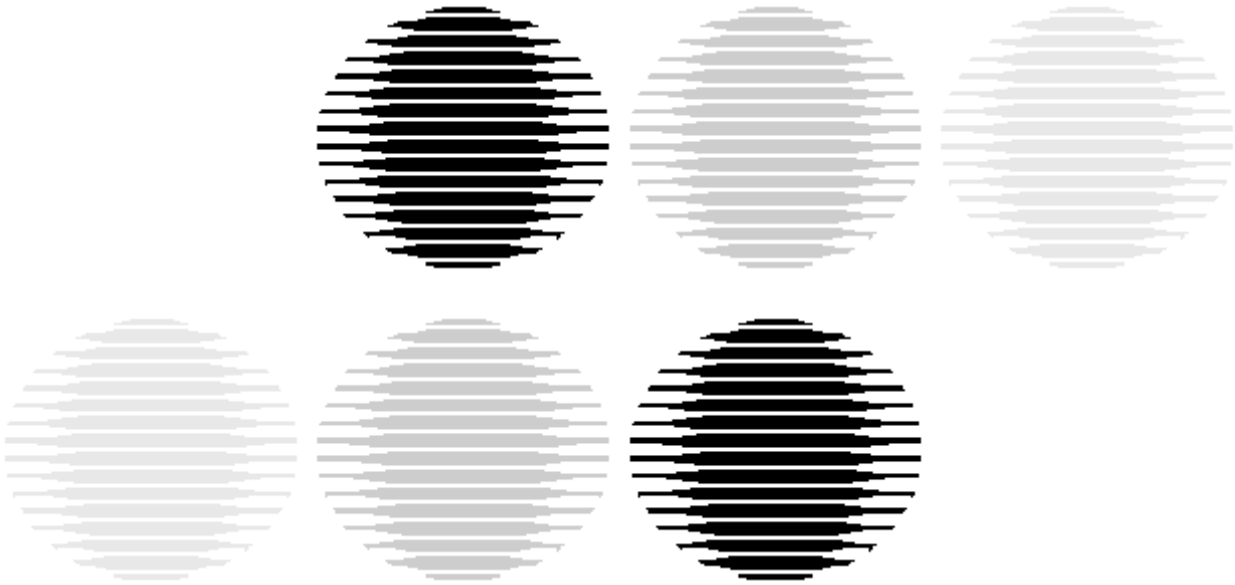
Curso ASP.NET por e-Learning



Relatório de projecto

990329 Paulo Silva

Junho de 2004

Autor do projecto

Paulo Alexandre Nogueira da Silva

Orientador do projecto

Dr. Carlos Miguel Miranda Vaz de Carvalho

Agradecimentos

Após o grande empenho neste projecto, desde o mês de Março deste ano, venho neste pequeno espaço agradecer a todos aqueles que foram uma verdadeira ajuda ao longo deste tempo.

Em primeiro lugar agradeço à minha namorada Sílvia Duarte, por todo o apoio.

À Ana Ferreira, nunca esquecerei a sua preciosa ajuda na correcção dos textos, o meu muito obrigado.

A todos aqueles que me apoiaram neste ultimo semestre para que os meus objectivos fossem alcançados.

A terminar agradeço ao meu orientador Dr. Carlos Vaz de Carvalho por ter "apadrinhado" este projecto.

Índice geral

1. Introdução

Resumo

Este trabalho foi realizado no âmbito da cadeira de projecto do 5º ano, durante o 2º semestre do ano lectivo 2003/2004.

O objectivo deste trabalho é apresentar os conteúdos a dar em várias aulas relativamente a um curso ASP.NET por e-learning.

O curso está dividido em módulos e esses módulos em aulas, que têm como objectivo transmitir conhecimentos de ASP.NET a alunos que já tenham alguma noção de programação e principalmente programação para a web. Ficando no final do curso aptos a desenvolver aplicações para a web com um nível de complexidade bastante razoável.

Esquema do relatório

O relatório encontra-se dividido em 5 partes:

- Introdução, pequena introdução ao relatório em si;
- *e-Learning* aplicado a este curso;
- Curso ASP.NET em modo de *e-Learning*, este é o capítulo central do relatório, contém o curso propriamente dito.
- Conclusão, do projecto.

Motivação

O que me levou a propor este tema para projecto foi o facto de ter um conhecimento avançado dos clássicos ASP, e estar interessado em adquirir conhecimentos em ASP.NET para ficar com bons conhecimentos no desenvolvimento de aplicações para a Web, uma área que me agrada bastante.

Metodologia

A realização deste trabalho teve como principal fonte as pesquisas na Internet, de forma a adquirir material referente ao tema. A selecção foi feita de forma a obter o melhor material de ASP.NET. Os meus conhecimentos em ASP também ajudaram à elaboração do projecto.

A elaboração do projecto pode ser dividida em três fases:

- Na primeira fase recolhi o material acerca do ASP.NET, exemplos, dicas, formas de criar aplicações, etc. Neste passo não fiz qualquer selecção.
- Na segunda fase analisei e seleccionei o melhor material, o material que de uma forma simples demonstrasse a matéria em estudo.
- Na terceira e última fase elaborei os conteúdos para o curso ASP.NET por e-learning.

2. Curso

Introdução

Este curso de ASP.NET por e-learning é constituído por seis módulos, num total de 24 aulas. No final, o aluno deverá compreender o funcionamento do .NET e a nova forma de criar aplicações web. Criar aplicações simples e elaboradas em ASP.NET.

Objectivos do curso

O objectivo deste curso é ensinar, a pessoas com conhecimentos prévios em ASP, ou a pessoas com conhecimentos de programação, a nova forma de criar aplicações Web com a nova versão do Microsoft Active Server Pages, o ASP.NET.

Público-alvo

Destina-se a pessoas com conhecimentos em ASP ou em programação.

Pré-requisitos

Para obter um aproveitamento máximo do curso, o aluno deverá ter conhecimentos em:

- Microsoft Visual Basic
- HTML
- Conhecimento de ASP

- Base de Dados e SQL

Este curso é ideal para quem preencher estes requisitos.

No entanto, todo o aluno que possua conhecimentos de programação poderá aproveitar o curso, embora com mais um pouco de esforço, possa atingir o aproveitamento máximo.

Metodologia de aprendizagem

O curso está dividido em seis módulos, num total de vinte e quatro aulas. O aluno terá a ajuda do professor, quando esta for solicitada.

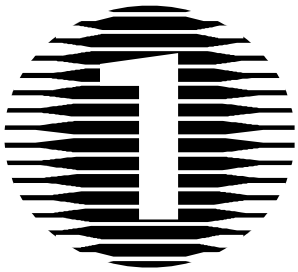
Requisitos técnicos

Software necessário para os exercícios:

- Windows 2000 ou Windows XP
- IIS V. 5.1 (Serviço de informação internet)
- Microsoft .NET Framework
- Web Matrix (Ferramenta gratuita da Microsoft)
- Microsoft Visual Studio .NET (preferencial)
- MSDN Library (opcional).

Hardware mínimo:

- Pentium II 300 MHz, 128 Mbytes de RAM, 1,5 GigaBytes de espaço livre no HD, Drive de CD-ROM, placa de som e acesso à Internet.



Módulo1 – Introdução ao ASP.NET

Objectivos do módulo

O objectivo deste primeiro módulo consiste em dar a conhecer as noções da nova forma de criar aplicações, o .NET e o .NET Framework que contém o ASP.NET, a sua importância, o seu funcionamento, as suas diferenças em relação ao ASP clássico, e o que é necessário para correr ASP.NET num PC.

