხედვით

ASP.NET-ის ძირითადი სიახლე მის წინამორბედებთან შედარებით ისაა, რომ აქ ყველა არსი აისახება კლასის საშუალებით, რომელიც ნაკრებიდან ჩაიტვირთება.

დანართების აგება ASP.NET-ში ხდება კლასების კონსტრუირებით, რომლებიც ურთიერთმოქმედებს სხვა კლასებთან. ზოგი კლასი იწარმოება პლატფორმის საბაზო კლასებიდან, ზოგს შეუძლია ინტერფეისების რეალიზება ამ პლატფორმაში, ზოგიც ურთიერთქმედებს პლატფორმის საბაზო კლასებთან მათი მეთოდების გამოძახების გზით.

ASP.NET-ის კლასიკური სინტაკსი ისევ შენარჩუნებულია, ოღონდაც მისი კოდები, რომლებიც ინახება ნაკრების ფაილებში სერვერის მხარეს, გარდაქმნილია კლასების განსაზღვრების სახით.

1.2. ASP გვერდის კოდის მაგალითები

ქვემოთ, 1.1 ლისტინგში მოცემულია ტრადიციული ASP გვერდის მარტივი მაგალითი, რომელშიც JavaScript -ის სერვერული კოდი შერწყმულია HTML–ის სტატიკურ კოდთან. ამ მაგალითში ნაჩვენებია ერთდროულად რამდენიმე კოდირების მეთოდიკა სერვერის მხარეს. კოდში მოთავსებულია სცენარის ბლოკი, რომელიც runat=server ატრიბუტითაა წარმოდგენილი და რომელიც შეიცავს Add( ) ფუნქციას.

ლისტინგი 1.1: ASP გვერდის მაგალითი

```
< ! --  faili test.asp   -- >
<%@  language =’JScript’  %>

<script  language=’JScript’  runat=server>
function Add(x, y)
{
 return x + y;
}
</script>
<html> <body>
 <h1> ASP-is testirebis gverdi </h1>

 <h2> 2+2=<%=Add(2,2) %> </h2>
 <table border=2>
 <%
```

```
   for (var i=0; i<10; i++)  {
 %>
    <tr><td> striqoni <%=i%> sveti-0 </td>
          <td> striqoni <%=i%> sveti-1 </td> </tr>
 <%
      }
 %>
 </table>

 <%
  Response.Write("<h2> Cawerilia uSualod Response-Si </h2>" );
%>
</body> </html>
```

სერვერის მხარეზე სინტაკსი "<%=" გამოიყენება Add() მეთოდის გამოსაძახებლად და შედეგის გამოსატანად h2 ელემენტში, ხოლო სცენარის სინტაკსი სერვერის მხარეზე "<% " კი ცხრილის 10 სტრიქონის პროგრამულად გენერირებისათვის. ჩადგმული ობიექტი Response გამოიყენება ამ სტრიქონების ბოლოში h2 ელემენტის პროგრამულად დასამატებლად.

ASP.NET-ში ფაილი.asp მიიღებს ფაილი.aspx ტიპს, ხოლო კოდის ტექსტი და შედეგები იქნება მსგავსი. ASP.NET- ში JScript-ის ენასთან ერთად გამოიყენება სხვა ენებიც, მაგალითად, C#.NET, VB.NET. 1.2 ლისტინგში ნაჩვენებია C#-ის ვარიანტი:

ლისტინგი 1.2: aspx გვერდის მაგალითი C#-ის და Page დირექტივის გამოყენებით

```
< ! --  faili test.aspx   -- >
<%@  Page language ='C# '  %>

<script  runat=server>
function Add(int x, int y)
{
  return x + y;
}
</script>
<html> <body>
<h1> ASP.NET-is testirebis gverdi </h1>
```

```
<h2> 2+2=<%=Add(2,2) %> </h2>
<table border=2>
<%
  for (var i=0; i<10; i++)  {
%>
   <tr><td> striqoni <%=i%> sveti-0 </td>
        <td> striqoni <%=i%> sveti-1 </td> </tr>
<%
     }
%>
</table>

<%
   Response.Write("<h2> Cawerilia uSualod Response-Si </h2>" );
%>
</body>
</html>
```

დირექტივაში Page უფრო ხშირად ათავსებენ გვერდის დონის ატრიბუტებს, რომლებითაც იმართება ASP.NET გვერდების გამოტანა.

ტრადიციული ASP (ლისტინგი 1.1 .asp) და ახალი ASP.NET (ლისტინგი 1.2 .aspx) შედარებისას განსხვავება მათი მუშაობის მწარმოებლურობაშია. ASP არის ინტერპრეტატორი და ყოველი ახალი გაშვებისას მისი JScript და HTML პროგრამები ინტერპრეტირდება მუშა ფაილებში, რაც მოითხოვს დროს. ASP.NET კი კომპილატორია, იგი ერთხელ (პირველი შესრულების დროს) ქმნის მუშა ფაილებს და შემდეგ ინახავს მას კომპილირებული სახით .NET–ის კლასებში. კომპილირებული კლასი შეიცავს როგორც სერვერული სცენარების კოდებს, ასევე სტატიკურ HTML კოდებს. შემდეგი გაშვებისას ეს კოდები იტვირთება მუშა პროგრამების სახით, რითაც უზრუნველყოფილია მაღალმწარმოებლურობა.

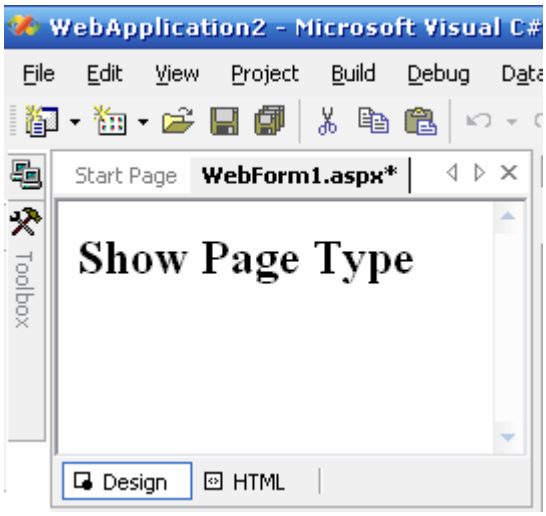
ამგვარად, შეიძლება ვთქვათ, რომ ყოველთვის, როდესაც იქმნება ASP.NET გვერდი, მაშინ იქმნება ახალი კლასი.

1.3. Web.UI.Page კლასი

როგორც ავღნიშნეთ, ყოველი გვერდი კომპილირდება კლასის განსაზღვრებაში. ამიტომაც საინტერესოა, თუ როგორ იქმნება კლასი და როგორ იმართება ეს პროცესი.

ჩვენი პირველი მაგალითისთვის გამოვიტანოთ გვერდის ტიპი და მისი საბაზო ტიპი. 1.3 ნახაზზე ნაჩვენებია .aspx ფაილი და მისი გამოტანა ბროუზერით. გვერდის ტიპი და მისი საბაზო ტიპი დამოტანილია GetType( ) მეთოდით და BaseType თვისებით.

```
< ! --  faili ShowPageType.aspx   -- >
<%@  Page language =’C# ’  %>
<html> <body>

<h2> Show Page Type  </h2>

<%
   Response.Output.Write(“<p>Page type {0} </p>”, this.GetType());
   Response.Output.Write(“<p>Page base type {0} </p>”,
this.GetType().BaseType);
%>
</body>
</html>
```



ნახ.1.3. .aspx-ფაილი და შედეგი VS-ბროუზერში

1.4. ASP.NET-ში მდგომარეობათა მართვა

ASP.NET-ის გვერდების დამახასიათებელი თვისებაა ის, რომ მათ არ გააჩნია მდგმარეობა. ჩვეულებრივად არავითარი ინფორმაცია არ ინახება სერვერზე მომხმარებელთა მოთხოვნებს შორის (თუმცა არსებობს მეთოდები, რომელთაც საჭიროების შემთხვევაში ამის გაკეთება შეუძლია).

ეს საკითხი, თავიდან უცნაურად ჩანს, რადგან მდგომარეობების მართვა ინტერაქტიული სეანსებისთვის მეტად მნიშვნელოვანია მომხმარებლისთვის. ამ თვალსაზრისით ASP.NET გვთავაზობს მეტად მოხერხებულ საშუალებას ასეთი პრობლემის გვერდის ასავლელად და სეანსების მართვის თითქმის გამჭვირვალე პროცესის განსახორციელებლად.

არსებითად, ინფორმაცია Web-ფორმაზე მართვის ელემენტების მდგომარეობათა შესახებ (მონაცემები, შეტანილი ტექსტურ ველებში, ამონარჩევი კომბობოქსებიდან და ა.შ.) ინახება მდგომარეობათა ასახვის დამალულ ველებში (viewstate), რომლებიც გვერდის შემადგენელი ნაწილია. იგი გენერირდება სერვერის მიერ და გადაეცემა მომხმარებელს.

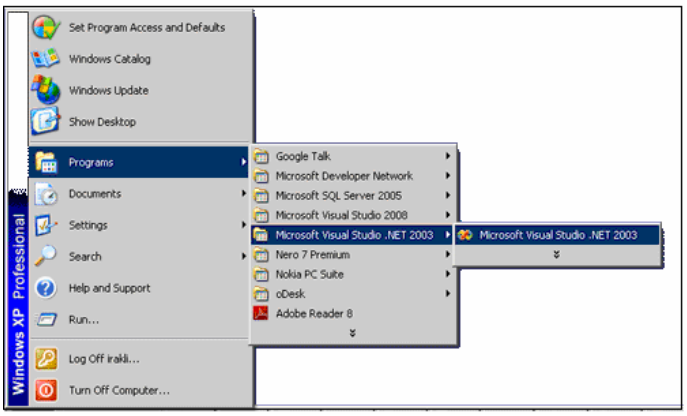
თუ შემდგომში საჭირო იქნება ფორმის მონაცემთა გადაგზავნის ტიპის სერვერული დამუშავება, მაშინ ხდება ამ ინფორმაციის დაბრუნება (postback) სერვერზე. აქ ეს ინფორმაცია გამოიყენება გვერდის ობიექტური მოდელის ხელმეორედ შესავსებად, ლოკალური ცვლილებების ხელით განხორციელების შესაძლებლობით (შემდგომში ამ საკითხს პრაქტიკულად შევეხებით).

II თავი: პრაქტიკული: მარტივი ელემენტები

2.1. ASP.NET აპლიკაციის შექმნის ეტაპები

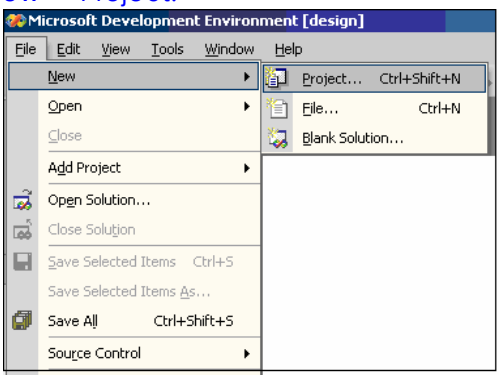
ASP.NET აპლიკაციის შესაქმნელად საჭიროა შემდეგი საფეხურების შესრულება:

1. პროგრამების პანელიდან ავირჩიოთ:
   Start->Programs->Microsoft Visual Studio .NET 2003



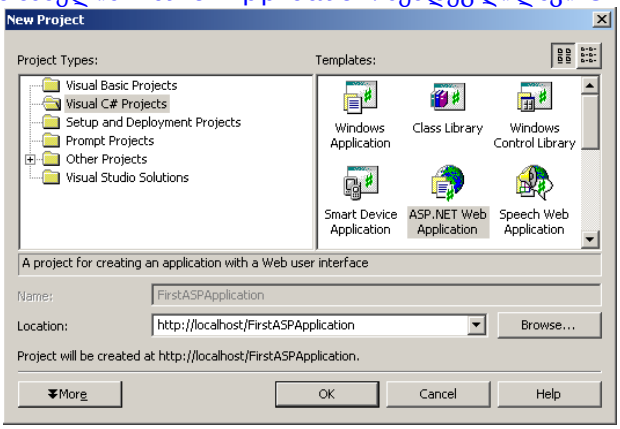
ნახ.2.1

2. პროგრამის გაშვების შემდეგ აირჩიეთ მენიუს პუნქტი:
   File -> New -> Project.



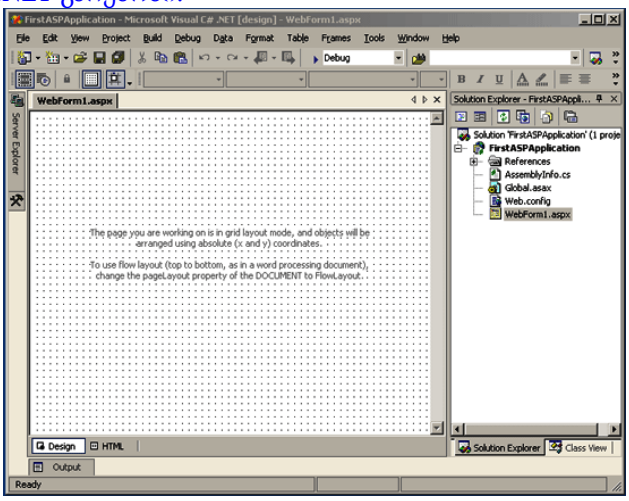
ნახ.2.2

3. გაიხსნება ახალი პროექტების ტიპის არჩევის ფანჯარა. Project Types ფანჯარაში აირჩიეთ Visual C# Project, ხოლო Templates ფანჯარაში ASP.NET Web Application. Location ველში http://localhost/ შემდეგ აკრიფეთ აპლიკაციის სახელი, ამ შემთხვევაში აპლიკაციის სახელია FirstASPApplication. შემდეგ ღილაკი OK.



ნახ.2.3

4. Visual Studio შექმნის აპლიკაციას და გახსნის Microsoft Visual C#.NET გარემოში.



ნახ.2.4

შექმნილ პროექტს შექმისთანავე შეიცავს რამდენიმე ფაილს:

- AssemblyInfo.cs
- Global.asax
- Web.config
- WebForm1.aspx და WebForm1.aspx.cs

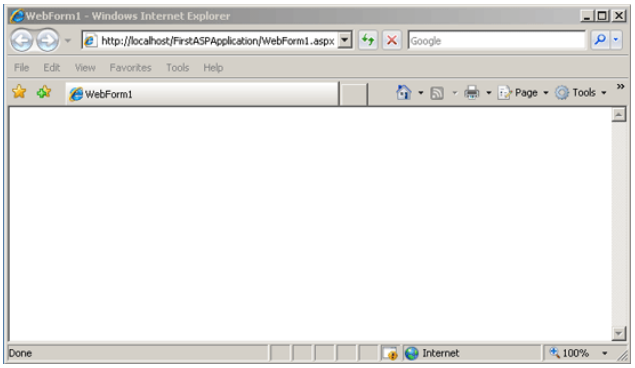
AssemblyInfo.cs არის შეიცავს ინფორმაციას აპლიკაციის შესახებ, როგორიცაა აპლიკაციის სახელი, ავტორი, ვერსია და სხვა.

Global.asax ფაილი ემსახურება აპლიკაციის დონის მოვლენების დამუშავებას, როგორიცაა შეცდომების დაჭერა, ახალი სესიის შექმნა, სესიის დასრულება და სხვა.

Web.config არის XML ფაილი რომელიც შეიცავს აპლიკაციის კონფიგურაციის მონაცემებს: სესიის პარამეტრები, მონაცემთა ბაზასთან კავშირის პარამეტრები, მომხმარებლების ავტორიზაციასა და აუთენთიფიკაციის კონფიგურაციის პარამეტრები.

WebForm1.aspx და WebForm1.aspx.cs ქმნიან ერთ ვებ-გვერდს. WebForm1.aspx ფაილი შეიცავს ვიზუალურ ელემენტებს, ხოლო WebForm1.aspx.cs ვებ-ფორმის კლასის მოვლენების დამუშავების მეთოდებს და ბიზნეს ლოგიკას.

5. შექმნილი ვებ-გვერდის ნახვა შესაძლებელია CTRL+F5 დაჭერით ან მენიუს პუნქტი Debug -> Start Without Debugging არჩევით:



ნახ.2.5

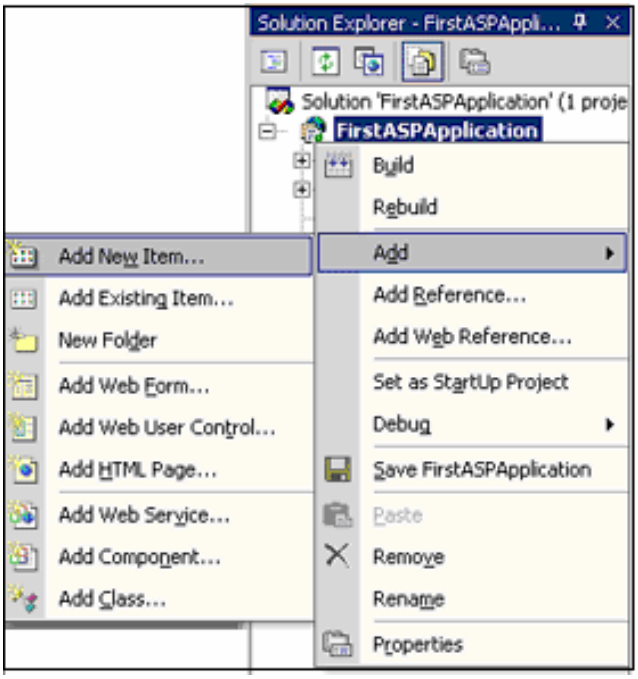
გახსნილი ვებ-გვერდი ცარიელია, რადგან არ შეიცავს ვიზუალურ ელემენტებს.