Sumário

Aula 1 – Visão geral do Microsoft .Net e do .Net Framework

.NET

Para iniciar, vamos, então, saber o que é o .NET.

O .NET é uma tecnologia da Microsoft, permite a interligação de uma grande variedade de tecnologias, de dispositivos móveis, de servidores, e muitos outros dispositivos, permitindo aceder à informação, onde e sempre que for necessário.

O .NET possibilita que aplicações, novas ou já existentes, liguem os seus dados e transacções independentemente do sistema operativo, do tipo de computador ou do dispositivo móvel que esteja a ser utilizado, ou que linguagem de programação tenha sido utilizada na sua criação. Ou seja, permite o acesso a informações a qualquer hora, em qualquer lugar e em qualquer dispositivo.

Aqui ficou uma ideia do que é o .NET, mas poderá fazer uma pesquisa na internet para obter muito mais informação e ficar com uma ideia mais aprofundada do .NET.

.NET Framework

Do ponto de vista dos programadores, .NET Framework é o sistema operacional.

O .NET Framework é o ambiente que permite o desenvolvimento e a execução de aplicações .NET. O ambiente é dividido em duas partes principais, nomeadamente, a *common language runtime* (CLR), que permite compilar e executar diversas linguagens, desde que preparadas para este ambiente, sendo, algumas delas, o VB.NET e o C#.NET, entre diversas outras disponíveis no mercado; e uma biblioteca de classes, estruturada de forma hierárquica, que inclui um avançado sistema de páginas activas, o ASP.NET, um ambiente para construir aplicações Windows, o Windows Forms, e ainda um sistema de acesso a dados, o ADO.NET.

Uma característica interessante do .NET é a de que linguagens diferentes podem utilizar classes escritas noutras linguagens, fazendo por exemplo, herança das mesmas.

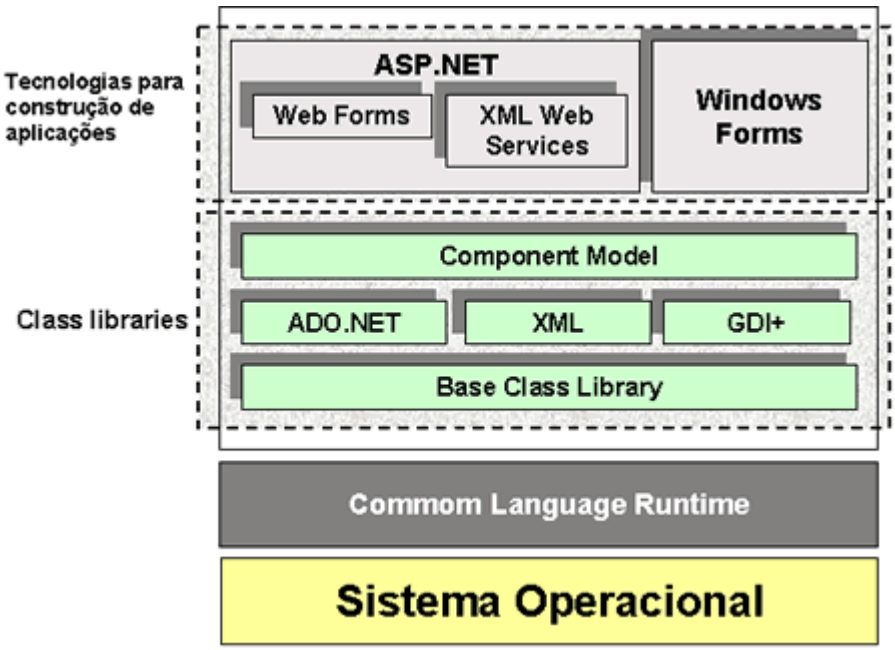


Imagem 1 – Framework

Dadas as necessidades e complexidade das aplicações actuais, existem centenas de classes disponibilizadas pelo .NET Framework que oferecem diversos “serviços”, como por exemplo:

- Acesso a base de dados;
- Conexões TCP/IP;
- Segurança;
- Desenho e impressão;

Usar as classes da biblioteca é algo simples. Em geral, basta criar um objecto da classe desejada e invocar os seus métodos e propriedades.

Vejamos um exemplo em C# para mostrar o conteúdo de um ficheiro na consola:

```
// Programa C# para mostrar um ficheiro na consola
// Este programa usa uma classe da .NET FrameWork

// Indica que "namespaces" estão a ser referenciados
using System;
using System.IO;
public class Class1
{
      public static int Main(string[] args)
      {
```

```
            // Cria objeto para ler ficheirod. Note que não é
necessário libertar o objeto criado
            StreamReader L = new StreamReader(@"c:\ficheiro.txt");
            // lê o ficheiro todo e coloca-o numa variável com um
método da classe
            string S = L.ReadToEnd();
            // Mostra na consola
            Console.WriteLine(S);
            return 0;
      }
}
```

Em conclusão, podemos dizer que o .NET Framework consiste numa biblioteca de classes que reúne todas as funções normalmente associadas ao sistema operativo, que facilita a criação de aplicações.

Aula 2 – Conceitos ASP.NET

O que é o ASP.NET?

ASP (Active Servers Page) é uma tecnologia que a Microsoft criou para permitir o desenvolvimento de páginas www de forma rápida, fácil e sem complicações.

Desde o seu aparecimento, houve um aumento significativo na quantidade dos programadores para www. Mas, como nem tudo é perfeito, o ASP, com o tempo, denunciou alguns pontos fracos que foram surgindo com o acréscimo da necessidade dos utilizadores e da exigência das aplicações.

Neste contexto, a Microsoft criou a ASP.NET, que não é apenas uma nova versão do ASP, mas sim, uma mudança profunda no modelo de programação do ASP, uma forma completamente diferente de construir aplicações Web.

Agora ficou tudo mais fácil na vida do programador, não somente na criação de páginas Web, mas, também, na criação de aplicações Web mais elaboradas – Web Services. Agora, o programador poderá desenvolver aplicações Web como desenvolve aplicações em Visual Basic para o Windows, ou seja, um modelo de programação "orientado a eventos". As aplicações construídas em ASP.NET são colocadas no Microsoft IIS e usam protocolos de internet como HTTP e SOAP.

Os benefícios desta nova forma de criar aplicações para a Web são inúmeros, nomeadamente:

- Orientação a objectos – As aplicações Web são escritas numa das linguagens suportadas pelo Framework.NET e essas são orientadas a objectos.

- Páginas compiladas – Após a página ter sido requisitada, o Framework verifica se essa página já foi compilada e, caso não tenha sido, compila só a primeira vez. Sendo assim, nas próximas requisições, a página não será compilada e a execução será muito mais rápida.

- Componentes – agora, tudo pode ser designado de componente, nomeadamente, Web Controls, Html Controls, User Controls, Custom Controls e outros Controls complexos que só o ASP.NET tem.

- Suporte do Framework.NET – Como o ASP.NET é do Framework, para além de suportar as classes do ASP.NET, suporta todas as classes do Framework, pelo que, a maior parte dos métodos de que necessitamos já estão feitos.

- Configuração da aplicação – Toda a configuração da aplicação é feita através de um ficheiro XML. Sendo assim, não é necessário recompilar a aplicação após uma eventual mudança, pois o Framework faz isso automaticamente.

O que é necessário para começar a programar em ASP.NET?

Agora que já possui algumas noções sobre ASP.NET e a nova forma de desenvolver aplicações Web, passemos à prática, vejamos o que é necessário para começar a programar em ASP.NET.

O material mais importante é um computador. Esse computador terá de ter como sistema operativo, o Windows 2000, XP ou NT. (De realçar o facto que apenas nestes sistemas operativos corremos aplicações ASP.NET.); ter instalado o IIS (Internet Information Services) e o .NET Framework, sendo apenas necessários na máquina onde a aplicação vai estar hospedada e na máquina de desenvolvimento; ter uma ferramenta de desenvolvimento, como por exemplo o Visual Studio. NET 2003, uma ferramenta poderosíssima para o desenvolvimento de aplicações e não, apenas, o ASP.NET. O WebMatrix é uma ferramenta gratuita para o desenvolvimento fácil de aplicações ASP.NET, mas também poderá usar o simples NotePad, embora este exija conhecimentos profundos nesta área. Poderá também utilizar outra ferramenta qualquer do mercado.

Agora, já possui os conhecimentos necessários para criar as suas aplicações ASP.NET.

Processamento de uma página ASP.NET

Como já foi dito, anteriormente, as páginas ASP.NET são compiladas. E o que é que isto significa?

Para entender isto, devemos reparar na plataforma .NET.

Na verdade, a grande jogada da Microsoft é a plataforma .NET, sendo apresentada como uma nova plataforma sobre a qual podemos desenvolver os nossos sistemas voltados para um ambiente distribuído via WEB.

O .NET Framework, tal como foi referido na primeira aula, é a base da plataforma .NET, o ambiente onde podemos criar e executar nossas aplicações quer sejam elas aplicações Web , VB , C#.

Na criação de uma página ASP.NET, ao ser invocada pela primeira vez, o código é compilado para um código intermédio, chamado MSIL *(Microsoft Intermediate Language)* não fazendo a distinção da linguagem usada na criação da página.

Após o código MSI ter sido criado é entregue ao .NET Framework que fará a conversão para a linguagem binária e em seguida executa o código. Esta conversão é feita pelo CLR *(Common Language Runtime)* que executa a gestão de todos os serviços necessários, memória, tipos de dados, código, etc.

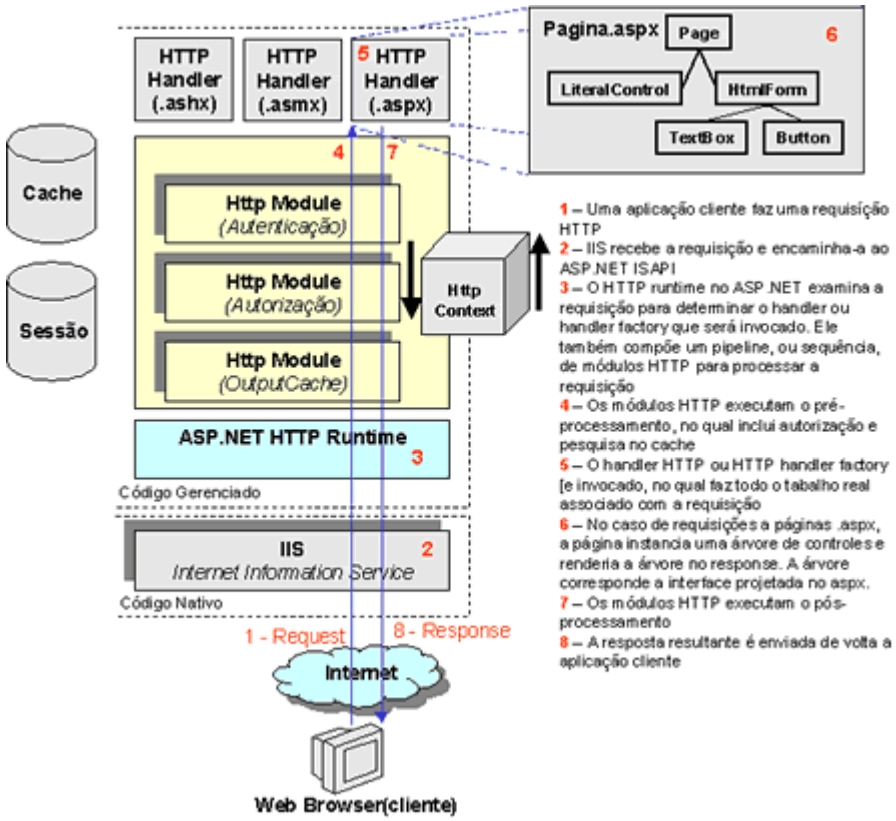


Imagem 2 – Processamento na invocação de uma página ASP.NET

Todo este processo é transparente para o utilizador.

Modelos de programação ASP.NET

As páginas ASP.NET poderão ser escritas em qualquer linguagem compatível com o .NET, embora as mais usadas sejam o VB.NET e C#.

A linguagem utilizada é identificada em cada página, sendo esse identificador usado pelo Framework para compilar a página, que, como já verificamos, anteriormente, é apenas compilada na primeira invocação.

No ASP.NET existem dois modelos de programação: Code Behind e Code in Page.

No modelo Code Behind, encontramos uma verdadeira separação do HTML e do código, que é o aconselhado, embora para principiantes seja mais adequado o modelo Code in Page.

Neste modelo, para cada ficheiro .aspx, extensão dos ficheiros ASP.NET, existe um ficheiro .aspx.vb (caso tenha escrito em VB) onde será escrito todo o código. No arquivo aspx, existirá, apenas, a parte HTML e a parte da declaração dos componentes do ASP.NET. O reaproveitamento de código neste modelo é enorme e facilita muito a programação.

No modelo Code in Page, as páginas .aspx ficam estruturadas de uma forma idêntica aos ASP clássicos, pelo que a diferença reside na existência de tags que não existem no ASP clássico. Estas tags permitem criar herança, implementar interfaces, importar classes, etc. Mesmo assim, com uma boa estruturação, a página fica muito mais legível do que o ASP clássico.

Uma página ASP.NET…

Para terminar esta segunda aula, vamos testar se o seu PC está apto a correr aplicações ASP.NET, ou seja, se tem instalado, como já referimos, o IIS e o .NET Framework. Comecemos, então, por um exemplo muito simples em HTML e a partir daí rumaremos à conquista de uma forma muito simples do ASP.NET.

Abra o simples NotePad, e insira o seguinte código HTML:

curso.html

```
<html>
<head>
<title>Curso ASP.NET</title>
</head>
<body>
      <p> Estou no curso de ASP.NET </p>
</body>
</html>
```

Grave com o nome de curso.html dentro do directório definido pelo IIS, que por defeito é “Inetpub\wwwroot”.

Agora abra um browser e invoque a página:

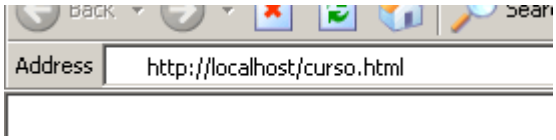


Imagem 3 – Invocação da página curso.html

Ao invocar temos “http”, que representa o protocolo usado, “localhost”, que neste caso, representa o computador local e, por último, o ficheiro que queremos correr.

Desta forma, estará a chamar a página que está no servidor.

Por outro lado, se abri-se directamente o ficheiro também funcionava, porque trata-se de um ficheiro html, embora o mesmo não aconteça se for um ficheiro ASP ou ASP.NET, pois estes têm de ser corridos num servidor (IIS).

Agora, vamos apenas alterar a extensão do ficheiro criado anteriormente para “curso.aspx”. Após ter efectuado este passo, abra o browser e invoque a página .aspx: http://localhost/curso.aspx. Se tudo correr bem, aparecerá o resultado esperado e ficará a saber que o seu computador encontra-se apto a correr aplicações ASP.NET.