2.2. ახალი ვებ-გვერდის შექმნა

შექმნილ აპლიკაციას დავამატოთ ახალი ASPX ვებ-გვერდი.

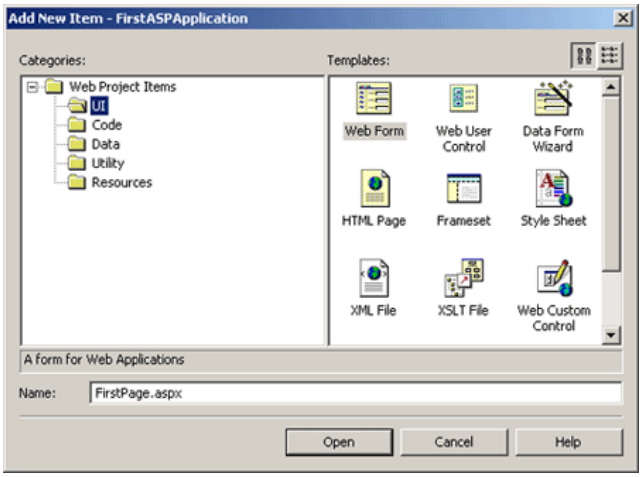
ახალი ფაილის დამატება შესაძლებელია კლავიატურის ღილაკების კომბინაციის CTRL+SHIFT+A საშუალებით, ან მონიშნეთ პროექტი Solution Explorer ფანჯარაში, თაგუს მარჯვენა ღილაკზე დაჭერით გამოიძახეთ კონტექსტური მენიუ და აირჩიეთ:

Add->Add New Item



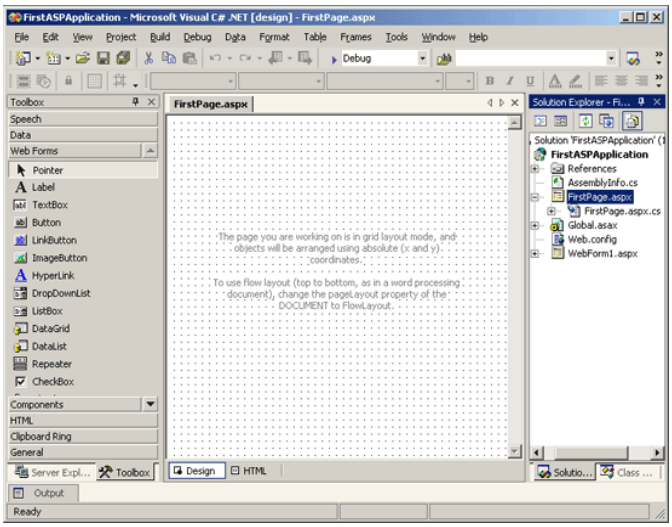
ნახ.2.6

გაიხსნება ახალი ფაილის არჩევის ფანჯარა, მარცხენა ფანჯარაში შესაძებელია ფაილების კატეგორიის არჩევა, ხოლო მარცხენაში ფანჯარაში წინასწარ განსაზღვრეული შაბლონებიდან შესაბამისი ტიპის ფაილების არჩევა. კატეგორიების ფანჯარაში აირჩიეთ UI ხოლო შაბლონების ფანჯარაში Web Form. Name ველში შეიყვანეთ ფაილის სასურველი სახელი:



ნახ.2.7

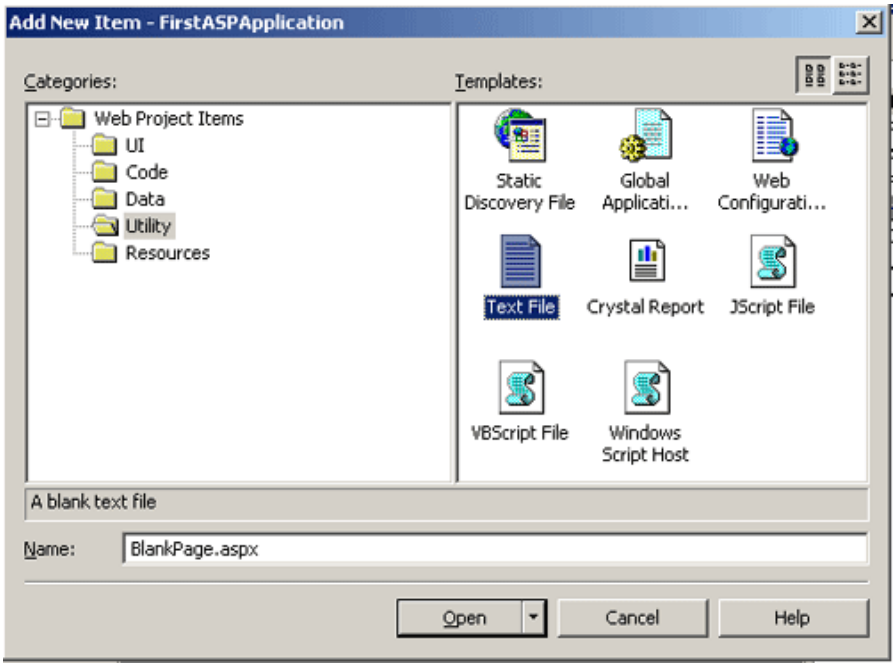
Open ღილაკზე დაჭერით პროექტს დაემატება ახალი ფაილი FirstPage.aspx და FirstPage.aspx.cs. გვერდი ავტომატურად გაიხსნება დიზაინის რეჟიმში.



ნახ.2.8

2.3. ახალი გვერდის დამატება შაბლონის გამოყენების გარეშე

შესძლებელია პროექტს დაემატოს ახალი ვებ-გვერდი შაბლონების გამოყენების გარეშე, ამისათვის გახსენით ახალი ფაილის დამატების ფანჯარა, მონიშნეთ Utility კატეგორია, შაბლონების ფანჯარაში აირჩიეთ Text file. Name ველში აკრიფეთ ფაილის სახელი და გაფართოება მიუთითეთ .aspx, მაგალითად აკრიფეთ BlankPage.aspx.



ნახ.2.9

პროექტს დაემატება შესაბამის ფაილი რომელიც არის ცარიელი (არ შეიცავს HTML და C# კოდს).

2.4. ფუნქციონალური ვებ-გვერდის შექმნა

შევქმნათ ვებ-გვერდი რომელიც ვებ-ბრაუზერის ეკრანზე გამოტანს მიმდინარე თარიღსა და დროს:

1. შექმენით ASP.NET პროექტი და დაამატეთ ცარიელი ASPX გვერდი სახელით CurrentDate.aspx.
2. გახსენით ფაილი და აკრიფეთ შემდეგი ტექსტი:

```
<%@ Page Language="C#" %>
<html>
      <head>
      </head>
      <body>
            <p><%= System.DateTime.Now %></p>
      </body>
</html>
```

ნახ.2.10

გაუშვით პროექტი CTRL+F5 ღილაკების კომბინაციის აკრეფით, ან კონტექსტური მენიუში აირჩიეთ პუნქტი View in Browser. ვებ-ბრაუზერში დაინახავთ მიმდინარე თარიღს და დროს. მაგ.: 20.09.2008 14:19:34.

გვერდის დასაწყისში მოთავსებული დირექტივა-<%@ Page Language="C#" %> მიუთითებს, რომ გვერდი უნდა დაკომპილირდეს C# ენის კომპილატორის საშუალებით. შესაძლებელია C# მაგივრად იყოს VB, რაც ნიშნავს, რომ გვერდის კოდი შექმნილია Visual Basic .Net ენით. ვებ-ბრაუზერში გვერდის გამოძახებისას, დირექტივაში მითითებული ენის კომპილატორი წაიკითხავს და შეასრულებს გვერდზე მოთავსებულ კოდს.

<% %> ტეგებს შორის შესაძლებელია კოდის მოთავსება, რომელიც შესრულდება გვერდის გამოძახების პროცესში სერვერის მხარეს, ხოლო გამოსახულების შედეგი გადაეცემა ვებ-ბრაუზერს. გამოსახულება <%= System.DateTime.Now %> გვერდის გამოძახებისას დააბრუნებს მიმდინარე თარიღის და დროის მნიშვნელობას.

2.5. სერვერული კონტროლების გამოყენება

ჩვეულებრივი HTML კონტროლების გარდაქმნა სერვერულ კონტროლებად შესაძლებელია მისთვის 2 ატრიბუტის დამატებით: Runat და ID. Runat ატრიბუტს მიენიჭება ყოველთვის მიენიჭება მნიშვნელობა Server, ხოლო ID ატრიბუტი წარმოადგენს კონტროლის იდენტიფიკატორს-სახელს, რომლის საშუალებითაც შესაძლებელია მივმართოთ კონტროლს სერვერილი სკრიპტის კოდში. იგი უნდა იყოს უნიკალური და არ უნდა ემთხვეოდეს სხვა სერვერული კონტროლების იდენტიფიკატორებს.

წინა მაგალითი შესაძლებელია შესრულებული იყოს სერვერული კონტროლის გამოყენებით:

```
<%@ Page Language="C#" %>
<%
      CurrentDate.InnerText =
System.DateTime.Now.ToString();
%>
<html>
      <head>
      </head>
      <body>
            <p id="CurrentDate"
runat="server"></p>
      </body>
</html>
```

ნახ.2.11

<p> ტეგისთვის runat="server" ატრიბუტის მნიჭებით იგი გარდაიქმნა სერვერულ კონტროლად და შესაძლებელი გახდა გვერდის კოდში მისი მიმართვა.

2.6. Web-კონტროლების გამოყენება

ვებ-კონტროლები სერვერული კონტროლების ნაირსახეობაა. ისინი HTML კონტროლების მსგავსია, ოგონდ უფრო მეტი, რთული თვისებები და მეთოდები გააჩნია. კლიენტის ვებ-გვერდზე ისინი გამოისახება როგორც ერთი ან რამდენიმე HTML კონტროლის სახით.

გადავაკეთოთ წინა მაგალითი ვებ-კონტროლის გამოყენებით:

```
<%@ Page Language="C#" %>
<%
      CurrentDate.Text =
System.DateTime.Now.ToString();
%>
<html>
 <head>
 </head>
 <body>
  <asp:Label Runat="server"
ID="CurrentDate"></asp:Label>
 </body>
</html>
```

ნახ.2.12

asp:Label არის სერვერული კონტროლი რომელსაც გამოაქვს ტექსტი, რომელიც მას მიენიჭება Text ატრიბუტის საშუალებით.

2.7. Response ობიექტის გამოყენება

Response ობიექტის მეთოდების საშუალებით შესაძლებელია კლიენტს ანუ ვებ-ბრაუზერს გაუგზავნოს მონაცემები.

```
<%@ Page Language="C#" %>
<html>
  <head>
  </head>
  <body>
      <% Response.Write(System.DateTime.Now); %>
  </body>
</html>
```

ნახ.2.13

2.8. სერვერული ფუნქციის გამოყენება

ვებ-გვერდზე სერვერული მეთოდები და ფუნქციები აღიწერება <script> ტეგის საშუალებით.

```
<%@ Page Language="C#" %>
<script runat="server">
 string GetCurrentDateTime(){
  return System.DateTime.Now.ToString();
 }
</script>
<html>
 <head>
 </head>
 <body>
  <p><%=GetCurrentDateTime()%></p>
 </body>
</html>
```

ნახ.2.14

ამ მაგალითში აღწერილია ფუნქცია GetCurrentDateTime(), რომელიც აბრუნებს მიმდინარე დროის მნიშველობას. ფუნქციის გამოძახება ხდება სერვერული სკრიპტის ტეგებს შორის.

2.9. სერვერული კონტროლების გამოყენება web-გვერდის მოვლენების დამუშავების პროცედურაში

ASPX გვერდის გამოძახებისას ვებ-გვერდის ეკრანზე გამოტანამდე სრულდება რამდენიმე ეტაპი. გვერდზე სხვადასხვა მონაცემების გამოსატანად ძირითადად გამოიყენება გვერდის ჩატვირთვის ეტაპი: Page_Load. გვერდზე სერვერული მხარეს შესასრულებელი კოდი თავსდება <script> ტეგის საშუალებით, რომელსაც ენიჭება ატრიბუტი runat="server".

```
<%@ Page Language="C#" %>
<html>
 <head>
  <script runat="server">
 void Page_Load(Object sender, EventArgs e)
 {
  CurrentDate.Text = System.DateTime.Now.ToString();
 }
  </script>
</head>
  <body>
    <asp:Label Runat="server"ID="CurrentDate"></asp:Label>
  </body>
</html>
```

ნახ.2.15

2.10. ვებ-გვერდის ვიზუალური და პროგრამული ნაწილების განცალკევება

ASP.NET გვერდები შესაძლოა შედგებოდეს 2 ფაილისგან: 1. ვიზუალური გვერდისგან, სადაც განთავსებულია HTML კოდი და კომპონენტები; 2. კოდის ფაილისგან, სადაც მუშავდება ვებ-ბრაუზერის მომხმარებლის მიერ ინიცირებული სხვადასხვა შეტყობინებები და ბიზნეს ლოგიკის პროცედურები და ფუნქციები. ეს მეთოდი ცნობილია Codebehind-ის სახელწოდებით. განვიხილოთ მაგალითი, ამ მეთოდის გამოყენებით:

1. შექმენით ახალი ვებ-აპლიკაცია ან გახსენით უკვე არსებული;
2. დაამატეთ ახალი ვებ-გვერდი VisualPage.aspx (ცარიელი ASPX ფაილი);
3. დაამატეთ ახალი C# ფაილი ProgramPage.cs (ცარიელი C# ფაილი);
4. VisualPage.aspx შეავსეთ HTML კოდით და სერვერული კონტროლებით:

```
<%@ Page Language="C#" %>
<html>
 <head>
 </head>
 <body>

 </body>
</html>
```

ნახ.2.16

5. ProgramPage.cs ფაილში აკრიფეთ შემდეგი ტექსტი:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

 public class ProgramPage : System.Web.UI.Page
 {
  private void Page_Load(object sender,
System.EventArgs e)
  {

  }
  override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }

  private void InitializeComponent()
  {
   this.Load += new
System.EventHandler(this.Page_Load);
  }
 }
```

6. VisualPage.aspx ფაილში <body></body> ტეგებს შორის დაამატეთ სერვერული კონტროლი:

```
<body>
      <asp:Label Runat="server"
                ID="CurrentDate"></asp:Label>
</body>
```

ნახ.2.18

7. VisualPage.aspx გვერდის დასაწყისში დაამატეთ დირექტივა:

```
 <%@ Page language="c#"
Codebehind="ProgramPage.cs"
AutoEventWireup="false" Inherits="ProgramPage" %>
```

ეს დირექტივა მიუთითებს, რომ ამ გვერდის მოვლენები დამუშავდება ფაილში ProgramPage.cs და გვერდის შესაბამისი კლასის სახელია ProgramPage.

8. VisualPage.aspx მოთავსებული სერვერული კონტროლზე მიმართვისთვის პროგრამის კოდში საჭიროა ეს კონტროლი აღიწეროს ProgramPage.cs კლასში, ამისათვის მას დაამატეთ ველი:

```
public class ProgramPage : System.Web.UI.Page
 {
       protected System.Web.UI.WebControls.Label
       CurrentDate;
.....
```

ნახ.2.19

System.Web.UI.WebControls.Label მიუთითებს რომ ამ ველის ტიპია სერვერული კონტროლი Label-ის კლასი.

9. მიმდინარე თარიღის და დროის გამოსატანად სერვერული კონტროლის საშუალებით გვერდის ჩატვირთვის მოვლენის დამუშავების პროცედურაში-Page_Load მეთოდში კონტროლის მიმართვა ხდება მისი იდენტიფიკატორის მიხედვით CurrentDate და მის ატრიბუტს Text მიენიჭება მიმდინარე თარიღი და დროის მნიშვნელობა:

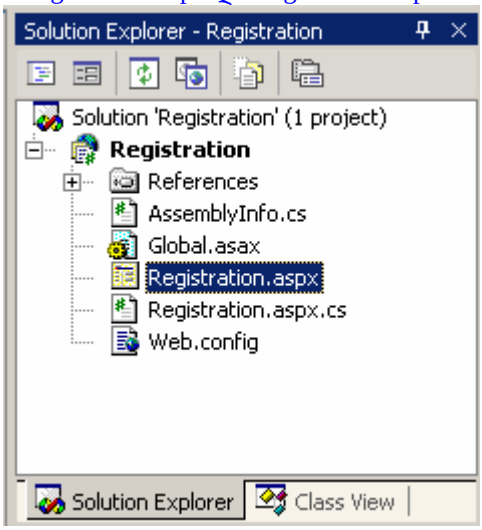
```
  private void Page_Load(object sender,
System.EventArgs e)
  {
   CurrentDate.Text = System.DateTime.Now.ToString();
  }
```

ნახ.2.20

2.11. ინტერაქტიული Web-გვერდის შექმნა

შევქმნათ ვებ-გვერდი რომელზეც მომხმარებელი შეიყვანს საკუთარ მონაცემებს და გადააგზავნის სერვერზე.

შექმენით ახალი ASP.NET აპლიკაცია სახელით Registration. დაამატეთ ფაილები Registration.aspx და Registration.aspx.cs.



ნახ.2.21

ვებ-გვერდის მოთავსებულია სერვერული კონტროლები: form, asp:TextBox, asp:TextBox, asp:DropDownList, asp:CheckBoxList, asp:Button, asp:Label. გვერდის ჩატვირთვისადსს ღილაკზე დაჭერისას გამოიძახება onclick მოვლენაზე მიბმული მეთოდი Register_Click.

გახსენით Registration.aspx და შეიყვანეთ შემდეგი კოდი:

```
<%@ Page language="c#" Codebehind="Registration.aspx.cs"
  AutoEventWireup="false"
Inherits="FirstASPApplication.Registration" %>
<HTML>
 <HEAD>
  <title>რეგისტრაციის ფორმა</title>
 </HEAD>
 <body>
  <form method="post" runat="server" id="registration">
   შეიყვანეთ საკუთარი მონაცემები:
   <table border="1">
    <tr>
```

```
     <td>სახელი:</td>
     <td>
      <asp:TextBox id="FirstName"
runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>გვარი:</td>
     <td>
      <asp:TextBox id="LastName"
runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>სქესი:</td>
     <td><asp:RadioButtonList id="Sex" runat="server"
                    RepeatDirection="Horizontal">
       <asp:ListItem Value="მდედრობითი"></asp:ListItem>
       <asp:ListItem Value="მამრობითი"></asp:ListItem>
      </asp:RadioButtonList></td>
    </tr>
    <tr>
     <td>ქალაქი</td>
     <td><asp:DropDownList id="City" runat="server">
       <asp:ListItem Value="თბილისი"></asp:ListItem>
       <asp:ListItem Value="ქუთაისი"></asp:ListItem>
       <asp:ListItem Value="რუსთავი"></asp:ListItem>
       <asp:ListItem Value="გორი"></asp:ListItem>
       <asp:ListItem Value="ბათუმი"></asp:ListItem>
       <asp:ListItem Value="თელავი"></asp:ListItem>
      </asp:DropDownList></td>
    </tr>
    <tr>
     <td>ინტერესების სფერო:</td>
     <td>
      <asp:CheckBoxList id="Interests" runat="server">
       <asp:ListItem Value="საინფორმაციო
ტექნოლოგიები"></asp:ListItem>
       <asp:ListItem
Value="სამართალმცოდნეობა"></asp:ListItem>
       <asp:ListItem Value="ეკონომიკა და
მენეჯმენტი"></asp:ListItem>
       <asp:ListItem Value="სამშენებლო
სფერო"></asp:ListItem>
```

```
      </asp:CheckBoxList>æ/td>
    </tr>
   </table>
   <asp:Button id="Register" runat="server"
Text="რეგისტრაცია"></asp:Button>
   <br />
   <asp:Label id="Message" runat="server"></asp:Label>
  </form>
 </body>
</HTML>
```

ნახ.2.22

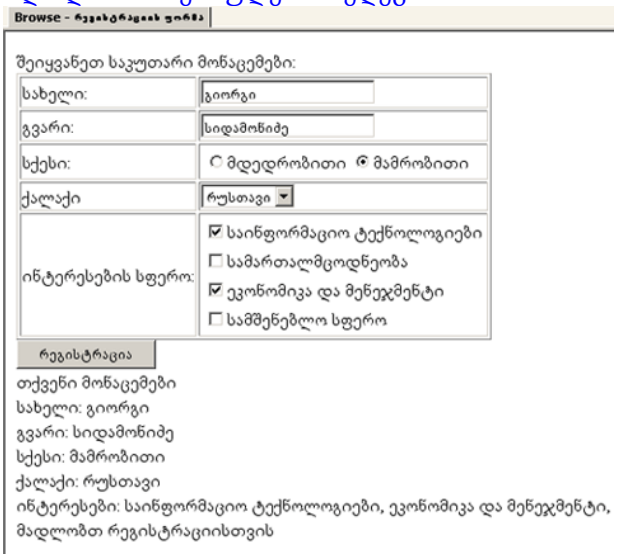
გახსენით Registration.aspx.cs და შეიყვანეთ შემდეგი კოდი:

```
using System;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace FirstASPApplication
{
 public class Registration : System.Web.UI.Page
  {
   protected System.Web.UI.WebControls.TextBox FirstName;
   protected System.Web.UI.WebControls.TextBox LastName;
   protected System.Web.UI.WebControls.RadioButtonList Sex;
   protected System.Web.UI.WebControls.DropDownList City;
   protected System.Web.UI.WebControls.CheckBoxList
                                              Interests;
   protected System.Web.UI.WebControls.Label Message;
   protected System.Web.UI.WebControls.Button Register;

    private void Register_Click(object sender,
                                System.EventArgs e)
   {
      StringBuilder sb = new StringBuilder();
      sb.Append("თქვენი მონაცემები<br>");
      sb.AppendFormat("სახელი: {0}<br>", FirstName.Text);
      sb.AppendFormat("გვარი: {0}<br>", LastName.Text);
      sb.AppendFormat("სქესი: {0}<br>", Sex.SelectedValue);
    sb.AppendFormat("ქალაქი: {0}<br>", City.SelectedValue);
      sb.Append("ინტერესები: ");
```

```
    foreach(ListItem item in Interests.Items)
     {
       if(item.Selected)
         sb.AppendFormat("{0}, ", item.Value);
     }
    sb.Append("<br>მადლობთ რეგისტრაციისთვის");
    Message.Text = sb.ToString();
  }

   override protected void OnInit(EventArgs e)
   {
    InitializeComponent();
    base.OnInit(e);
   }
  private void InitializeComponent()
   {
    this.Register.Click += new
       System.EventHandler(this.Register_Click);
   }
 }
}
```

ნახ. 2.23

ვებ-გვერდი და მისი შესრულების შედეგი ჩანს 2.24 ნახაზზე:



ნახ.2.24

2.12. კონფიგურაციის ფაილები და მათი იერარქია. კონფიგურაციის მონაცემები (lab5)

ვებ-აპლიკაციათა სერვერების ერთ-ერთი მთავარი მოთხოვნაა კონფიგურაციის მდიდარი და მოქნილი საშუალებების ქონა. ეს მექანიზმი საშუალებას იძლევა, რომ აპლიკაციის კოდში რომელიმე პარამეტრის მნიშვნელობა არ იყოს სტატიკურად აღწერილი და იყოს შესაძლებელი მისი ცვლილება აპლიკაციის კოდში შესწორებების შეტანის და რეკომპილაციის გარეშე. აგრეთვე ვებ სერვერების და აპლიკაციების ადმინისტრატორებს საშუალებას აძლევს ადვილად ცვალონ პარამეტრები აპლიკაციის გაშვების შემდეგ.

კონფიგურაციის ფაილები არის XML ფორმატის, რომელიც შეიცავს აპლიკაციის კონფიგურაციის მონაცემებს: სესიის პარამეტრები, მონაცემთა ბაზასთან კავშირის პარამეტრები, მომხმარებლების ავტორიზაციასა და აუთენთიფიკაციის კონფიგურაციის პარამეტრები და სხვა.

კონფიგურაციის ფაილებში ცვლილებები ავტომატურად აღიქმება სისტემის მიერ და არ საჭიროებს სერვერის გადატვირთვას.

უმეტეს შემთხვევაში კონფიგურაციის ფაილი მოთავსებულია აპლიკაციის ფესვურ დირექტორიაში. web.config არის სპეციალური ფაილი რომელშიც აღწერილი პარამეტრები გამოიყენება დირექტორიაში არსებული ფაილების და კლასების მიერ.

კონფიგურაციის პარამეტრის მნიშვნელობის დადგენა ხდება კონგიგურაციის მთელი იერარქიის დონეების გავლით. თუ პარამეტრის მნიშვნელობა განსაზღვრულია სხვადასხვა დონეებზე, მაშინ მისი იერარქიაში ყველაზე ზედა დონეზე მინიჭებული მნიშვნელობა

ASP.NET -ში არის კონფიგურაციის იერარქიული სტრუქტურა: სერვერის(მანქანური), ვებ-საიტის, აპლიკცაიის, აპლიკაციის ქვეკატალოგების დონეებზე.

კონფიგურაციის ფაილების დონეები და შესაძლო ფიზიკური მისამართები:

- მანქანური: `C:\WinNT\Microsoft.NET\Framework\v.1.00\config\machine.config`
- Web-სერვერის: `C:\inetpub\wwwroot\web.config`
- Web-საიტის: `D:\MyApplication\web.config`

- Web-საიტის ქვედირექტორიის: D:\MyApplication\MyDir \web.config

machine.config და web.config ფაილების XML ფაილებია და ფესვური ელემენტია: <configuration>.

კონფიგურაციის ფაილის მაგალითი:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <appSettings>
  <add key="mySetting" value="mySettingValue"></add>
 </appSettings>
 <system.web>
  <compilation defaultLanguage="c#" debug="true" />
  <customErrors mode="RemoteOnly" />
  <authentication mode="Windows" />
  <authorization>
   <allow users="*" />
  </authorization>
  <trace enabled="false" requestLimit="10" pageOutput="false"
traceMode="SortByTime" localOnly="true" />
  <sessionState mode="InProc" cookieless="false" timeout="20"
/>
  <globalization requestEncoding="utf-8"
responseEncoding="utf-8" />
 </system.web>
</configuration>
```

კონფიგურაციის ფაილების საშუალებით შესაძლებელია განისაზღვროს ცალკეული ვებ-გვერდების და ქვედირექტორიების პარამეტრები <location> ტეგის საშუალებით:

```
<configuration>
 <location path="EnglishPages">
<appSettings>
  <add key="BgColor" value="Red"></add>
 </appSettings>
 </location>
<location path="EnglishPages/OneJapanesePage.aspx">
<appSettings>
  <add key="BgColor" value="Green"></add>
 </appSettings>
 </location>
</configuration>
```

ამ შემთხვევაში EnglishPages ქვეკატალოგში არსებულ გვერდებზე BgColor პარამეტრს ენიჭება Red მნიშვნელობა. მხოლოდ OneJapanesePage.aspx გვერდზე ექნება BgColor პარამეტრს განსხვავებული მნიშვნელობა.

შემდეგ მაგალითში კონფიგურაციის ფაილის საშუალებით განსაძრვრულია რომ აპლიკაციის სესიის დასრულების დრო არის 20 წუთი. ანუ მომხმარებლის სესია დასრულდება მისი ბოლო აქტიურობიდან 20 წუთის შემდეგა და სესიაში დამახსოვრებული მონაცემები დაიკარგება.

```
<configuration>
  <system.web>
  <sessionState timeout="20" />
</system.web>
</configuration>
```

შემდეგ მაალითში მოყვანილია თუ როგორ არის შესაძლებელი პროგრამის კოდში კონფიგურაციის პარამტრების მნიშვნელობის გაგება. კონფიგურაციის პარამეტრების წვდომისთვის გამოიყენება ConfigurationSettings კლასის სტატიკური ფუნქცია AppSettings. ამ შემთხვევაში კონფიგურაციის ფაილიდან ხდება მომხმარებლის მიერ განსაზღვრული პარამეტრის BgColor მნიშვნელობის დადგენა და მისი მინიჭება ვებ გვერდის BODY ტეგის BgColor ატრიბუტისთვის.

// ფაილი web.config:

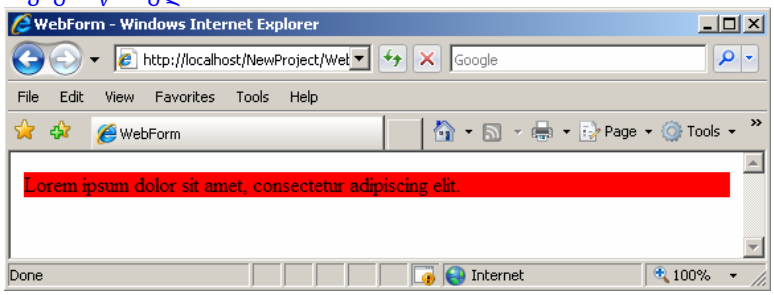
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<appSettings>
 <add key="BgColor" value="Red"></add>
 </appSettings>
</configuration>
```

// ფაილი WebForm6.aspx.cs:

```
using System;
using System.Web;
using System.Configuration;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

```
namespace NewProject
{
  public class WebForm1 : System.Web.UI.Page
  {
  protected System.Web.UI.WebControls.Panel panel1;

  private void Page_Load(object sender, System.EventArgs e)
  {
   String settingValue =
ConfigurationSettings.AppSettings["BgColor"];
   panel1.BackColor =
System.Drawing.Color.FromName(settingValue);
  }
 }
}
```

შედეგად გვერდის ჩატვირთვისას `panel1` ელემენტის ფონური ფერი იქნება წითელი

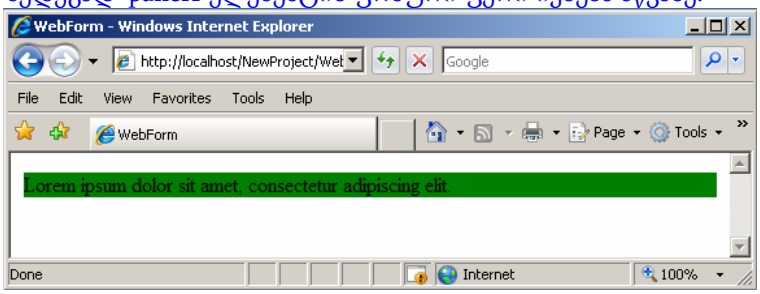


ნახ.2.25

ხოლო თუ web.cofig ფაილში შევცვლით BgColor-ის მნიშვნელობას სხვა ფერის დასახელებით, მაგალითად მწვანე ფერით:

<add key="BgColor" value="Green"></add>

შედეგად panel1 ელემენტის ფონური ფერი იქნება მწვანე:



ნახ.2.26

2.13. HTTP კონვეიერი. მისი შინაგანი სტრუქტურა. კლასები, მოვლენები, სპეციალიზებული დამმუშავებლები და მოდულები (lab6)

ASP.NET - ის პლატოფორმა აგებულია მოთხოვნების დამუშავების გაფართოებად არქიტექტურაზე, რომელსაც HTTP კონვეიერი ეწოდება. ყოველი გვერდის გამოძახებისას სრულდება კონვეიერში არსებული სხვადასხვა კლასების გამოძახება, რომლებიც ამ მოთხოვნის დამუშვებას ემსახურება. კონვეიერის გაფართოება სხვა კლასების დამატებით შესაძლებელია სამი გზით: სპეციალიზებული აპლიკაციით, სპეციუალური დამმუშავებლით და სპეციუალური მოდულით.

სპეციალური აპლიკაცია იქმნება ახალი კლასის შექმნით HttpApplication საბაზო კლასით. სპეციალური აპლიკაცია გამოიყენება აპლიკაციის დონის დავალებების შესასრულებლად სხვადასხვა მოვლენების დაჭერით და მათი დამუშავების გზით.

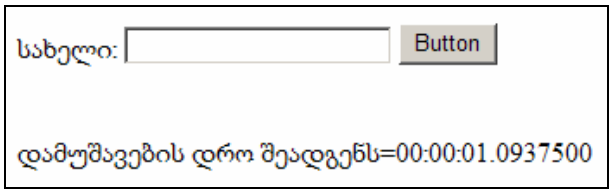
ვებ პროექტში Global.asax ფაილის დამატებისას აპლიკაციას ემატება კლასი რომელიც წარმოადგენს სპეციალურ აპლიკაციას. ამ კლასის საშუალებით შესაძლებელია სხვადასხვა მოვლენების დაჭერა. მაგალითად, თითოეული მოთხვნის დაწყება და დასასრული, სესიის დაწყება და დასასრული, შეცდომის მოვლენა და სხვ. მასში არსებული ფუნქციონალობის და რესურსებზე წვდომა შესაძლებელია Page და HttpContext კლასების ApplicationInstance თვისების საშაუალებით.