A diferença entre as duas páginas está no servidor, que irá compilar a página .aspx, e no caso de termos alguma programação, essa será tida em conta. A partir daí jamais será compilada, a não ser que hajam alterações no código. Desta forma, teoricamente, as páginas ASP.NET são mais rápidas.

Alterámos, apenas, a extensão do ficheiro, se o .NET Framework não encontrar um código que perceba, passa adiante, e mostra o ficheiro sem alguma acção. Pode correr, e funcionar, mas no fundo não acontece nada de novo, por isso, vamos pôr um pouco de programação, ainda à maneira clássica dos ASP. Para isso vamos alterar o ficheiro .aspx, para o seguinte código:

curso.aspx

```
<%@ Page Language="VB" %>
<html>
<head>
<title>Curso ASP.NET</title>
</head>
<body>
```

```
      <p><%= "Olá Portugal!" %></p>
</body>
</html>
```

Isto é um ficheiro ASP.NET, mas só na sua extensão. Como teremos oportunidade de ver nas aulas do 2º módulo, existe uma nova maneira de resolver este problema no ASP.NET.

Invoque de novo a página no browser, o resultado é o mesmo, mas agora o trabalho do .NET Framework é um pouco diferente, pois já encontrou programação.

Aula 3 – Diferenças entre ASP e ASP.NET

Para finalizar este módulo praticamente teórico, vamos ver quais as diferenças entre os ASP clássicos e as novas ASP.NET.

Em primeiro lugar, temos agora o uso de classes, os recursos que podemos necessitar estão em Namespaces, uma espécie de pacotes tipo as bibliotecas do C++ e do Java. Existe, agora, a hipótese de criar as nossas soluções, para disponibilizar em várias formas, como aparelhos móveis, versões de HTML, Web services, etc.

Nas ASP.NET, o conceito de vários formulários deixa de existir, a partir de agora, uma página corresponde a um formulário, não sendo possível usar múltiplos formulários.

Linguagens compiladas, as versões clássicas das ASP são baseadas em scripting como o VBScript ou JScript. Estas sofrem de duas grandes lacunas: são interpretadas, e não são muito fortes nas funções base. As ASP.NET introduziram suporte para linguagens compiladas. Podemos usar além do Visual Basic e C++, a nova linguagem C#, ou outras externas, como o Java, Perl, Cobol, etc.

Com as ASP.NET, existe uma ligação permanente entre o servidor e os formulários, bastando, para isso, a utilização dos atributos runat=”server”.

Nesta aula, solicita-se a realização de uma pesquisa na internet para que possa conhecer melhor as diferenças entre ASP e ASP.NET.

Auto-avaliação do módulo 1

Para avaliar os seus conhecimentos neste módulo, responda ao seguinte formulário, assinalando, entre as várias opções, a mais correcta.

1. O que é o .NET?
    1. Sistema operativo
    2. Uma linguagem de programação
    3. Uma plataforma de software

2. O .NET Framework contém?
    1. Centenas de classes
    2. Um sistema operativo
    3. Uma só linguagem de programação

3. As páginas ASP.NET são compiladas para …
    1. um ficheiro executável
    2. um ficheiro de texto
    3. MSIL

4. O modelo de programação Code Behing …
    1. apresenta o código e o HTML num único ficheiro
    2. apresenta separação de código e HTML
    3. é idêntico ao ASP clássico

5. No ASP.NET temos …
    1. vários formulários
    2. um só formulário
    3. não podemos ter formulários

questionário.doc

Módulo 2 – Usando “Web Forms” e “Controls”

Objectivos do módulo

O objectivo deste segundo módulo consiste em ensinar a criar formulários ASP.NET, usar controlos, validar esses controlos e tratar eventos provenientes desses controlos. No fim deste módulo, o aluno já deverá ter entendido a nova forma de criar aplicações em ASP.NET.

Sumário

Aula 1 – O que são Web Forms e Controls?

Para iniciar este segundo módulo, no qual iremos falar de “*Web Forms*” e de controlos, vamos então saber o que são e qual a finalidade deste objectos.

O objectivo do ASP.NET é elevar o grau de abstracção no desenvolvimento de aplicações Web, estabelecendo uma analogia com as aplicações desktop. O programador manipula um conjunto de objectos que são responsáveis por se representarem.

O modelo de programação resultante oferecido pelo ASP.NET é designado de “*Web Forms”*. Esta abordagem permite o aumento da produtividade.

O modelo de programação dos “*Web Forms”* é construído a partir de componentes visuais designados por *“server-side controls”*. O seu nome indica que, apesar de se representarem através do HTML, enviado para o cliente, residem no servidor. Todos os *“server-side controls”* derivam de *“System.Web.UI.Control”* e são contentores de *“server-side controls”*.

Nas ASP.NET, o conceito de vários formulários deixa de existir, pelo que, uma página passa a ser um formulário, não sendo possível utilizar múltiplos formulários.

A resposta a um pedido é o resultado da representação em HTML da página. Em ASP.NET existem duas hierarquias de *“server-side controls”*, nomeadamente, a hierarquia de *“HtmlControls”* e a de *“WebControls”*.

HtmlControls

Estes tipos de controlos então organizados numa hierarquia que é utilizada quando se adiciona o atributo *“runat=server”* aos elementos HTML comuns, podendo este atributo ser acrescentado a qualquer elemento HTML que esteja na página ASP.NET.

Para cada elemento marcado com este atributo existe uma instância (do tipo que lhe corresponde na hierarquia) como campo da classe que implementa a página, pois agora cada página ASP.NET é uma classe. O nome destes campos corresponde ao valor do atributo *“id”* do elemento.

De seguida, vamos analisar um quadro que estabelece a associação entre o elemento e a classe.

Elemento	Classe
<input type="text" runat="server"/>	HtmlInputText
<input type="radio" runat="server"/>	HtmlInputRadioButton
<input type="checkbox" runat="server"/>	HtmlInputCheckBox
<form runat="server"> </form>	HtmlForm
<select runat="server"/>	HtmlSelect
	HtmlGenericControl
	HtmlGenericControl
<input type="button" runat="server"/>	HtmlInputButton
<input type="reset" runat="server"/>	HtmlInputButton
<input type="submit" runat="server"/>	HtmlInputButton

WebControls

Um dos grandes problemas, actualmente, é a compatibilidade entre browsers, das suas aplicações, e a complexidade das mesmas que têm de criar. É realmente um ponto difícil tentar a compatibilidade entre todos os browsers. Para isso, as ASP.NET introduzem o conceito dos *“WebControls”*, que englobam tarefas comuns e providenciam um modelo de programação bastante limpo. Estes controlos enviam puro HTML para o browser, não existe, de alguma forma, scripts do lado do cliente, e mesmo que existam por exemplo nos controlos de validação, estes só ficam em browsers onde funcionem, tudo automaticamente.

Os “WebControls” estão divididos em 5 famílias:

1. Controlos que “criam” os seus elementos HTML equivalente.
2. Lista de controlos, que providenciam o fluir dos dados pela página
3. Controlos ricos, que providenciam conteúdo rico para conteúdo de interface
4. Controlos de validação, para se poder validar campos em formulários
5. Controlos móveis, que englobam tags de WML para aparelhos móveis

A primeira família de controlos é similar aos controlos HTML. Os controlos de lista vão permitir uma fácil criação de listas e radio buttons, assim como checkboxes. Os controlos ricos consistem num calendário bastante flexível e no AdRotator. O controlo Calendar vai criar um output de puro HTML para browsers de baixo nível, ou DHTML para browsers de alto nível. Os controlos de validação vão disponibilizar formas simples para se construir validações nos formulários. Para além disso, poderá construir os seus próprios controlos, como veremos no Módulo 4 Aula 3.

Agora veremos alguns *“Web Controls”* que o ASP.NET disponibiliza. Esses controlos poderão ser criados automaticamente por um ambiente de desenvolvimento, como verificaremos mais tarde, mas, para já, para uma melhor compreensão, vamos entrar no “código”.

Label : <asp:Label id="Label1" runat="server">Label</asp:Label>
TextBox: <asp:TextBox runat="server"></asp:TextBox>
CheckBox: <asp:CheckBox runat="server"></asp:CheckBox>
Button: <asp:Button runat="server" Text="Button"></asp:Button>
DropDownList: <asp:DropDownList runat="server"></asp:DropDownList>
Image: <asp:Image runat="server"></asp:Image>
Table : <asp:Table runat="server"></asp:Table>
Hyperlink: <asp:HyperLink runat="server">/asp:HyperLink>

Para finalizar esta aula, vejamos algumas propriedades de alguns *“Web Controls”* como TextBox, Button, Label e Hyperlink.

- TextBox

Sintaxe	Propriedades
<asp:TextBox id="TextBox1" runat="server" AutoPostBack="True|False" Columns="caracteres" MaxLength="caracteres" Rows="rows" Text="Texto" TextMode="Single|Multiline|Password" Wrap="True|False" > </asp:TextBox>	**id - informa o nome do controle** **AutoPostBack - indica se o formulário será enviado via método Post caso ocorra um evento do tipo OnTextChanged (falaremos numa próxima aula)** **Columns - O tamanho do controle em termos de número de caracteres** **Rows - O número de linhas a exibir quando TextMode for igual a Multiline.** **Text - O texto inicial a ser exibido.** **TextMode - Define o comportamento do controle.Podemos ter os valores:** **- *SingleLine* - <input type="text">** **- *Multiline* - <input type="textarea">** **- *Password* - <input type="password">** **Wrap - indica se vai haver quebra de linha à medida que o utilizador digita.**

- Button

Sintaxe	Propriedades

<asp:Button
  id="Button1"
  runat="server"
  Text="Texto"
  Command="comando"
  CommandArgument="arg_comando"
>
</asp:Button>
	id - informa o nome do controle
	Text - O texto inicial a ser exibido.
	CommandName - O comando associado com o botão.
	CommandArgument - Fornece argumentos extras para o comando a ser executado.

- Label

Sintaxe	Propriedades

<asp:Label
  id="Label1"
  runat="server"
  Text="Texto"
>
</asp:Label>
	id - informa o nome do controle
	Text - O texto inicial a ser exibido. Em HTML o texto não permite edição.

- Hyperlink

Sintaxe	Propriedades

<asp:Hyperlink
  id="Hyperlink1"
  runat="server"
  NavigateUrl="url"
  Text="HyperlinkTexto"
  ImageUrl="url_imagem"
  Target="Window"
/>
</asp:Hyperlink>
	id - informa o nome do controle
	NavigateUrl - A URL para o qual o utilizador será direccionado.
	ImageUrl - Define uma imagem que será exibida.
	Text - O texto inicial a ser exibido.
	Target - Define em qual janela a URL definida deverá ser aberta.

Estes são alguns *“Web Controls”.* Se quiser saber mais, pesquise num ambiente de desenvolvimento ASP.NET como o WebMatrix.

Aula 2 – Criação de Web Forms

Agora que já sabemos o que são *“Web Forms”* e Controlos, passaremos à prática. Vamos construir o nosso primeiro formulário, utilizando, para esse efeito, o WebMatrix que poderá encontrar de uma forma gratuita em www.asp.net.

Em primeiro lugar, vamos abrir o WebMatrix. Após a abertura aparece uma caixa de diálogo, que nos pede os dados necessários para a criação de um ficheiro .aspx ou de outro tipo.

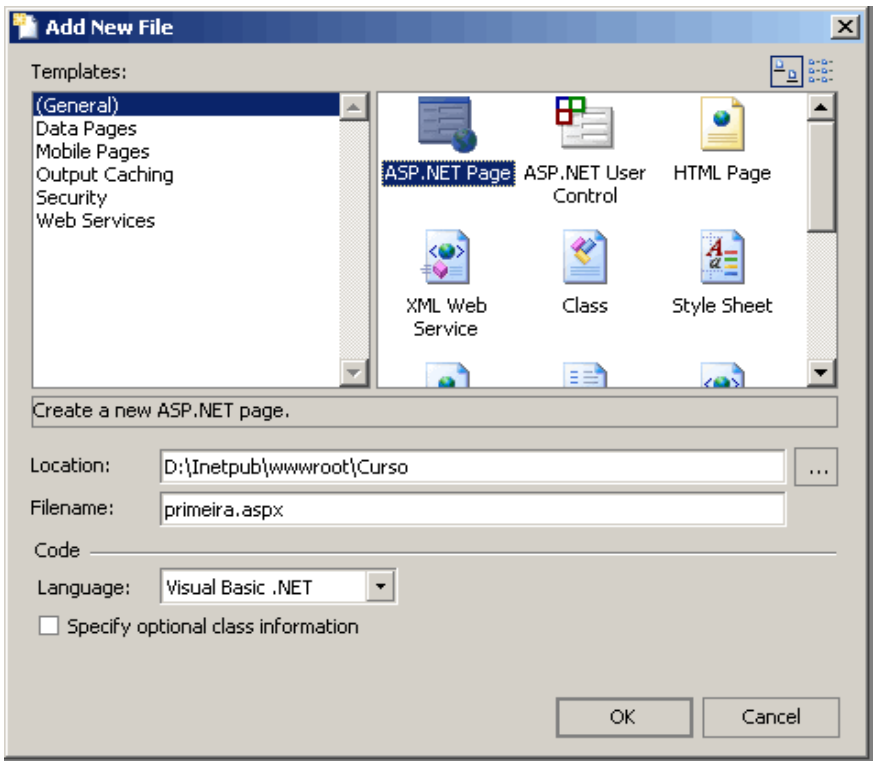


Imagem 4- Caixa de diálogo inicial do WebMatrix

Então, vamos criar uma página ASP.NET, um ficheiro .aspx, seleccionando “ASP.NET Page”, em “Location”, definimos o caminho para a pasta definida pelo IIS, que normalmente é “Inetpub\wwwroot\” mas que poderá ser alterada nas propriedades do IIS. No nosso exemplo, vamos colocar o ficheiro em “D:\Inetpub\wwwroot\Curso”, de seguida damos o nome ao ficheiro, por exemplo, primeira.aspx, e por fim, seleccionamos uma das linguagens para a criação da página, neste caso, “Visual Basic .NET”. Finalmente, é só clicar em “OK” e o ficheiro é criado na pasta indicada.

É claro que não vamos explicar as funcionalidades do WebMatrix, isso ficaria para um outro curso, mas o que será necessário será explicado.

Na parte inferior, encontram-se umas etiquetas. Seleccione a etiqueta “All”.

Imagem 5 – Etiquetas das várias visões do WebMatrix

Nesta visão, poderá ver-se todo o código e o HTML, o WebMatrix já criou a estrutura da página .aspx, e é de notar que o modelo de programação usado é o Code in Page, ou seja o código e o HTML estão no mesmo ficheiro.

Então, vamos, para já, analisar a estrutura que o WebMatrix criou.

```
<%@ Page Language="VB" %>
<script runat="server">

    ' Insert page code here
    '
</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <!-- Insert content here -->
    </form>
</body>
</html>
```

Como podemos verificar, na primeira linha está identificada a linguagem utilizada, neste caso, “VB”, entre as tags `<script runat="server"> </script>` é a onde vai ficar o código da página, de reparar na expressão `runat="server”` que significa que o código vai ser executado no servidor.