თუ აპლიაკცაის დავამატებთ ფაილს Globa.asax რომელშიც გადატვირთულია მეთოდები `Application_BeginRequest`, `Application_EndRequest` ამ აპლიკაციაში ყველა გვერდის გამოძახებისას გვერიდის ბოლოში გამოჩნდება გვერდის დამუშავების დრო:

```
// ფაილი Globa.asax:
using System;
using System.Collections;
using System.ComponentModel;
using System.Web;
using System.IO;
using System.Data;
using System.Web.SessionState;
```

```
namespace NewProject
{
 public class Global : System.Web.HttpApplication
 {
  protected void Application_BeginRequest(Object sender,
EventArgs e)
  {
   this.Context.Items["startTime"] = DateTime.Now;
  }
  protected void Application_EndRequest(Object sender,
EventArgs e)
  {
   DateTime start =
(DateTime)this.Context.Items["startTime"];
   TimeSpan ts = DateTime.Now - start;
   this.Context.Respsonse.Output.Write("<br />დამუშავების დრო
შეადგენს={0}",ts);
  }
 }
}
```

ყოველი გვერდის ბოლოში იქნება შეტყობინება, რომელიც გვერდის გამოძახების დაწყებიდან მის გამოძახების დასრულებამდე გავიდა:



ნახ.2.27

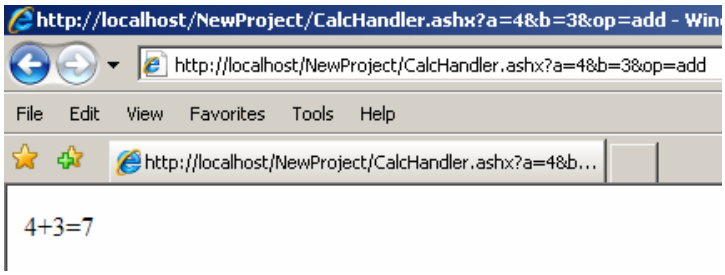
სპეციალური დამმუშავებელი არის კლასი რომლის გაზური ინტერფეისია IhttpHandler. რომ შევძლოთ ამ დამმუშვებლის გამოძახება აგრეთვე საჭიროა კონფიგურაციის ფაილში ინფორმაციის დამატება, რომელიც მიუთითებს, თუ როდის უნდა მოხდეს ამ კლასის გამოძახება. ასეთი დამმუშავებლის შექმნის მარტივი ხერხია აპლიკაციაში ashx გაფართოების ფაილის დამატება, რაც კონფიგურაციაში დამატებითი პარამეტრების რეგისტრირებას აღარ საჭიროებს.

შემდეგ სპეციალურ დამმუშავებელში რეალიზებულია კალკულაციის რამდენიმე ფუნქცია.

// ფაილი CalcHandler:

```
<%@ WebHandler Language="C#" Class="CalcHandler" %>
using System;
using System.Web;
 public class CalcHandler : IHttpHandler
 {
  public void ProcessRequest(HttpContext context)
  {
   int a = Int32.Parse(context.Request["a"]);
   int b = Int32.Parse(context.Request["b"]);
   switch(context.Request["op"])
    {
    case "add":
       context.Response.Output.Write("{0}+{1}={2}",a,b,a+b);
       break;
    case "substract":
       context.Response.Output.Write("{0}-{1}={2}",a,b,a-b);
       break;
    case "multiply":
       context.Response.Output.Write("{0}*{1}={2}",a,b,a*b);
        break;
    default:
       context.Response.Output.Write("ოპერაციის ტიპი უცნობია");
      break;
   }
  }

  public bool IsReusable{get{return true;}}
 }
```

ამ დამმუშავებლის გამოყენება შესაძლებელია თუ გამოვიძახებთ ამ ფაილს და გადავცემთ მას საჭირო პარამეტრებს. მაგალითად http://localhost/NewProject/CalcHandler.ashx?a=4&b=3&op=add:



ნახ.2.28

2.14. შეცდომების დიაგნოსტიკა. ტრასირება და მონიტორის მწარმოებლურობის მთვლელები. გამართვის პროცესი (lab7)

ASP.NET იძლევა აპლიკაციის მონიტორინგის და სხვადასხვა პრობლემების დიაგნოზის საშუალებას, როგორც შექმნის პროცესში ასევე მისი გაშვების შემდეგაც.

ტრასირება

ტრასირება საშუალებას იძლევა პროგრამის მუშაობის პროცესში შემოწმდეს სხვადასხვა ცვლადების მნიშვნელობები, გამოვიდეს შეტყობინებები, სესიის, აპლიკაციის, Cookies-ის მონაცემები და სხვა.

ცალკეული ვებ-გვერდისთვის ტრასირების ჩასართავად გამოიყენება დირექტივა:

```
<%@ Page Trace="true" %>
```

// ფაილი WebForm1.aspx“

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false" Inherits="WebApplication1.WebForm1"
Trace="true" %>
<HTML>
 <HEAD>
  <title>WebForm1</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <TABLE cellSpacing="0" cellPadding="0" border="0">
    <TR>
     <TD>სახელი:</TD>
     <TD>
      <asp:TextBox id="txtFirstname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>გვარი:</TD>
     <TD>
      <asp:TextBox id="txtLastname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>ასაკი:</TD>
     <TD>
      <asp:TextBox id="txtAge"
runat="server"></asp:TextBox></TD>
```

```
    </TR>
    <TR>
     <TD></TD>
     <TD>
       <asp:Button id="btnSave" runat="server"
              Text="დამახსოვრება"></asp:Button></TD>
    </TR>
   </TABLE>
  </form>
 </body>
</HTML>
```

ღილაკზე დაჭერის შემთხვევაში მოხდება ტრასირების ინფორმაციაში მომხმარებლის მიერ საკუთარი მონაცემების დამატება, მაგალითად:

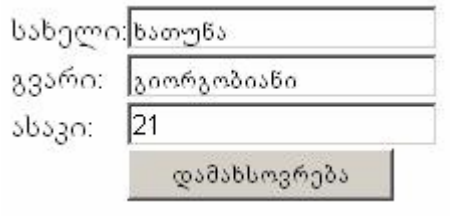
```
Trace.Write("UserDefined", String.Format("Session[\"Age\"] =
                        {0}", Session["Age"]));
```

`Trace.Warn()` მეთოდით გამოტანილი შეტყობინებები წითელი ფერისაა.

ფაილი WebForm1.aspx-ის ფრაგმენტი:

```
private void btnSave_Click(object sender, System.EventArgs e)
{
   Session["FirstName"] = txtFirstname.Text;
   Session["LastName"] = txtLastname.Text;
   Session["Age"] = Int32.Parse(txtAge.Text);
   Trace.Warn("სესიაში შენახული მონაცემები:");
   Trace.Write("UserDefined",
          String.Format("Session[\"FirstName\"] = {0}",
          Session["FirstName"]));
   Trace.Write("UserDefined",
          String.Format("Session[\"LastName\"] = {0}",
          Session["LastName"]));
   Trace.Write("UserDefined", String.Format("Session[\"Age\"]
          = {0}", Session["Age"]));
}
```

ამ გვერდის ბრაუზერში ნახვისას გამოვა შემდეგი სახის ინფორმაცია (ნახ.2.29-ა,ბ):



ნახ.2.29-ა

Request Details

Session Id:	np14nff2srnoqe45u5six445	**Request Type:**	POST
Time of Request:	12/22/2008 3:42:13 PM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	1.57892978078774E-05	0.000016
aspx.page	Begin Init	3.36387230440207E-05	0.000018
aspx.page	End Init	5.27663295312778E-05	0.000019
aspx.page	Begin InitComplete	6.52674434147211E-05	0.000013
aspx.page	End InitComplete	7.78387319550882E-05	0.000013
aspx.page	Begin LoadState	8.98786980930316E-05	0.000012
aspx.page	End LoadState	0.000155722576189628	0.000066
aspx.page	Begin ProcessPostData	0.000172073271252896	0.000016
aspx.page	End ProcessPostData	0.000232909686330422	0.000061
aspx.page	Begin PreLoad	0.000248999732667974	0.000016
aspx.page	End PreLoad	0.000261791570130102	0.000013
aspx.page	Begin Load	0.000273976898057387	0.000012
aspx.page	End Load	0.000287435394760292	0.000013
aspx.page	Begin ProcessPostData Second Try	0.000299550548030654	0.000012
aspx.page	End ProcessPostData Second Try	0.000311229615932989	0.000012
aspx.page	Begin Raise ChangedEvents	0.000323274594546427	0.000012
aspx.page	End Raise ChangedEvents	0.000370091115665657	0.000047
aspx.page	Begin Raise PostBackEvent	0.000384988192835502	0.000015
	სესიაში შენახული მონაცემები:	0.000406957872928177	0.000022
UserDefined	Session["FirstName"] = ხათუნა	0.000425098021743005	0.000018
UserDefined	Session["LastName"] = გიორგობიანი	0.00044005023614329	0.000015
UserDefined	Session["Age"] = 21	0.000459403404027803	0.000019
aspx.page	End Raise PostBackEvent	0.000474415768134022	0.000015

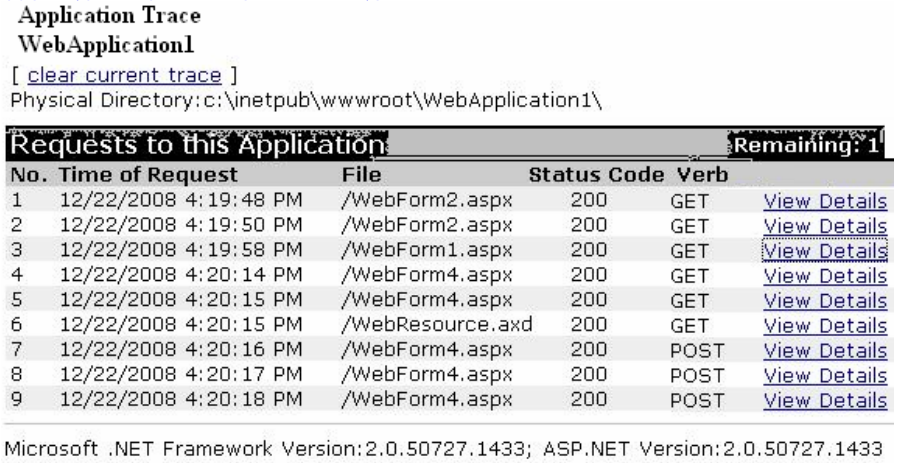
ნახ.2.29-ბ

შესაძლებელია აგრეთვე ტრასირების ჩართვა აპლიკაციის დონეზე:

```
<configuration>
 <system.web>
  <trace enabled="true" enabled="true"
      traceMode="SortByCategory"
```

```
      requestLimit="40"
      pageOutput="false"
      localOnly="true"
 />
 </system.web>
</configuration>
```

ამ შემთხვევაში ტრასირება ყველა გვერდისთვის არის ჩართული და მისი გამოტანა შესძლებელია trace.axd ფაილის გამოძახების საშუალებით, რომელიც უნდა მიეთითოს ვებ აპლიკაციის დასახელების შემდეგ. ამ დროს ინახება requestLimit პარამეტრით განსაზღვრული ვებ-გვერდების გამოძახების რაოდენობა.



ნახ.2.30

View Details დაჭერით შესაძლებელია თითოელ ვებ გვერდის გამოძახებასთან დაკავშირებული ტრასირების ინფორმაციის მიღება.

შეცდომების დამუშვება კონფიგურაციის ვებ-საიტის ფაილის დონეზე:

```
<configuration>
 <system.web>
  <customErrors mode="RemoteOnly"
defaultRedirect="/genericerror.htm">
 <error statusCode="500" redirect="/error/callsupport.htm" />
 <error statusCode="404" redirect="/error/notfound.aspx" />
 <error statusCode="403" redirect="/error/noaccess.aspx" />
```

```
  </customErrors>
 </system.web>
</configuration>
```

`defaultRedirect` მიუთითებს, რომ თუ შეცდომების კოდების ჩამონათვალში გათვალისწინებული შეცდომისგან განსხვავებული შეცდომა მოხდება საიტი გადამისამართდეს მითითებულ გვერდზე.

ვებ-გვერდის შეცდომის დამუშავება შესაძლებელია ვებ-გვერდის კლასის მეთოდში. ეს მეთოდი ავტომატურად გამოიძახება ამ გვერდზე შეცდომის შემთხვევაში:

```
private void Page_Error(object sender, EventArgs e)
{
 Response.Write(Server.GetLastError().Message);
 Server.ClearError();
}
```

აპლიკაციის დონეზე შეცდომების დაჭერა და დამუშვება შესაძლებელია Global.asax ფაილში. იგი შეიცავს მეთოდს, რომელიც შეცდომის დროს გამოიძახება ავტომატურად. შემდეგ მაგალითში მოცემულია შეცდომის დამუშვება, შეცდომის შესახებ ინფორმაციის ჩაწერა Windows -ის მოვლენათა რეესტრში (EventLog):

```
void Application_Error(Object sender, EventArgs e)
 {
  String Message = "\n\nURL:\n http://localhost/" +
Request.Path
    + "\n\nMESSAGE:\n " + Server.GetLastError().Message
    + "\n\nSTACK TRACE:\n" +
Server.GetLastError().StackTrace;

   String LogName = "Application";
   if (!EventLog.SourceExists(LogName))
   {
    EventLog.CreateEventSource(LogName, LogName);
   }

   EventLog Log = new EventLog();
   Log.Source = LogName;
   Log.WriteEntry(Message, EventLogEntryType.Error);
 }
```

2.15. შემოწმებათა სისტემა კლიენტისა და სერვერის მხარეს (ლაბ.8)

ASP.NET-ში სტანდარტულ კონტროლებს შორის არის აგრეთვე სერვერული კონტროლები, რომლებიც საშუალებას იძლევა შემოწმდეს მომხმარებლის მიერ ვებ-გვერდზე შეტანილი მონაცემები. არსებობს სხვადასხვა ტიპის შემოწმების კონტროლები, როგორიცაა მნიშვნელობის რაიმე განსაზღვრულ დიაპაზონში შემოწმება, განსაზღვრული შაბლონის მსგავსებაზე შემოწმება, აგრეთვე შემოწმება კონტროლში შეტანილია თუ არა მნიშვნელობა.

შემოწმების კონტროლებია:

- RequiredFieldValidator - ამოწმებს შევსუებული თუ არა კონტროლი
- CompareValidator - ადარებს ორი კონტროლის მნიშნველობებს
- RangeValidator - ამოწმებს შეტანილი მნიშვნელობა არის თუ არა განსაზღვრულ საზღვრებში.
- RegularExpressionValidator - ამოიწმებს ეთანხმება თუ არა შეტანილი მნიშვნელობა განსაზღვრულ შაბლონურ გამოსახულებას.
- CustomValidator - საშუალებას იძლევა მომხმარებლის მიერ განისაზღვროს შემოწმების ლოგიკა.
- ValidationSummary - გამოაქვს შეტყობინება შეცდომების შესახებ, რომლებსაც აბრუნებს ვალიდაციის კონტროლები.

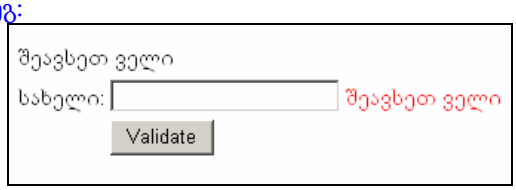
შემოწმება, როგორც წესი ხდება სერვერის მხარეს, თუმცა თუ ვებ-ბროუზერს უზრუნველყოფს DHTML სტანდარტს, შესაძლებელია შესრულდეს შემოწმება კლიენტის მხარეს. ამ შემთხვევაში არ ხდება გვერდის გადაგზავნა სერვერზე, სანამ ყველა შემოწმების კონტროლი არ იქნება დასაშვები. როცა კონტროლი არავალიდურია, მაშინ ეკრანზე გამოდის კონტროლის Text ატრიბუტის მნიშვნელობა.

შემდეგ მაგალითში ნაჩვენებია `RequiredFieldValidator` შემოწმების კონტროლის გამოყენების ხერხი. ეს კონტროლი ამოწმებს შეტანილი არის თუ არა კონტროლში რაიმე მნიშვნელობა. თუ მომხმარებელი შეიტანს რაიმე მნიშვნელობას, მაშინ ის შემოწმების კონტროლი არის ვალიდური. თუ ყველა შემოწმების კონტროლი ვალიდურია, მაშინ ვებ გვერდიც ვალიდურია:

```
<html>
 <head>
  <script language="C#" runat="server">
    void ValidateBtn_Click(Object Sender, EventArgs E)
     { if (Page.IsValid == true)
         {
           lblOutput.Text = "გვერდი ვალიდურია!";
         }
       else {
             lblOutput.Text = "ველის მნიშვნელობა ცარიელია";
            }
     }
  </script>
 </head>

 <body>
  <form runat="server" ID="Form1">
   <table>
    <tr>
     <td colspan="3">
      <asp:Label ID="lblOutput" Text="შეავსეთ ველი"
runat="server" /><br>
     </td>
    </tr>
    <tr>
     <td align="right">
      სახელი:
     </td>
     <td>
      <ASP:TextBox id="TextBox1" runat="server" />
     </td>
     <td>
      <asp:RequiredFieldValidator
id="RequiredFieldValidator2" ControlToValidate="TextBox1"
Display="Static" Width="100%"
 runat="server" ErrorMessage="შეავსეთ ველი">
</asp:RequiredFieldValidator>
     </td>
    </tr>
    <tr>
     <td></td>
     <td>
      <ASP:Button id="Button1" text="Validate"
OnClick="ValidateBtn_Click" runat="server" />
     </td>
```

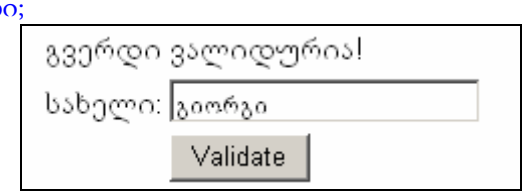
```
      <td></td>
     </tr>
    </table>
   </form>
  </body>
</html>
```

თუ მომხმარებელი ველს დატოვებს შეუვსებელს, მაშინ ღილაკზე დაჭერის შემდეგ:



ნახ.2.31-ა

ველის შევსების შემთხვევაში ღილაკზე დაჭერის შემდეგ გამოვა შემდეგი ეკრანი;

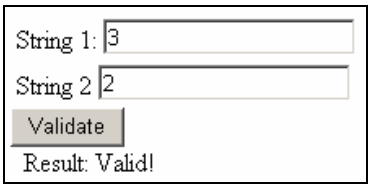


ნახ.2.31-ბ

განვიხილოთ CompareValidator (შედარების) კონტროლი. ეს კონტროლი ამოწმებს ორი სხვადასხვა ველის მნიშვნელობას. ამ კონტროლის ატრიბუტში `ControlToValidate` მიეთითება კონტროლის იდენტიფიკატორი, რომლის მნიშვნელობაც უნდა შემოწმდეს, ხოლო ატრიბუტით `ControlToCompare` კონტროლის იდენტიფიკატორი, რომლის მნიშვნელობასთანაც უნდა შემოწმდეს. თუ ამ კონტროლების მნიშვნელობები აკმაყოფილებს შედარების პირობებს, მაშინ `CompareValidator` –ის მნიშვნელობა ვალიდურია.