A seguir, temos o normal HTML, de notar a expressão `<form runat="server">`, é aqui que se inicia o formulário da nossa página. Como já tivemos oportunidade de verificar, só pode existir um formulário por página, ou seja, a página é um formulário.

Agora vamos adicionar à nossa página vários *“Web Controls”*, duas caixas de texto, um botão e um rótulo.

Para adicionar as caixas de texto (TextBox), adicionamos as seguintes tags à nossa página entre as tags `<form runat="server"> </form>`:

```
<asp:TextBox id="caixatexto1" runat="server"></asp:TextBox>
<asp:TextBox id="caixatexto2" runat="server"></asp:TextBox>
```

Para adicionar o botão e a Label vamos ao modo de “Design” do WebMatrix, como se vê, já lá estão as duas caixas de texto que adicionámos. Para adicionar o botão teremos que seleccionar na barra lateral esquerda, onde se encontram os “Web Controls” o Control “Button”:

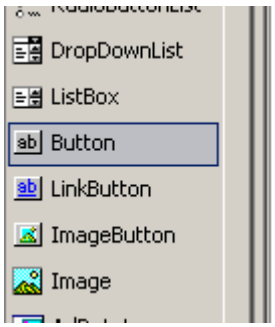


Imagem 6 – Control Button no WebMatrix

E de seguida vamos arrastar até ao local pretendido:

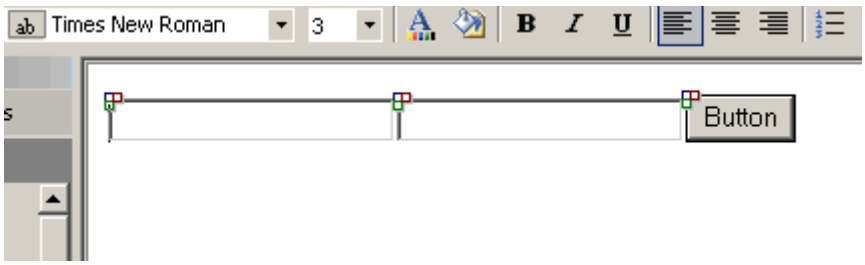


Imagem 7 – Formulário

Vamos alterar as propriedades do botão, para isso poderá ir à etiqueta HTML, ou poderá alterar na caixa de propriedades do WebMatrix, que é o que faremos agora. Vamos alterar as propriedades do nosso botão de acordo com a tabela que se segue, e adicionar uma Label (Etiqueta), para apresentar o resultado, da mesma forma como adicionamos o botão.

Objecto	Propriedade	Valor
Botão	ID	btSomar
	Text	Somar
Etiqueta	ID	lblResultado
	Text	

Assim, está concluída a criação dos elementos do nosso formulário. Como já repararam, o objectivo deste “Web Form” é somar dois números, mas isso fica para as próximas aulas, visto que, agora, o importante, é a criação do formulário.

Neste momento, o nosso formulário ficou:

Imagem 8- Formulário aspecto final

E o nosso HTML e código ficou:

```
<%@ Page Language="VB" %>
<script runat="server">

    ' Insert page code here
</script>
<html>
<head>
</head>
<body>
  <form runat="server">
   <asp:TextBox id="caixatexto1" runat="server"></asp:TextBox>
   <asp:TextBox id="caixatexto2" runat="server"></asp:TextBox>
   <asp:Button id="btnSomar" runat="server" Text="Somar"></asp:Button>
   <asp:Button id="btnSomar" </asp:Button>
   <asp:Label id="lblresultado" runat="server"></asp:Label>
  </form>
</body>
</html>
```

Invocando a nossa página num browser, podemos verificar que o formulário não reage a eventos, mas esse assunto ficará para uma próxima aula.

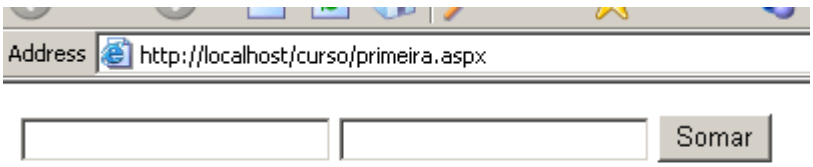


Imagem 9 - Formulário invocado no browser

Exercício proposto

Neste momento, já é capaz de criar um formulário ASP.NET, por isso, sugiro para terminar esta aula, reestruturar o formulário criado e alterar outras propriedades ao seu gosto, esperando que no final obtenha algo deste género:

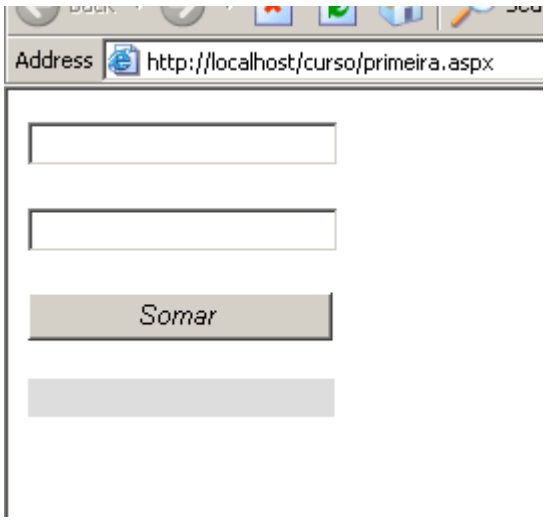


Imagem 10 – Formulário pretendido


Curso\primeira.aspx

Aula 3 – Usando controles para validação de formulários

No desenvolvimento de aplicações é necessário garantir que os dados inseridos pelo utilizador pertencem ao domínio esperado. A validação de dados é um tópico essencial em muitas aplicações Web, tais como formulários de autenticação, de entrada de valores e informações pessoais, só para citar alguns exemplos muito usados. A tarefa de validar dados pode ser muito trabalhosa, pois envolve validar dados tanto no lado do cliente como no lado do servidor.

Com os ASP.NET apareceram os controles de validação de formulários (*Validators*), pelo que, agora, o criador de aplicações ASP.NET não têm de preocupar-se com o código em “JavaScript” para validação de formulários, uma vez que, com estes novos controlos, o servidor “cria” todo o código “JavaScript”. Porém, essa validação não é só feita no cliente, também é feita no servidor, visto que o cliente não poderá suportar *script* ou o pedido poderá ter sido alterado. Uma grande vantagem no modelo de validação do ASP.NET é que não precisamos de saber onde ele será executado, se no servidor ou no browser, pois ele adapta-se ao tipo de browser que estiver a ser utilizado. Se o browser for incompatível, a validação será feita apenas no servidor.

Os controles ASP.NET para a validação de formulários são os seguintes:

- RequiredFieldValidator: Verifica se o campo está vazio;

```
<form runat="server">
   <p>
   <b>Nome : </b>
   <asp:Textbox id="cxNome" runat="server"></asp:Textbox>
   <asp:RequiredFieldValidator id="validacaonome" runat="server"
   display="Dynamic" ErrorMessage="Tem de introduzir o seu nome!"
   ControlToValidate="cxNome"></asp:RequiredFieldValidator>
</p>
   <p>
   <asp:Button id="bEnviar" runat="server" Text="Enviar"></asp:Button>
   </p>
</form>
```

Neste exemplo, usamos um controlo de validação para verificar se o campo “Nome” contém algum caracter, o controlo tem um identificador que identifica o

elemento ASP.NET **id="validacaonome"**, contém a expressão **runat="server"**, cujo significado já vimos anteriormente, a expressão **display="Dynamic"** que significa que a visualização da mensagem de erro é dinâmica, a expressão **ErrorMessage="Tem de introduzir o seu nome!"** que define a mensagem de erro e a expressão **ControlToValidate="cxNome"** que identifica qual o controlo a validar.