```
<%@ Page clienttarget=downlevel %>
<HTML>
 <HEAD>
  <script language="C#" runat="server">
```

```
        void Button1_OnSubmit(Object sender, EventArgs e) {

            if (comp1.IsValid) {
               lblOutput.Text = "Result: Valid!";
            }
            else {
               lblOutput.Text = "Result: Not valid!";
            }
        }
  </script>
 </HEAD>
 <body>
  <form runat="server" ID="Form1">
   <table>
    <tr>
     <td>
      String 1:
      <asp:TextBox id="txtComp" runat="server"></asp:TextBox>
     </td>
     <td>
     String 2
    <asp:TextBox id="txtCompTo" runat="server"></asp:TextBox>
     </td>
    </tr>
   </table>
   <asp:Button runat="server" Text="Validate" ID="Button1"
onclick="Button1_OnSubmit" />
   <br>
   <asp:CompareValidator id="comp1"
ControlToValidate="txtComp" ControlToCompare="txtCompTo"
Type="String"
    runat="server" Operator="GreaterThan" />
   <br>
   <asp:Label ID="lblOutput" runat="server" />
  </form>
 </body>
</HTML>
```



ნახ.2.32

- CustomValidator კონტროლის გამოყენება:

CustomValidator კონტროლი ძირითადად გამოიყენება, მაშინ როცა ვალიდაციის სხვა სტანდარტული კონტროლების მიერ რაიმე პირობების შემოწმება ვერ ხერხდება. ამისთვის ეს კონტროლი იძახებს მომხარებლის მიერ განსაზღვრულ ფუნქციას, რომელშიც შესაძლებელია სხვადასხვა პირობის შემოწმება. შესაძლებელია განისაზღვროს როგორც კლიენტის მხარეზე შემოწმების ფუნქცია, ასევე სერვერულ მხარეზეც. ClientValidationFunction ატრიბუტით განისაზღვრება კლიენტური მხარის ფუნქცია, ხოლო OnServerValidate ატრიბუტით სერვერული მხარის.

შემდეგ მაგალითში CustomValidator კონტროლის გამოყენებით მოწმდება შეყვანილი ციფრი ლუწია თუ კენტი. შემოწმება სრულდება როგორც კლიენტის ასე სერვერის მხარეს:

```
<html>
 <head>
  <script language="C#" runat="server">

 void ValidateBtn_OnClick(object sender, EventArgs e)
  {
    if (Page.IsValid)
       {
        lblOutput.Text = "Page is valid!";
       }
     else
       {
        lblOutput.Text = "Page is not valid! :-(";
       }
   }

void ServerValidate(object source, ServerValidateEventArgs value)
  { // even number?
    try {
          int num = Int32.Parse(value.Value);
          if (num%2 == 0) {
                  value.IsValid = true;
                   return;
               }
         }
    catch (Exception) {}
    value.IsValid = false;
  }
```

```
 </script>
</head>
<body>
 <h3><font face="Verdana">CustomValidator Example</font></h3>
 <p>
  <form runat="server" ID="Form1">
   <asp:Label id="lblOutput" runat="server" Text="Enter an
    even number:" /><br>
   <p>
    <asp:TextBox id="Text1" runat="server" />
    <asp:RequiredFieldValidator id="RequiredFieldValidator1"
     runat="server" ControlToValidate="Text1"
     ErrorMessage="Please enter a number"
     Display="Dynamic" ></asp:RequiredFieldValidator>
    <asp:CustomValidator id="CustomValidator1"
     runat="server" ControlToValidate="Text1"
     ClientValidationFunction="ClientValidate"
     OnServerValidate="ServerValidate" Display="Static">
          Not an even number!
   </asp:CustomValidator>
   <p>
    <asp:Button text="Validate"
     onclick="ValidateBtn_OnClick" runat="server"
     ID="Button1" />
    <script language="javascript">
      function ClientValidate(source, arguments)
      {
         // even number?
         if (arguments.Value%2 == 0)
           arguments.IsValid = true;
         else
           arguments.IsValid = false;
      }
    </script>
  </form>
 </body>
</html>
```

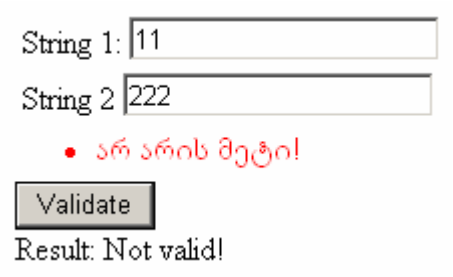
- ValidationSummary კონტროლის გამოყენება:

ამ კონტროლის საშუალებით შესაძლებელია ვალიდაციის კონტროლების შეტყობინებების გამოტანა ერთ სიაში ვებ გვერდის ერთ კონტრეტულ ადგილზე. სხვა ვალიდაციის კონტროლებს აქვთ

განსაზღვრული შეცდომის შესახებ შეტყობინებაა, ატრიბუტით ErrorMessage. ამ ატრიბუტის მნიშვნელობა გამოჩნდება ValidationSummary კონტრლოში. ამ კონტრლის გამოყენებით კონტროლების მნიშვნელობების შედარების მაგალითი შესაძლებელია გადაკეთდეს შემდეგნაირად:

```
<%@ Page clienttarget=downlevel %>
<HTML>
 <HEAD>
  <script language="C#" runat="server">
  void Button1_OnSubmit(Object sender, EventArgs e)
   {
      if (comp1.IsValid)
        {
          lblOutput.Text = "Result: Valid!";
        }
       else {
         lblOutput.Text = "Result: Not valid!";
      }
  }
   </script>
 </HEAD>
 <body>
  <form id="Form1" runat="server">
   <table>
    <tr>
     <td>String 1:
      <asp:textbox id="txtComp"
runat="server"></asp:textbox></td>
    </tr>
    <tr>
     <td>String 2
      <asp:textbox id="txtCompTo"
runat="server"></asp:textbox></td>
    </tr>
   </table>
     <asp:ValidationSummary id="ValidationSummary1"
      runat="server"></asp:ValidationSummary><asp:button
      id="Button1" onclick="Button1_OnSubmit" runat="server"
      Text="Validate"></asp:button>
     <br>
     <asp:comparevalidator id="comp1" runat="server"
      Type="String" ControlToCompare="txtCompTo"
      ControlToValidate="txtComp"
```

```
      Operator="GreaterThan" ErrorMessage="არ არის მეტი!"
      Display="None"></asp:comparevalidator>
     <asp:label id="lblOutput"
runat="server"></asp:label></form>
 </body>
</HTML>
```



ნახ.2.33

III თავი. პრაქტიკული: კომპლექსური ელემენტები

3.1. ASP.NET პაკეტში მონაცემებთან მუშაობა (ლაბ.9)

ASP.NET-ში მონაცემებთან მუშაობა საკმაოდ მარტივია. სტანდარტულ ელემენტებს შორის არის მონაცემების გამოტანის და დამუშვების ელემენტებიც. ამ ელემენტების თვისება DataSource და მეთოდი DataBind უზრუნველყოფენ მონაცემთა მიბმას კონტორლებზე. DataSource თვისებას ენიჭება მონაცემთა კოლექცია, მაგალითად მონაცემთა მასივი (კლასის ობიექტი, რომელშიც რეალიზებულია IDataReader ინტერფეისი), ან ობიექტი კლასისა DataSet. როდესაც მზადაა მონაცემების წყარო, რომ მოხდეს მისი წაკითხვა, იძახებენ მეთოდს DataBind(). ამის შემდეგ ელემენტი კითხულობს მონაცემებს და გამოაქვს ეკრანზე გამოტანისთვის საჭირო ფორმატით.

მონაცემების მიბმას უზრულველყოფს სხვადასხვა ელემენტები, ისეთი მარტივი როგორიცაა, მაგალითად ListBox, CheckBoxList, RadioButtonList, DropDownList და სპეციალური ელემენტები DataGrid, DataList DataRepeater. ამ ელემენტებზე შესაძლებელია .NET-ში აღწერილი კოლექციის კლასების ობიექტების მიბმა.

შემდეგ მაგალითშია ნაჩვენები, მონაცემების ელემენტებზე მიბმის ნიმუში. მონაცემები ინახება ArrayList ტიპის ობიექტში, რომელიც ებმება ელემენტების DataSource თვისებას და შემდეგ გმოიძახება ვებ-გვერდის DataBind() მეთოდი, რომლის საშუალებითაც სრულდება ამ ვებ გვერდზე არსებული ყველა ელემენტისთვის DataSource პარამეტრით მინიჭებული მონაცემების გამოტანა ვებ-ბრაუზერში.

ფაილი `WebForm12.aspx.cs`:

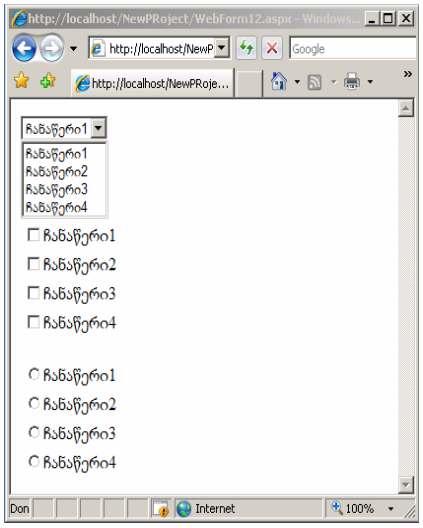
```
<%@ Page language="c#" Codebehind="WebForm12.aspx.cs"
AutoEventWireup="false" Inherits="NewPRoject.WebForm12" %>
<HTML>
 <HEAD>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:dropdownlist id="ddl1"
runat="server"></asp:dropdownlist><br>
   <asp:listbox id="lb1" runat="server"></asp:listbox><br>
```

```
   <asp:CheckBoxList id="cbl1"
runat="server"></asp:CheckBoxList><br>
   <asp:RadioButtonList id="rbl1"
runat="server"></asp:RadioButtonList>
    </form>
 </body>
</HTML>
```

ფაილი WebForm12.aspx.cs (ფრაგმენტი):

```
public class WebForm12 : System.Web.UI.Page
{
  protected System.Web.UI.WebControls.DropDownList ddl1;
  protected System.Web.UI.WebControls.ListBox lb1;
  protected System.Web.UI.WebControls.CheckBoxList cbl1;
  protected System.Web.UI.WebControls.RadioButtonList rbl1;

 private void Page_Load(object sender, System.EventArgs e)
      {
   if(!Page.IsPostBack)
   {
    ArrayList vals = new ArrayList();
    vals.Add("ჩანაწერი1");
    vals.Add("ჩანაწერი2");
    vals.Add("ჩანაწერი3");
    vals.Add("ჩანაწერი4");

    ddl1.DataSource = vals;
    lb1.DataSource = vals;
    cbl1.DataSource = vals;
    rbl1.DataSource = vals;
    Page.DataBind();
   }
  }
}
```

ბრაუზერში გამოვა შედეგი:



ნახ.3.1

ვებ-გვერდიდან მონაცემთა ბაზასთან დაკავშირება სრულდება ADO.NET ტექნოლოგიის გამოყენებით. არსებობს მონაცემების ბაზიდან ამოღების 2 მეთოდი: IDataReader ინტერფეისის ნაკადის და DataSet -ის კლასისი ობიექტის საშუალებით.

ყველაზე ეფექტური მეთოდი მონაცემების ამოღების არის IDataReader ინტერფეისის ნაკადის საშუალებით. ამ დროს არ ხდება ამოღებული მონაცემების შენახვა (caching).

შემდეგ მაგალითში შესრულდება ბაზიდან მონაცემების წაკითხვა და რეზულტატის DropDownList ელემენტში გამოტანა. DropDownList ელემენტს აქვს 2 თვისება DataTextField და DataValueField, პირველი მათგანი განსაზღვრავს მონაცემთა ცხრილში ველის დასახელებას, რომლის მნიშვნელობები გამოჩნდება ელემენტის ჩანაწერებში, ხოლო მეორე ელემენტის ჩანაწერთა მნიშვლეობების სვეტის დასახელებას. არსებულ მონაცემთა ბაზაში განსაზღვრულია ცხრილი Authors. ამ ცხრილიდან გამოგვაქვს ავტორის სახელები და მათი იდენტიფიკატორები. სიაში რომელიმე ჩანაწერის არჩევის შემდეგ, ღილაკზე დაჭერის შემთხვევაში ეკრანზე გამოგვავს ავტორის იდენტიფიკატორი.

ფაილი Authors.aspx:

```
<%@ Page language="c#" Codebehind="Authors.aspx.cs"
AutoEventWireup="false" Inherits="NewProject.Authors" %>
<HTML>
 <HEAD>
  <title>Authors</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:DropDownList id="ddlAuthors"
runat="server"></asp:DropDownList>
   <asp:Button id="btnSelect" runat="server"
Text="Button"></asp:Button><BR>
   <asp:Label id="selValue" runat="server"></asp:Label>
  </form>
 </body>
</HTML>
```

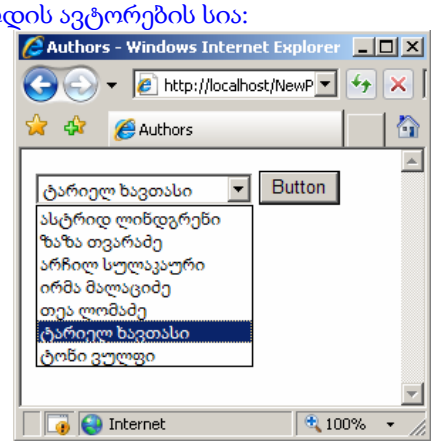
ფაილი Authors.aspx.cs :

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

```
using System.Web;
using System.Web.UI.WebControls;
namespace NewProject
{
public class Authors : System.Web.UI.Page
{
  protected System.Web.UI.WebControls.Label selValue;
  protected System.Web.UI.WebControls.Button btnSelect;
  protected System.Web.UI.WebControls.DropDownList
ddlAuthors;

private void Page_Load(object sender, System.EventArgs e)
       {
         if(!IsPostBack)
         {
           SqlConnection cn = new
               SqlConnection("server=(local);uid=sa;pwd=1;dat
               abase=Books");
           SqlCommand cmd = cn.CreateCommand();
           cmd.CommandText = "SELECT * FROM Authors";

           try
           {
            cn.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            ddlAuthors.DataSource = reader;
            ddlAuthors.DataTextField = "Name";
            ddlAuthors.DataValueField = "ID";
            ddlAuthors.DataBind();
           }
           finally
           {
            cn.Dispose();
           }
          }
       }
       private void btnSelect_Click(object sender,
System.EventArgs e)
       {
          selValue.Text = "თქვენ აირჩიეთ ავტორი: " +
ddlAuthors.SelectedValue;
       }
       override protected void OnInit(EventArgs e)
       {
              InitializeComponent();
```

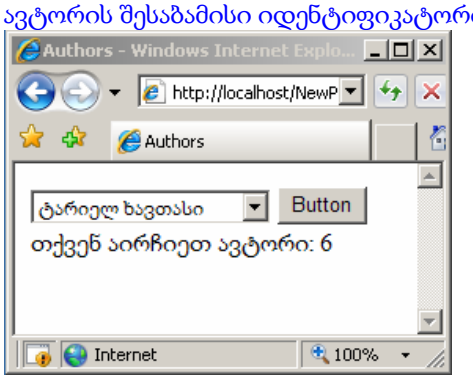
```
            base.OnInit(e);
        }
        private void InitializeComponent()
        {
            this.Load += new
        System.EventHandler(this.Page_Load);
            btnSelect.Click +=new
        EventHandler(btnSelect_Click);
        }
    }
}
```

ვებ-გვერდზე გამოდის ავტორების სია:



ნახ.3.2

რომელიმე ჩანაწერის არჩევის და ღილაკზე დაჭერის შემდეგ გამოდის ეკრანზე ავტორის შესაბამისი იდენტიფიკატორი:



ნახ.3.3

განვიხილოთ DataSet-ის ობიექტის საშუალებით მონაცემთა გამოტანის მეთოდი. DataSet - ის ობიექტი შესაძლებელია შეიცავდეს მონაცემების რამდენიმე ცხრილს, ამიტომ საჭიროა მიუთითოთ თუ რომელი ცხრილთან გვსურს კონკრეტულ შემთხვევაში მუშაობა. როდესაც DataSet მიებმევა რომელიმე ელემენტს, ამ ელემენტის DataMember ველის საშუალებით განისაზღვრება რომელი ცხრილის მიბმა არის საჭირო. თუ არ არის მითითებული, მაშინ აიღება პირველი ცხრილი. DataSet-ში არსებული მონაცემების ფილტრაციისა და სორტირებისთვის გამოიყენება DataView ობიექტი.

შემდეგ მაგალითში ნაჩვენებია DataSet ობიექტის გამოყენება, მისი საშუალებით მონაცემების სორტირება და ფილტრაცია.

ფაილი Employees.aspx:

```
<%@ Page language="c#" Codebehind="Employees.aspx.cs"
AutoEventWireup="false" Inherits="NewProject.Employees" %>
<HTML>
 <HEAD>
  <title>Employees</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:TextBox id="txtFilter" runat="server"></asp:TextBox>
   <asp:Button id="btnFilter" runat="server"
Text="ფილტრი"></asp:Button><BR>
   <asp:DataGrid id="dgEmployees"
        runat="server"></asp:DataGrid>სორტირება: 
   <asp:Button id="btnSortByID" runat="server"
Text="ID"></asp:Button> ან
   <asp:Button id="btnSortByLName" runat="server"
        Text="LastName"></asp:Button>
  </form>
 </body>
</HTML>
```

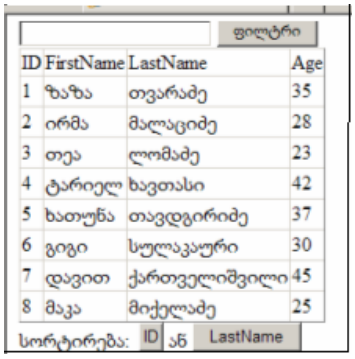
ფაილი Employees.aspx.cs:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.WebControls;
```

```
namespace NewProject
{
 public class Employees : System.Web.UI.Page
 {
  protected System.Web.UI.WebControls.DataGrid dgEmployees;
  protected System.Web.UI.WebControls.TextBox txtFilter;
  protected System.Web.UI.WebControls.Button btnFilter;
  protected System.Web.UI.WebControls.Button btnSortByLName;
  protected System.Web.UI.WebControls.Button btnSortByID;
  protected DataSet _ds;

private void Page_Load(object sender, System.EventArgs e)
  {
   GetData();
   if(!IsPostBack)
   {
    BindGrid();
   }
  }
private void GetData()
  {
   SqlConnection cn = new
     SqlConnection("server=(local);uid=sa;pwd=1;database=HR");
  SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Employees", cn);
   _ds = new DataSet();
   da.Fill(_ds, "Employees");
  }
private void BindGrid()
  {
   dgEmployees.DataSource = _ds;
   dgEmployees.DataBind();
  }
override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }
private void InitializeComponent()
  {
   this.btnFilter.Click += new
         System.EventHandler(this.btnFilter_Click);
   this.btnSortByLName.Click += new
        System.EventHandler(this.btnSortByLName_Click);
```

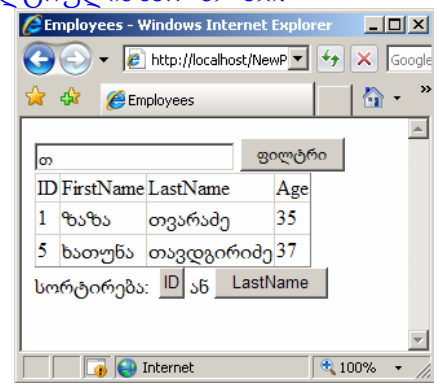
```
   this.btnSortByID.Click += new
System.EventHandler(this.btnSortByID_Click);
   this.Load += new System.EventHandler(this.Page_Load);
  }
private void btnFilter_Click(object sender, System.EventArgs e)
  {
   DataView dv = new DataView(_ds.Tables["Employees"]);
   dv.RowFilter = String.Format("LastName like '{0}%'",
        txtFilter.Text.Trim());
   dgEmployees.DataSource = dv;
   dgEmployees.DataBind();
  }
private void btnSortByID_Click(object sender, System.EventArgs e)
  {
   DataView dv = new DataView(_ds.Tables["Employees"]);
   dv.Sort = "ID ASC";
   dgEmployees.DataSource = dv;
   dgEmployees.DataBind();
  }
private void btnSortByLName_Click(object sender, System.EventArgs e)
  {
   DataView dv = new DataView(_ds.Tables["Employees"]);
   dv.Sort = "LastName ASC";
   dgEmployees.DataSource = dv;
   dgEmployees.DataBind();
  }
 }
}
```

გვერდის პირველი ჩატვირთვისას გამოდის ვებ-ბრაუზერში შემდეგი მონაცემები:



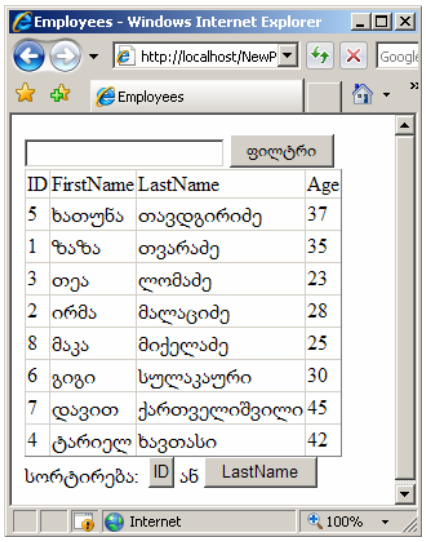
ნახ.3.4

თუ გვინდა გვარის მიხედვით გაფილტვრა უჯრაში შევიტანთ გვარის პირველ ასოს და დავაწვებით ღილაკს ”ფილტრი”. შემდეგ შემთხვევაში გაფილტრულია ასო ”თ”-თი:



ნახ.3.5

სორტირება გვარის ან იდენტიფიკატორის საშუალებით სრულდება შესაბამის ღილაკებზე დაჭერით. გვარის მიხედვით სორტირების მაგალითი:



ნახ.3.6

3.2. მონაცემთა ბადეები (ცხრილები), სორტირება და რედაქტირება. შაბლონები და მათი ელემენტები (ლაბ.10)

შემდეგ მაგალითში ნაჩვენებია DataGrid ელემენტის საშაულებით მონაცემების გამოტანის და ცლილებების მაგალითი. ცხრილის მონაცემების რედაქტირებისთვის ელემენტს ემატება EditCommandColumn, მისი საშუალებით გამოიძახება ელემენტის სხვადსხვა მეთოდები მოვლენების შემთხევაში, როგორიცაა ჩანაწერის ცვლილება, ცვლიელების გაუქმება, ცვლილების დამახსოვრება.

წაშლისთვის ემატება ღილაკი ButtonColumn რომელიც გადასცემს ბრძანებას Delete და იძახება შესაბამისი მეთოდი. ჩანაწერის განახლების ან წაშლისთვის იქმნება SQL ბრძანება-`SqlCommand`, და მისთვის საჭირო პარამეტერების გადაცემის შემდეგ გამოიძახება მისი მეთოდი `ExecuteNonQuery`, რომელიც მონაცემთა ბაზაში შეასრულებს ოპერაციას.

ფაილი EmployeesEdit.aspx:

```
<%@ Page language="c#" Codebehind="EmployeesEdit.aspx.cs"
AutoEventWireup="false" Inherits="NewProject.EmployeesEdit"
%>
<HTML>
 <HEAD>
  <title>Employees</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:DataGrid id="dgEmployees" DataKeyField="ID"
runat="server" AutoGenerateColumns="False">
    <Columns>
     <asp:EditCommandColumn ButtonType="LinkButton"
UpdateText="დამახსოვრება" CancelText="უარფოფა"
EditText="შეცვლა"></asp:EditCommandColumn>
     <asp:BoundColumn ReadOnly="True" DataField="ID"
HeaderText="ID"></asp:BoundColumn>
     <asp:BoundColumn DataField="FirstName"
HeaderText="სახელი"></asp:BoundColumn>
     <asp:BoundColumn DataField="LastName"
HeaderText="გვარი"></asp:BoundColumn>
     <asp:BoundColumn DataField="Age"
HeaderText="ასაკი"></asp:BoundColumn>
```

```
     <asp:ButtonColumn Text="წაშლა"
CommandName="Delete"></asp:ButtonColumn>
    </Columns>
   </asp:DataGrid>
  </form>
 </body>
</HTML>
```

ფაილი EmployeesEdit.aspx.cs:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.WebControls;
namespace NewProject
{
 public class EmployeesEdit : System.Web.UI.Page
 {
  protected System.Web.UI.WebControls.DataGrid dgEmployees;
  protected DataSet _ds;

private void Page_Load(object sender, System.EventArgs e)
  {
   if(!IsPostBack)
   {
    BindGrid();
   }
  }
private void BindGrid()
  {
   SqlConnection cn = new
SqlConnection("server=(local);uid=sa;pwd=1;database=HR");
   SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM
      Employees", cn);
   _ds = new DataSet();
   da.Fill(_ds, "Employees");
   dgEmployees.DataSource = _ds;
   dgEmployees.DataBind();
  }
override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }
private void InitializeComponent()
  {
```

```
    this.dgEmployees.CancelCommand += new
      System.Web.UI.WebControls.DataGridCommandEventHandler
         (this.dgEmployees_CancelCommand);
    this.dgEmployees.EditCommand += new
      System.Web.UI.WebControls.DataGridCommandEventHandler
         (this.dgEmployees_EditCommand);
    this.dgEmployees.UpdateCommand += new
      System.Web.UI.WebControls.DataGridCommandEventHandler
         (this.dgEmployees_UpdateCommand);
    this.dgEmployees.DeleteCommand += new
      System.Web.UI.WebControls.DataGridCommandEventHandler
         (this.dgEmployees_DeleteCommand);
    this.Load += new System.EventHandler(this.Page_Load);
  }
private void dgEmployees_EditCommand(object source,
       System.Web.UI.WebControls.DataGridCommandEventArgs e)
  {
   dgEmployees.EditItemIndex = e.Item.ItemIndex;
   BindGrid();
  }
private void dgEmployees_CancelCommand(object source,
       System.Web.UI.WebControls.DataGridCommandEventArgs e)
  {
   dgEmployees.EditItemIndex = -1;
   BindGrid();
  }
private void dgEmployees_UpdateCommand(object source,
       System.Web.UI.WebControls.DataGridCommandEventArgs e)
 {
    string updateCmd = "UPDATE Employees SET FirstName =
       @fName," + " LastName = @lName, Age = @age "+
                                          " WHERE ID=@id";
  SqlConnection cn = new
    SqlConnection("server=(local);uid=sa;pwd=1;database=Books");
    SqlCommand cmd =  new SqlCommand(updateCmd, cn);

    cmd.Parameters.Add("@fName",
            ((TextBox)e.Item.Cells[2].Controls[0]).Text);
    cmd.Parameters.Add("@lName",
            ((TextBox)e.Item.Cells[3].Controls[0]).Text);
    cmd.Parameters.Add("@age",
            ((TextBox)e.Item.Cells[4].Controls[0]).Text);
    cmd.Parameters.Add("@id",
        dgEmployees.DataKeys[dgEmployees.EditItemIndex]);
```

```
    try
    { cn.Open();
     cmd.ExecuteNonQuery();
     dgEmployees.EditItemIndex = -1;
    }
    finally
    { cn.Dispose();
    }
    BindGrid();
   }
   private void dgEmployees_DeleteCommand(object source,
     System.Web.UI.WebControls.DataGridCommandEventArgs e)
   {
    string deleteCmd = "DELETE FROM Employees WHERE ID=@id";
    SqlConnection cn = new
     SqlConnection("server=(local);uid=sa;pwd=1;database=HR");
    SqlCommand cmd =  new SqlCommand(deleteCmd, cn);
    cmd.Parameters.Add("@id",
dgEmployees.DataKeys[e.Item.ItemIndex]);

    try
    { cn.Open();
     cmd.ExecuteNonQuery();
    }
    finally
    { cn.Dispose();
    }
    BindGrid();
   }
  }
}
```

გვერდის ჩატვირთვის შემდეგ DataGrid ელემენტი (ნახ.3.7):

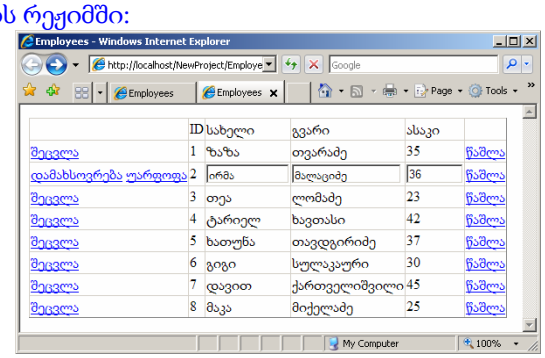
	ID	სახელი	გვარი	ასაკი	
შეცვლა	1	ზაზა	თვარაძე	35	წაშლა
შეცვლა	2	ირმა	მალაციძე	36	წაშლა
შეცვლა	3	თეა	ლომაძე	23	წაშლა
შეცვლა	4	ტარიელ	ხავთასი	42	წაშლა
შეცვლა	5	ხათუნა	თავდგირიძე	37	წაშლა
შეცვლა	6	გიგი	სულაკაური	30	წაშლა
შეცვლა	7	დავით	ქართველიშვილი	45	წაშლა
შეცვლა	8	მაკა	მიქელაძე	25	წაშლა

ნახ.3.7

ღილაკზე "შეცვლა" დაჭერის შემდეგ გამოიძახება მეთოდი, რომელიც განსაზღვრულია DataGrid კომპონენისთვის "Edit" ბრძანების მიღები შემთხვევაში:

```
private void dgEmployees_EditCommand(object source,
      System.Web.UI.WebControls.DataGridCommandEventArgs e)
  {
   dgEmployees.EditItemIndex = e.Item.ItemIndex;
   BindGrid();
  }
```

ამ მეთოდის გამოძახების შემდეგ შესაბამისი სსტრიქონი გადადის რედაქტირების რეჟიმში:



	ID	სახელი	გვარი	ასაკი	
შეცვლა	1	ზაზა	თვარაძე	35	წაშლა
დამახსოვრება უარყოფა	2	ირმა	მალაციძე	36	წაშლა
შეცვლა	3	თეა	ლომაძე	23	წაშლა
შეცვლა	4	ტარიელ	ხავთასი	42	წაშლა
შეცვლა	5	ხათუნა	თავდგირიძე	37	წაშლა
შეცვლა	6	გიგი	სულაკაური	30	წაშლა
შეცვლა	7	დავით	ქართველიშვილი	45	წაშლა
შეცვლა	8	მაკა	მიქელაძე	25	წაშლა

ნახ.3.8

"დამახსოვრება" ღილაკზე დაჭერის შემდეგ გამოიძახება მეთოდი `dgEmployees_UpdateCommand`. ამ მეთოდში მყარდება კავშირი მონაცემთა ბაზასთან `SqlConnection` ობიექტის საშუალებით, იქმნება `SqlCommand` რომელსაც გადაეცემა Sql ბრძანება და საჭირო პარამეტრები. `cmd.ExecuteNonQuery()` მეთოდის გამოძახების საშუალებით მონაცემთა ბაზაში შესაბამისი ჩანაწერი განახლდება.

თუ არ არის საჭირო შეცვლილი მონაცემების ცვლილება "უარყოფა" ღილაკზე დაჭერის შემდეგ რედაქტირებადი სტრიქონი დაბრუნდება ჩვეულებრივ რეჟიმში.

"წაშლა" ღილაკზე დაჭერის შემთხვევაში გამოიძახება მეთოდი `dgEmployees_DeleteCommand`. ისევე როგორც ჩანაწერის განახლების შემთხვევაში აქაც იქმნება `SqlCommand` ობიექტი, გადეცემა შესაბამის Sql ბრძანება და `cmd.ExecuteNonQuery()` მეთოდის გამოძახებით შესრულდება ჩანაწერის წაშლა ბაზაში.

შაბლონის საშუალებით მონაცემთა გამოტანას უზრუნველყოფს ელემენტები: DataList, Repeater.

ჩანაწერების გამოტანისთვის წინასწარ განისაზღვრება შაბლონები (ასევე საკმარისია განისაზღვროს მხოლოდ ItemTemplate შაბლონი):

```
ItemTemplate
AlternatingItemTemplate
SeparatorTemplate
HeaderTemplate
FooterTemplate
EditItemTemplate
SelectedItemTemplate
```

DataList-ს აქვს საშუალება ჩანაწერები გამოიტანოს ჰორიზონტალური მიმდევრობით RepeatLayout პარამეტრისთვის Flow მნიშვნელობის მინიჭებით, ან ჩვეულებრივად Table მნიშვნელობის მინიჭებით. წინა მაგალითებში გამოყენებული მონაცემები გამოვიტანოთ DataList –ის საშუალებით:

// ფაილი EmployeesView.aspx:

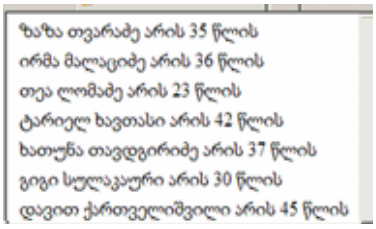
```
<%@ Page language="c#" Codebehind="EmployeesView.aspx.cs"
AutoEventWireup="false" Inherits="NewProject.Employees" %>
<HTML>
 <HEAD>
  <title>Employees</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:datalist id="dlEmployees" runat="server">
    <ItemTemplate>
     <%#DataBinder.Eval(Container.DataItem, "FirstName")%>
     <%#DataBinder.Eval(Container.DataItem, "LastName")%>
     არის
     <%#DataBinder.Eval(Container.DataItem, "Age")%>
     წლის
    </ItemTemplate>
   </asp:datalist></form>
 </body>
</HTML>
```

// ფაილი EmployeesView.aspx.cs:

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

```
using System.Web.UI.WebControls;

namespace NewProject
{
 public class EmployeesView : System.Web.UI.Page
 {
  protected System.Web.UI.WebControls.DataGrid dgEmployees;
  protected System.Web.UI.WebControls.DataList dlEmployees;
  protected DataSet _ds;

  private void Page_Load(object sender, System.EventArgs e)
  {
   if(!IsPostBack)
   {
    BindGrid();
   }
  }
  private void BindGrid()
  {
   SqlConnection cn = new
SqlConnection("server=(local);uid=sa;pwd=a;database=Books");
   SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM
Employees", cn);
   SqlCommandBuilder bldr = new SqlCommandBuilder(da);
   _ds = new DataSet();
   da.Fill(_ds, "Employees");
   dlEmployees.DataSource = _ds;
   dlEmployees.DataBind();
  }
  override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }
  private void InitializeComponent()
  {
   this.Load += new System.EventHandler(this.Page_Load);
  }
 }
}
```

ვებ-ბრაუზერში გამოვა შედეგი:



ნახ.3.9

3.3. ASP.NET პაკეტის მართვის სპეციალიზებული ელემენტები. კლასები System.Web.UI.Control და HtmlTextWriter. კლიენტის სცენარის გენერაცია (ლაბ.11)

ASP.NET-ში შესაძლებელია მოხმარებელმა თავად შექმნას სერვერული კონტროლები. ეს მას საშუალებას აძლევს მოახდინოს მომხმარებლის ინტერფეისის და სხვა ფუნქციონალობის ინკაპსულაცია კონტროლში, რომელის გამოყენება აპლიკაციაში მრავალჯერადად არის შესაძლებელი.

მომხმარებლის კონტროლის შექმნა შესაძლებელია `System.Web.UI.Control` ბაზური კლასის გამოყენებით. ჩვეულებრივ უნდა გადაიტვირთოს მისი მეთოდი Render რათა შევძლოთ HTML კოდი გენერაცია. Render მეთოდს გადაეცემა `HtmlTextWriter` ობიექტი, რომლის საშუალებითაც ხდება HTML კოდის გენერაცია.

მომხრებლის კონტროლებს კლასებს შესაძლებელია დაემატოს თვისებები, რომელთა საშალებითაც შესრულდება კონტროლის რომელიმე ველისთვის მნიშვნელობის მინიჭება ან ამ მნიშვნელობის გაგება.

შემდეგ მომხმარებლის კონტროლს დამატებული აქვს 2 თვისება `Message` და `Iterations`.

ამ თვისებებისთვის მნიშვნელობის მინიჭება ხდება ვებ გვერდის კოდში `Message="გამარჯობათ"`, `Iterations="3"`. კონტროლი გამოიტანს მითითებულ ტექსტს (`Message`) მითითებული რაოდენობით (`Iterations`):

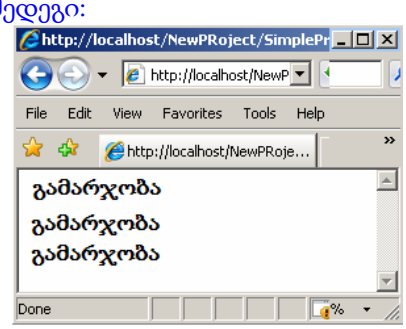
ფაილი SimplePropertyPage.aspx:

```
<%@ Register TagPrefix="SimpleControlSample"
Namespace="NewProject" Assembly="NewProject" %>
<html>
 <body>
  <form method="POST" runat="server" ID="Form1">
   <SimpleControlSample:SimpleProperty Message="Hello There"
Iterations="3" runat="server" ID="Simpleproperty1" />
  </form>
 </body>
</html>
```

// ფაილი SimpleProperty.cs:

```
using System;
using System.Web;
using System.Web.UI;
namespace NewProject
{
 public class SimpleProperty : Control
 {
  private String _message;
  private int _iterations = 1;

  public String Message
  {
   get { return _message;  }
   set { _message = value; }
  }

  public int Iterations
  {
   get { return _iterations; }
   set { _iterations = value; }
  }

  protected override void Render(HtmlTextWriter output)
  {
   for (int i=0; i<_iterations; i++)
   {
    output.Write("<h1>" + _message + "</h1>");
   }
  }
 }
```

}ეკრანზე მიიღება შედეგი:



ნახ.3.10

HtmlTextWriter ობიექტს გააჩნია ასევე სხვა მეთოდები HTML კონტროლების გენერაციისთვის: RenderBeginTag(), AddStyleAttribute(), AddAttribute(). ამ მეთოდების გამოყენება უფრო აადვილებს HTML ბლოკების კონსტრუქტორების კოდის წერას.

ფაილი SimpleTablePage.aspx:

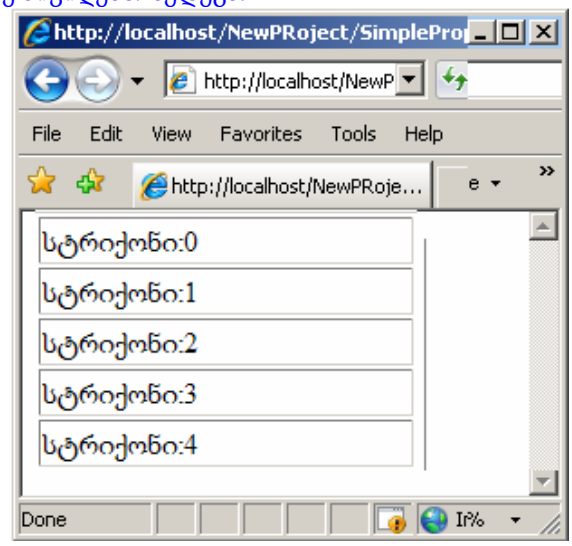
```
<%@ Register TagPrefix="SampleTableSample"
Namespace="NewProject" Assembly="NewProject" %>
<html>
 <body>
  <form method="POST" runat="server" ID="Form1">
       <SampleTableSample:SampleTable Rows="5" runat="server"
       ID="SampleTable1" />
  </form>
 </body>
</html>
```

ფაილი SampleTable.cs:

```
using System;
using System.Web;
using System.Web.UI;

namespace NewProject
{
 public class SampleTable : Control
 {
  private int _rows = 1;
  public int Rows
  {
   set {_rows = value;}
  }

  protected override void Render(HtmlTextWriter output)
  {
   output.AddAttribute("width", "50%");
   output.AddAttribute("border", "1");
   output.RenderBeginTag(HtmlTextWriterTag.Table);
   for (int i=0; i<_rows; i++)
   {
    output.RenderBeginTag(HtmlTextWriterTag.Tr);
    output.RenderBeginTag(HtmlTextWriterTag.Td);
    output.Write(String.Format("სტრიქონი:{0}",i));
    output.RenderEndTag();
    output.RenderEndTag();
```

```
    }
    output.RenderEndTag();
  }
 }
}
```

ეკრანზე მივიღებთ შედეგს:



ნახ.3.11

3.4. მართვის შედგენილი ელემენტები. მართვის მომხმარებელთა ელემენტები (ლაბ.12)

თუ ელემენტი შეიცავს სხვა ელემენეტებსაც, ამ შემთხვევაში მათ შედგენილ ელემენტებს უწოდებენ. შედგენილი ელემენტების შექმნა შესაძლებელია ქველემენტების შექმნით და მათი დამატებით მშობელი ელემენტის Controls კოლექციაში. უნდა შესრულდეს Control კლასის CreateChildControls მეთოდის გადატვირთვა, რომელიც გამოიძახება ქველემენტების გენერაციისთვის. თუ კონტროლის გამოყენება მოხდება ერთ გვერდზე მრავალჯერადად, ამ შემთხვევაში საჭიროა, რომ ამ კონტროლში იყოს System.Web.UI.INamingContainer ინტერფეისი

რეალიზებული. ეს უზრუნველყოფს ამ კონტროლის ეგზემპლარებისთის უნიკალური იდენტიფიკატორების უზრუნველყოფას.

შემდეგ მაგალითში CreateChildControls მეთოდში ხდება კონტროლების გენერაცია.

ფაილი Composition1.aspx:

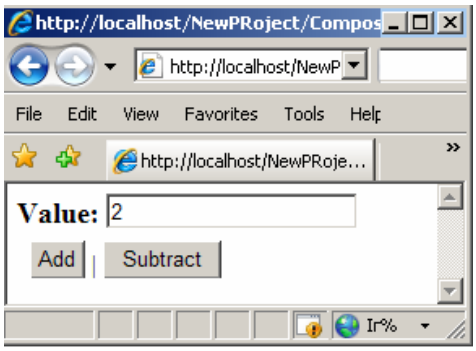
```
<%@ Register TagPrefix="CompositionSample"
Namespace="NewProject" Assembly="NewProject" %>
<html>
 <script language="C#" runat="server">
      private void AddBtn_Click(Object sender, EventArgs e) {
          MyControl.Value++;
      }
      private void SubtractBtn_Click(Object sender, EventArgs
e) {
          MyControl.Value--;
      }
 </script>
 <body>
  <form method="POST" action="Composition1.aspx"
runat="server" ID="Form1">
   <CompositionSample:Composition1 id="MyControl"
runat="server" />
   <br>
   <asp:button text="Add" OnClick="AddBtn_Click"
     runat="server" ID="Button1" NAME="Button1" />
   |
   <asp:button text="Subtract" OnClick="SubtractBtn_Click"
     runat="server" ID="Button2" NAME="Button2" />
  </form>
 </body>
```

ფაილი Composition1.cs:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NewProject
{
 public class Composition1 : Control, INamingContainer
 {
  public int Value
  {
   get
```

```
    {
     this.EnsureChildControls();
     return Int32.Parse(((TextBox)Controls[1]).Text);
    }
    set
    {
     this.EnsureChildControls();
     ((TextBox)Controls[1]).Text = value.ToString();
    }
   }

   protected override void CreateChildControls()
   {
    this.Controls.Add(new LiteralControl("<h3>" + "Value: "));
    TextBox box = new TextBox();
    box.Text = "0";
    this.Controls.Add(box);
    this.Controls.Add(new LiteralControl("</h3>"));
   }
  }
}
```

ეკრანზე გამოდის ფორმა სადაც არის ტექსტური ველი და ორი ღილაკი: Add და Substract. Add ღილაკზე დაჭერით ტექსტურ ველში არსებული რიცხვი გაიზრდება 1-ით, ხოლო Substract ღილაკზე დაჭერით შემცირდება 1-ით.



ნახ.3.12

აქამდე განვიხილეთ შედგენილი ელემენტების შექმნის ხერხი მხოლოდ პროგრამული კოდის საშუალებით, თუმცა ასევე შესაძლებელია მათი შექმნა სპეციალური გვერდების საშუალებით, რომლებსაც მომხმარებელთა ელემენტებს უწოდებენ (user controls). ამ

გვერდების გაფართოებაა .ascx, მათზე ძესაძლებელია სხვადასხვა HTML და სერვერული კონტროლების განთავსება. მომხმარებლის ელემენტების გამოყენება შესაძლებელია aspx გვერდებზე გვერდის @Register დირექტივის Src ატრიბუტის საშუალებით. თუ საჭიროა კონტროლის ჩატვირთვა პროგრამულად მაშინ გამოიყენება გვერდის მეთოდი LoadControl(), რომელსაც გადაეცემა მომხარებლის ელემენტის ფაილის მისამართი.

ქვემოთ მოცემულია მომხმარებლის ელემენტის მაგალითი, რომლეშიც ხდება Label კონტროლისთვის ტექსტის ცვლილება:

ფაილი WebUserControl1.ascx:

```
<%@ Control Language="c#" AutoEventWireup="false"
Codebehind="WebUserControl1.ascx.cs"
Inherits="NewProject.WebUserControl1"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%
>
<asp:Label ID="lblMessage" Runat="server"></asp:Label>
```

ფაილი WebUserControl1.ascx.cs:

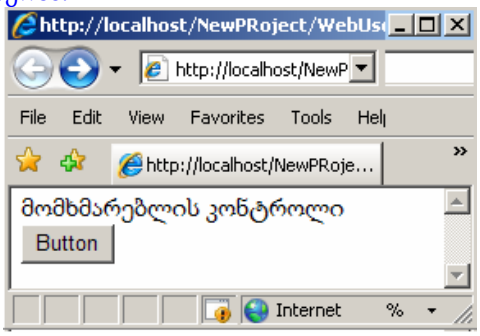
```
namespace NewProject
{
 using System;
 public class WebUserControl1 : System.Web.UI.UserControl
 {
  protected System.Web.UI.WebControls.Label lblMessage;

  public string Text
  {
   set{lblMessage.Text = value;}
  }

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

  override protected void OnInit(EventArgs e)
  {
       this.Load += new System.EventHandler(this.Page_Load);
       base.OnInit(e);
  }
 }
}
```

ამ კონტროლის გამოყენების მაგალითი aspx ვებ-გვერდზე:

ფაილი WebUserControlPage.aspx:

```
<%@ Page language="c#" %>
<%@ Register TagPrefix="control" TagName="WebUserControl1"
Src="WebUserControl1.ascx" %>
<HTML>
 <script language="C#" runat="server">

      void btnSubmit_Click(Object sender, EventArgs E) {
          MyWebUserControl.Text = "მომხმარებლის კონტროლი
შეიცვალა!";
      }

  </script>
 <HEAD>
 </HEAD>
 <body >
  <form id="Form1" runat="server">
   <control:WebUserControl1 id="MyWebUserControl"
     Text="მომხმარებლის კონტროლი"
     runat="server"></control:WebUserControl1>
   <br>
   <asp:Button id="btnSubmit"  runat="server"
OnClick="btnSubmit_Click"
    Text="Button"></asp:Button>
  </form>
 </body>
</HTML>
```

გვერდის ჩატვირთვისას ეკრანზე ჩანს (ნახ.3.13) კონტროლის საწყისი მდგომარეობა:



ნახ.3.13

კომპოზიტური კონტროლის მაგალითი:

CreateChildControls მეთოდში ხდება კონტროლების გენერაცია.

```
<%@ Register TagPrefix="CompositionSampleControls"
Namespace="CompositionSampleControls"
Assembly="CompositionSampleControls" %>
<html>
 <script language="C#" runat="server">
      private void AddBtn_Click(Object sender, EventArgs e) {
          MyControl.Value++;
      }
 private void SubtractBtn_Click(Object sender, EventArgs e) {
          MyControl.Value--;
      }
 </script>
 <body>
  <form method="POST" action="Composition1.aspx"
runat="server" ID="Form1">
   <CompositionSampleControls:Composition1 id="MyControl"
runat="server" />
   <br>
   <asp:button text="Add" OnClick="AddBtn_Click"
runat="server" ID="Button1" NAME="Button1" />
   |
   <asp:button text="Subtract" OnClick="SubtractBtn_Click"
runat="server" ID="Button2" NAME="Button2" />
  </form>
 </body>
</html>

using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CompositionSampleControls
{
 public class Composition1 : Control, INamingContainer
 {
  public int Value
  {
   get
   {
    this.EnsureChildControls();
    return Int32.Parse(((TextBox)Controls[1]).Text); }
```

```
   set
   {
    this.EnsureChildControls();
    ((TextBox)Controls[1]).Text = value.ToString(); }
  }

  protected override void CreateChildControls()
  {
   this.Controls.Add(new LiteralControl("<h3>" + "Value: "));
   TextBox box = new TextBox();
   box.Text = "0";
   this.Controls.Add(box);
   this.Controls.Add(new LiteralControl("</h3>"));
  }
 }
}
```

35. ASP.NET პაკეტში მდგომარეობათა ტიპები. დანართებისა და სეანსების მდგომარეობები (ლაბ.13)

HTTP პროტოკოლი არ იძლევა ვებ გვერდების მდგომარებობის შენახვის საშუალებას, რადგან გვერდების ყოველ ახალ მოთხოვნაზე სრულდება მონაცემების ახლიდან დამუშავება და გადაგზავნა, ამის შემდეგ სერვერზე არსებული ყველა მონაცემი იკარგება. მაგრამ არსებობს საჭიროება, რომ ხდებოდეს მდგომარეობის შენახვა გვერდების გამოძახებებს შორის.