Ao invocar a página num browser IE e submeter o formulário com o campo “Nome”, vazio obtemos o seguinte resultado:

Imagem 11 – Demonstração do controlo de validação "RequiredFieldValidator"

- CompareValidator: Compara o valor de um campo com um determinado valor ou com o valor de outro campo;

```
<form runat="server">
  <p>
    <b>Senha</b>
    <asp:textbox id="senha1" runat="server"/>
  </p>
  <p>
    <b>Confirmar senha</b>
    <asp:textbox id="senha2" runat="server"/>
  </p>
  <p>
    <asp:CompareValidator id="ValidarSenha" runat="server"
    ErrorMessage="Senhas não coincidentes" ControlToValidate="senha1"
    ControlToCompare="senha2"/>
  </p>
  <p>
    <asp:button id="bValidar" runat="server" text="Validar"/>
  </p>
</form>
```

Neste exemplo, usamos o controlo de validação para verificar se o conteúdo de dois campos coincide. A expressão **ControlToCompare="senha2"**, identifica o controlo a ser comparado com o controlo que está a ser validado.

- RangeValidator: Verifica se o valor de um campo está entre uma determinada frequência de dois valores;

```
<form runat="server">
 <p>
    <b>Idade</b>
    <asp:textbox id="cxidade" runat="server"/>
 </p>
 <p>
   <asp:RangeValidator id="ValidarIdade" runat="server"
      ControlToValidate="cxIdade" Type="Integer"
      MinimumControl="8" MaximumControl="100"
      ErrorMessage = "Idade compreendida entre 8 e 100."
      Display="Dynamic"/>
 </p>
 <p>
    <asp:button id="bValidar" runat="server" text="Validar"/>
 </p>
</form>
```

Neste exemplo, usamos o controlo de validação para verificar se a idade introduzida pelo utilizador está compreendida em 8 e 100, a expressão **Type="Integer"** identifica o tipo de dados a comparar, que, neste caso, são dados do tipo inteiro, a expressão **MinimumControl="8"** define o valor mínimo aceite e a expressão **MaximumControl="100"** define o valor máximo.

- RegularExpressionValidator: Verifica o valor de um campo com uma determinada expressão regular;

```
<form runat="server">
<p>
   <b>Código Postal : </b>
   <asp:textbox id="cp" columns="60" runat="server"/>
   <asp:regularexpressionvalidator
     id="validacaocp"
     runat="server"
     validationexpression="\d{4}\-\d{3}"
     text="Código postal inválido (nnnn-nnn)!"
     controltovalidate="cp"/>
</p>
<p>
    <asp:button id="bValidar" text="validar" runat="server"/>
</p>
</form>
```

No exemplo de demonstração deste controlo utilizamos a validação do campo código postal, usando, para isso, uma propriedade deste controlo,

validationexpression="\d{4}\-\d{3}", que obriga o utilizador a introduzir uma sequência de quatro algarismos, um traço e mais uma sequência de 3 algarismos. Estas expressões de validação têm de ser definidas pelo criador da aplicação, consoante o campo a validar. Em seguida, apresentaremos mais algumas expressões de validação e os respectivos significados:

Expressão	Significado
"^\d {1,3}$"	Obriga a introdução de 1 a 3 dígitos
".+@.+\..+"	Validação de endereço de e-mail
"\d{1,6}"	Apenas dígitos <= 6

Agora, vamos ver um quadro com os símbolos e seus significados usados nas expressões regulares:

Símbolo	Significado
[. . .]	Aceita qualquer um dos caracteres delimitados por [ ]
[^...]	Não aceita nenhum dos caracteres delimitados por [ ]
\w	Aceita qualquer caracter alfanumérico.
\W	Aceita qualquer caracter separador.
\s	Não aceita nenhum caracter separador.
\d	Aceita qualquer dígito.
\D	Não aceita nenhum dígito.
(...)	Agrupa itens numa unidade
{n,m}	Aceita entre n e m ocorrências do item anterior
{n, }	Aceita n ou mais ocorrências do item anterior
?	Aceita zero ou uma ocorrências do item anterior
+	Aceita uma ou mais ocorrências do item anterior
*	Aceita zero ou mais ocorrências do item anterior

- CustomValidator: Permite criar um controlo de validação personalizado;

- ValidationSummary: Permite a exibição de um resumo de todas as validações feitas na página.

  No WebMatrix, estes controlos encontram-se na barra de “Web Controls”.

Imagem 12 – Controles de validação no WebMatrix

Para verificar se a página respeitou todas as validações, no nosso código podemos chamar a propriedade IsValid do objecto Page, como no exemplo:

```
If Page.IsValid Then
      ‘código se a página for válida
End If
```

Exercício proposto

Agora que já entendeu o funcionamento dos controles de validação, sugiro que abra a página já criada na aula anterior, e adicione controles de validação para validar os seguintes dados:

Os dois campos não poderão estar vazios;

Os valores dos dois campos têm de ser numéricos com 4 dígitos;

Os valores dos dois campos não poderão ser iguais;

Use um controle para exibir as validações feitas na página.

 Curso\primeira.aspx

Aula 4 – Tratamento de eventos

Até agora, temos visto como adicionar controlos às nossas páginas, mas as páginas não têm qualquer reacção, ou seja, são completamente estáticas. Para que as nossas aplicações ASP.NET produzam alguma reacção, temos que colocar a aplicação a reagir a eventos, tal como no Visual Basic. As ASP.NET foram feitas com um pensamento de orientação por objectos, isto é, no topo da árvore temos um objecto, que é o objecto Page. Cada controlo ASP.NET, application, e página são descendentes de Page. Isto significa que cada página é uma instância do objecto Page. Isto é importante a partir do momento em que nos apercebemos que o evento “Load” de Page é algo a que podemos responder. Durante a interacção com o utilizador, outros eventos são criados, e assim o evento “UnLoad” é disparado quando se sai da página.

Vejamos, como primeiro exemplo, como uma página reage ao evento “Load”, mostrando, para isso, a data e hora actual em formato Português quando a página é “aberta”.

O primeiro passo é adicionar uma Label à nossa página com as propriedades desejadas, será nesta Label que aparecerá a data e hora actuais.

Para que possamos usufruir das definições, de dias, horas, entre outras, de Portugal, teremos que importar um Namespace chamado System.Globalization.

Os Namespaces são uma espécie de bibliotecas de recursos, como já tivemos oportunidade de ver no primeiro módulo.

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

Agora, temos de definir a função que responde ao evento “Load”. Como já vimos, o código é implementado entre as tags `<script runat="server"> </script>` a função a implementar é `Sub Page_Load()`, no caso de a nossa linguagem utilizada for VB.

Dentro de Page_Load vamos declarar uma variável do tipo DateTime, que ficará com a data e hora do momento no servidor.

```
Dim data As DateTime = DateTime.Now()
```

Assim, para que a data e a hora sejam visualizadas no formato Português, temos que declarar um objecto que ficará com a cultura pretendida, neste caso, a portuguesa.

```
Dim objCultura = New CultureInfo("pt-PT")
```

Por fim, vamos então aplicar o nosso objecto cultural à formatação da data e atribuir o valor ao Label criado.

```
lblDataHoras.Text = data.ToString("F", objCultura)
```

Para terminar, só temos que “fechar” a nossa função com `End Sub`.