ASP.NET ვებ აპლიკაციებში არსებობს მდგომარეობებსი შენახვის სხვადასხვა ხერხი.

აპლიკაციის მდგომარეობა

განვიხილოთ აპლიკაციის მდგომარეობა (Application State). აპლიკაციის მდგომარეობაში ინახება გლობალური პარამეტერების მნიშვნელობები. ძირითადად აპლიკაციის პარამეტერებისთვის მნიშვნელობების მინიჭება ხდება Application_OnStart მეთოდში Global.asax ფაილში, ხოლო მათი შემდგომი ცვლილება ხდება ცალკეულ ASP.NET გვერდებზე. აპლიკაციის ცვლადების სიცოცხლის ხანგრძლოივობა ტოლია აპლიკაციის სიცოცხლის ხანგრძლივობისა.

// ფაილი Global.asax:

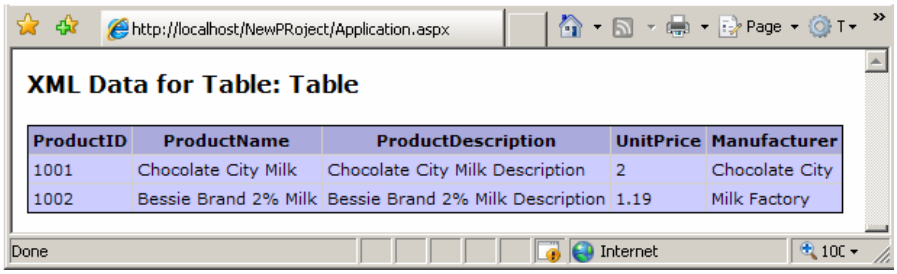
```
<%@ Import Namespace="System.Data" %>
```

```
<%@ Import Namespace="System.IO" %>
<script language="C#" runat="server">
    void Application_Start(Object sender, EventArgs e)
     {
        DataSet ds = new DataSet();
        FileStream fs = new
           FileStream(Server.MapPath("schemadata.xml"),
                  FileMode.Open,FileAccess.Read);
        StreamReader reader = new StreamReader(fs);
        ds.ReadXml(reader);
        fs.Close();
        DataView view = new DataView(ds.Tables[0]);
        Application["Source"] = view;
     }
</script>

//ფაილი Application.aspx:
<%@ Import Namespace="System.Data" %>
<html>
  <script language="C#" runat="server">
    void Page_Load(Object Src, EventArgs E )
     {
      DataView Source = (DataView)(Application["Source"]);
      MySpan.Controls.Add(new
                  LiteralControl(Source.Table.TableName));
      MyGridView.DataSource = Source;
      MyGridView.DataBind();
     }
  </script>
<body>
<form runat="server">
  <h3><font face="Verdana">Reading Data in
                       Application_OnStart</font></h3>
  <h4><font face="Verdana">XML Data for Table:
<asp:PlaceHolder runat="server" id="MySpan"/></font></h4>
<asp:GridView id="MyGridView" runat="server"
    Width="900"
    BackColor="#ccccff"
    BorderColor="black"
    ShowFooter="false"
    CellPadding=3
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#aaaadd"
```

```
    EnableViewState="false"
  />

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

//ფაილი SchemaData.xml:

```
<NewDataSet>
  <Table>
    <ProductID>1001</ProductID>
    <CategoryID>1</CategoryID>
    <ProductName>Chocolate City Milk</ProductName>
    <ProductDescription>Chocolate City Milk
                        Description</ProductDescription>
    <UnitPrice>2</UnitPrice>
    <ImagePath>images/milk5.gif</ImagePath>
    <Manufacturer>Chocolate City</Manufacturer>
  </Table>
  <Table>
    <ProductID>1002</ProductID>
    <CategoryID>1</CategoryID>
    <ProductName>Bessie Brand 2% Milk</ProductName>
    <ProductDescription>Bessie Brand 2% Milk
                        Description</ProductDescription>
    <UnitPrice>1.19</UnitPrice>
    <ImagePath>images/milk1.gif</ImagePath>
    <Manufacturer>Milk Factory</Manufacturer>
  </Table>
</NewDataSet>
```



ნახ.3.14

სესიის მდგომარეობა:

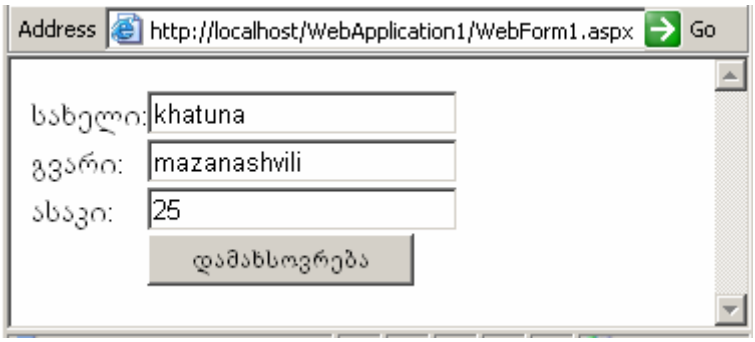
მომხმარებლის სესიის განმავლობაში მნიშვნელობების შენახვისათვის გამოიყენება სესიის ცვლადებში. ეს ცვლადები არის ყველა მომხმარებლისთვის უნიკალური. აღსანიშნავია რომ სესიის ცვლადების მნიშვნელობები იკარგება სესიის დასრულებისთანავე.

სესიის ცვლადებისთვი მნიშვნელობების მინიჭება აგრეთვე შესაძლებელია სესიის ინიციალიზების დროს Global.asax ფაილში.

```
protected void Session_Start(Object sender, EventArgs e)
{
       Session["FirstName"] = "khatuna";
       Session["LastName"] = "mazanashvili";
       Session["Age"] = 25;
}
```

ASP.NET გვერდზე შესაძლებელია სესიის ცვლადების გამოტანა `Page_Load` მეთოდში:

```
private void Page_Load(object sender, System.EventArgs e)
{
       if(!IsPostBack)
       {
              txtFirstname.Text =
(string)Session["FirstName"];
              txtLastname.Text = (string)Session["LastName"];
              txtAge.Text = Session["Age"].ToString();
       }
}
```



ნახ.3.15

შესაძლებელია სესიის მდგომარეობაში დამახსოვრება:

ფრაგმენტი WebForm1.aspx.cs ფაილიდან:

```
private void btnSave_Click(object sender, System.EventArgs e)
  {    Session["FirstName"] = txtFirstname.Text;
       Session["LastName"] = txtLastname.Text;
       Session["Age"] = Int32.Parse(txtAge.Text);  }
```

// ფაილი WebForm1.aspx:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false" Inherits="WebApplication1.WebForm1"
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
 <HEAD>
  <title>WebForm1</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <TABLE cellSpacing="0" cellPadding="0" border="0">
    <TR>
     <TD>სახელი:</TD>
     <TD>
      <asp:TextBox id="txtFirstname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>გვარი:</TD>
     <TD>
      <asp:TextBox id="txtLastname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>ასაკი:</TD>
     <TD>
      <asp:TextBox id="txtAge"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD></TD>
     <TD>
      <asp:Button id="btnSave" runat="server"
Text="დამახსოვრება"></asp:Button></TD>
    </TR>
   </TABLE>
```

```
  </form>
 </body>
</HTML>
```

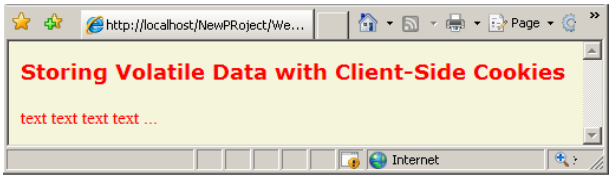
Cookies გამოყენება

Cookies არის ფაილი რომელიც იქმნება კლიენტის კომპიუტერზე და გადაეცემა ვებ სერვერს სხვადსხვა გვერდების გამოძახებისას. მასში შესაძლებელია სხვადსხვა ინფორმაციის შენახვა, თუმცა მისი მაქსიმალური ზომა შეზღუდულია 4096 კილობაიტამდე.

შემდეგ მაგალითში გვერდის ჩატვირთვისას მოწმდება გადმოეცა თუ არა კლიენტისგან ქუქი სახელით preferences1. თუ არ არის გადმოცემულიმაშინ იქმნება ახალი ქუქი და გადაეგზავნება კლიენტის კომპიუტერს. იგივე გვერდზე გამოიყენება GetStyle ფუნქცია, რომელიც აბრუნებს ქუქიში დამახსოვრებულ ცალკეულ მნიშვნელობებს. ეს ფუნქცია გამოიყენება ვებ გვერდზე ფონტის სახელის, ფონის ფერის და ტექსტის ფერის დასაყენებლად.

```
<html>
 <script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs E)
    { if (Request.Cookies["preferences1"] == null)
       { HttpCookie cookie = new HttpCookie("preferences1");
         cookie.Values.Add("ForeColor","red");
         cookie.Values.Add("BackColor","beige");
         cookie.Values.Add("FontName","Verdana");
         Response.Cookies.Add(cookie);
        }
      }
    protected String GetStyle(String key)
     {
        HttpCookie cookie = Request.Cookies["preferences1"];
        if (cookie != null)
         { switch (key)
            { case "ForeColor" :
                          return cookie.Values["ForeColor"];
                          break;
              case "BackColor" :
                          return cookie.Values["BackColor"];
                          break;
              case "FontName" :
                          return cookie.Values["FontName"];
                          break;
            }
```

```
        }
        return "";
      }
 </script>
 <style>
    body {
          font: <%=GetStyle("FontName")%>;
           background-color: <%=GetStyle("BackColor")%>;
         }
    </style>
 <body style="color:<%=GetStyle("ForeColor")%>">
  <h3><font face="Verdana">Storing Volatile Data with Client-
                           Side Cookies</font></h3>
   text text text text ...<br>
 </body>
</html>
```



ნახ.3.16

ViewState-ის გამოყენება

ASP.NET-ში შესაძლებელია რომ თითოელმა კონტროლმა შეინახოს მისი შიგა მდგომარეობა გვერდების გამოძახებებს შორის ViewState-ის გამოყენებით. ViewState-ის საშუალებით ხდება ვებ-გვერდის გამოძახებებს შორის მონაცემების დამახსოვრება.

შემდეგ მაგალითში, როდესაც მომხარებელი დააჭერს ღილაკს, ვებგვერდის ViewState-ში შეინახება მნიშვნელობა ”Hello!”.

```
<html>
 <head>
  <script runat="server" language="C#">

 void Button1_Click(object sender, System.EventArgs e)
 {  ViewState["Text"] = "Hello!"; }
 void Page_Load(object sender, System.EventArgs e)
 { if(ViewState["Text"]!=null)
     lblText.Text = ViewState["Text"].ToString(); }

  </script>
```

```
 </head>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:Label Runat="server" ID="lblText"></asp:Label>
   <asp:Button id="Button1" runat="server"
OnClick="Button1_Click" Text="Button"></asp:Button>
  </form>
 </body>
</html>
```

3.6. Web-ის უსაფრთხოება ASP.NET-ში. სერვერის, კლიენტების, ფორმების და როლების აუთენტიფიკაცია. პაროლების შენახვა (ლაბ.14–15)

ვებ აპლიკაციებში ხშირად საჭიროა მომხმარებელთა იდენტიფიკაცია და მათთვის სხვადასხვა რესურსებზე წვდომის უფლების განსაზღვრა. ASP.NET-ში არსებობს აუთენტიფიკაციის რამდენიმე მეთოდი: Windows, Forms, Passport. ეს მოთოდები განისაზღვრება კონფიგურაციის ფაილში `authentication` ტეგის `mode` ატრიბუტის საშუალებით:

```
<configuration>
       <system.web>
              <authentication/>
       </system.web>
</configuration>
```

მეთოდები:

```
<authentication mode="Windows" />
<authentication mode="Forms" />
<authentication mode="Passport" />
<authentication mode="None" />
```

ვებ–აპლიკაციაზე მომხმარებლის წვდომის უფლებას განსაზღვრავს `authentication` ელემენტი, ხოლო აპლიკაციის გარკვეულ ნაწილებზე განისაზღვრება `authorization` ელემეტით: deny და allow ქველემენტების საშუალებით:

*** – განსაზღვრავს ყველა მომხმარებელს,**

? – ანონიმურ მომხმარებლებს.

მაგალითად, ანონიმურ მომხმარებელთათვის საიტზე წვდომის უფლების გასათიშად ვებ-საიტის კონფიგურაციის ფაილში ჩაიწერება:

```
<configuration>
 <system.web>
```

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

შესაძებელია ცალკეულ მომხმარებელზე და მომხმარებელთა ჯგუფებზე წვდომის უფლების მინიჭება. შემდეგი ჩანაწერი ნიშნავს, რომ წვდომის უფლება აქვთ `someone` და `Admins` ჯგუფში შემავალ მომხმარებლებს:

```
<authorization>
    <allow users="someone" />
    <allow roles="Admins" />
    <deny users="*" />
</authorization>
```

შესძლებელია აგრეთვე ცალკეულ ვებ-გვერდებზე წვდომის უფლებების დაყენებაც:

```
<location path="webpage.aspx">
 <authorization>
  <allow roles="managers" />
  <deny users="?" />
 </authorization>
</location>
```

წინა ფრაგმენტი გვიჩვენებს, რომ `webpage.aspx` წვდომის უფლება აქვთ მხოლოდ `managers` ჯგუფის მომხმარებლებს.

Forms მეთოდით მომხმარებელთა ავტორიზაციის დროს საჭიროა მომხმარებლის სარეგისტრაციო გვერდის მითითება:

```
<configuration>
 <system.web>
  <authentication mode="Forms">
   <forms name=".ASPXCOOKIE" loginUrl="login.aspx"
protection="All"
            timeout="30" path="/"></forms>
  </authentication>
 </system.web>
</configuration>
```

ვებ საიტზე მომხმარებლების ავტორიზაციის მაგალითი:

```
// ფაილი Web.config:
<configuration>
  <system.web>
      <authentication mode="Forms">
```

```
        <forms name=".ASPXUSERDEMO" loginUrl="login.aspx"
protection="All" timeout="60" />
      </authentication>
      <authorization>
        <deny users="?" />
      </authorization>
    <globalization requestEncoding="UTF-8"
responseEncoding="UTF-8" />
  </system.web>
</configuration>
```

// ფაილი Default.aspx:

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

   void Page_Load(Object Src, EventArgs E )
     { Welcome.Text = "Hello, " + User.Identity.Name;
     }
    void Signout_Click(Object sender, EventArgs E)
      { FormsAuthentication.SignOut();
        Response.Redirect("login.aspx");
      }

 </script>
 <body>
  <h3><font face="Verdana">Using Cookie
Authentication</font></h3>
  <form runat="server" ID="Form1">
   <h3><asp:label id="Welcome" runat="server" /></h3>
   <asp:button text="Signout" OnClick="Signout_Click"
runat="server" ID="Button1" NAME="Button1" />
  </form>
 </body>
</html>
```

// ფაილი Login.aspx:

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

    void Login_Click(Object sender, EventArgs E) {
//authenticate user: this samples accepts only one user with a
//name of someone@www.contoso.com and a password of 'password'
if((UserEmail.Value=="someone")&&(UserPass.Value=="password"))
  {
```

```
   FormsAuthentication.RedirectFromLoginPage(UserEmail.Value,
                            PersistCookie.Checked);
 }
else
  { Msg.Text = "Invalid Credentials: Please try again"; }
 }
 </script>
 <body>
    <form runat="server" ID="Form1">
   <h3><font face="Verdana">Login Page</font></h3>
   <table>
    <tr>
      <td>Email:</td>
      <td><input id="UserEmail" type="text" runat="server"
             NAME="UserEmail" /></td>
      <td
      </td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input id="UserPass" type="password" runat="server"
             NAME="UserPass" /></td>
      <td>
      </td>
    </tr>
    <tr>
     <td>Persistent Cookie:</td>
     <td><ASP:CheckBox id="PersistCookie" runat="server" />
     </td>
     <td></td>
    </tr>
   </table>
   <asp:button text="Login" OnClick="Login_Click"
           runat="server" ID="Button1" NAME="Button1" />
   <asp:Label id="Msg" ForeColor="red" Font-Name="Verdana"
           Font-Size="10" runat="server" />
  </form>
 </body>
</html>
```

ნახ.3.17

ლიტერატურა

1. სურგულაძე გ., თურქია ე., ბულია ი. Web-აპლიკაციების აგება (ASP&ADO.NET). სახელმძღვ. სტუ. თბ., 2009.

т.я Майоя Дж.я п #:я Искусствоя программирования.я Энц, клопед, яя программ, ста.я о ер.ся англ.,я "DiaSoft",я Со б.,я т00т.я

3. Robinson S., Cornes O., Glynn J., Harvey B., McQueen C., Moemeka J., Nagel C., Skinner M., Watson K. Professional C#. Bimingham, WroxPress, 2001.