Invocando a página no browser temos:

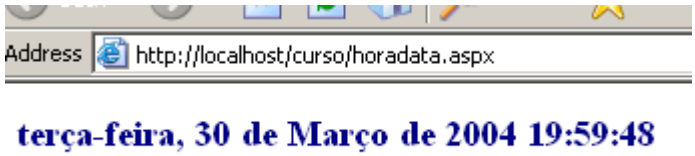


Imagem 13 – Data e hora actual

De seguida, mostraremos mais um exemplo, neste caso, a nossa página vai reagir ao clicar num botão.

Vamos criar uma página com os seguintes elementos “Web Controls”:

Objecto	Propriedade	Valor
Caixa de texto	ID	txtTexto
	Columns	80
Botão	ID	btConfirmar
	Text	Confirmar
Etiqueta	ID	lblResultado
	Font-Size	10
	Font-Bold	True
	Font-Names	Verdana
	Forecolor	Navy

Agora, temos que implementar uma função que execute quando se clicar no botão, ou seja quando ocorrer o evento Click no botão, essa função irá copiar o

conteúdo da caixa de texto para a etiqueta em maiúsculas e mencionar o número de caracteres do texto inserido na caixa de texto.

Então, vamos definir o cabeçalho da função:

```
Sub btConfirmar_Click (sender As Object, e As EventArgs)
```

Como se pode verificar, o nome da função contém o nome do objecto, neste caso, o botão e o evento a que ela responde. O primeiro parâmetro *sender* é o objecto que disparou o evento, e o segundo *e* é a lista de informações sobre o evento. Isso não é uma obrigação, mas um padrão adoptado pela maioria dos controles existentes.

Seguidamente, implementaremos o código que executará o que pedimos.

Em primeiro lugar, vamos transformar o texto da caixa de texto em maiúsculas, declarando, para isso, uma variável do tipo String que guardará esse resultado, e vamos também declarar uma variável do tipo Inteiro para guardar o número de caracteres do texto:

```
Dim texto as String
Dim ncaracteres as Integer
```

Agora, vamos transformar o texto em maiúsculas, utilizando, para isso, o método ToUpper() da classe String.

```
texto = txtTexto.Text.ToUpper()
```

De seguida, vamos contar o número de caracteres do texto, usando, para isso, o método Length() da classe String:

```
ncaracteres = texto.Length()
```

Por fim, teremos, apenas, que apresentar o resultado na Etiqueta. Para isso, atribuímos à propriedade Text da etiqueta o resultado:

```
lblResultado.Text = texto & " <br> Total de caracteres " & ncaracteres
```

Finalmente, fechamos a nossa função com `End Sub` e concluímos mais um exemplo que usa eventos.

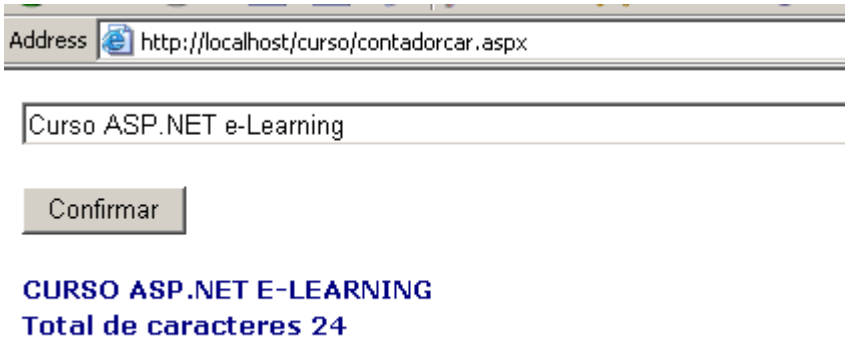


Imagem 14 – Resultado final exemplo eventos

Exercício proposto

Agora que já entendeu como se responder a eventos, sugiro como exercício, completar a página que tem vindo a ser criada nas últimas aulas, criar uma função para reagir ao botão e adicionar mais operações. Para tal, aconselho o uso de uma DropDownList.

	Curso\primeira.aspx

Aula 5 – Formulários autenticados

O ASP.NET, conforme mencionámos anteriormente, realiza, autonomamente, muitas tarefas que antes tínhamos que programar manualmente, sendo uma delas os controlos de validação, e outra, a tarefa e o controlo de login, ou de autenticação do utilizador, pelo que no ASP clássico os passos para realizar esta tarefa eram os seguintes:

1. Escolher o nome de uma variável de sessão (uma variável de sessão é uma variável que é reconhecida em todas as páginas da aplicação) e os valores que ela conterá (1 se o utilizador estiver autenticado, por exemplo);
2. Criar uma página para validar o login do utilizador;
3. Se o login for válido, atribuir 1 à variável de sessão escolhida;
4. Em todas as páginas, testar o valor da variável de sessão. Se o valor for diferente de 1 redireccionar o utilizador para a página de login;
5. Ao redireccionar para a página de login deve transmitir a url da página que o utilizador tentou aceder;
6. Após o login, a página de login deverá verificar se recebeu algum url por parâmetro e, se sim, deve redireccionar para essa página, permitindo assim que o utilizador chame directamente uma página;

Com o ASP.NET todo isto ficou mais fácil de implementar graças à classe *System.Web.Security*, pois, embora a estrutura de funcionamento seja a mesma, agora só temos de implementar alguns passos e o ASP.NET executa os restantes para nós.

Para que o ASP.NET possa realizar o processo de autenticação para nós, temos que informar que desejamos realizar esse processo. Essa informação é dada ao ASP.NET através de um ficheiro chamado *web.config*.

Este ficheiro é escrito em formato XML e pode ser inserido em qualquer directório da aplicação. O ficheiro contém configurações a nível de site que serão válidas a partir do directório em que ele se encontra e em todos os sub-directórios. Por isso, é comum que o ficheiro *web.config* seja sempre inserido na raiz da aplicação.

Através do web.config vamos fornecer as seguintes informações:

1. Que tipo de autenticação irá realizar-se, baseada em formulário, ou em outro tipo de autenticação, tal como a autenticação integrada com o sistema operacional, normalmente utilizada em ambientes de intranet;
2. Que formulário utilizaremos para realizar a autenticação;
3. Quais os utilizadores que poderão aceder às páginas. O mais comum é permitir o acesso a utilizadores autenticados;
4. O nome da variável que será utilizada para identificar o utilizador autenticado. Neste ponto, é importante destacar: o ASP.NET não utiliza variáveis de sessão, mas sim cookies (ficheiro que reside na máquina do utilizador); o nome indicado não é realmente o nome, mas um sufixo para o nome.

Neste momento já entendeu que o ASP.NET controlará o processo de autenticação automaticamente. A partir do momento em que configuramos o *web.config* para realizar a autenticação via formulário, o ASP.NET passa a verificar em todos os acessos à página se o utilizador está autenticado ou não, verificando se o cookie de autenticação existe. Se não existir, ele desviará a execução para o formulário de login que informamos no *web.config*, transmitindo via GET a página à qual o usuário tentou aceder.

Uma vez que, neste processo, usamos cookies ao invés de variáveis de sessão, o cookie poderia ficar gravado na máquina do utilizador entre uma visita e outra, e, realmente, pode, se for necessário. Existem cookies temporários e cookies permanentes, os temporários são destruídos quando o browser é fechado, os permanentes são gravados em disco. O ASP.NET permite definir se o cookie é temporário ou permanente.

Isso é útil para que se possa fornecer ao utilizador a opção de lembrar as suas informações de login, para que, quando o utilizador retornar ao site o cookie de login já esteja lá e, consequentemente, ele possa aceder às páginas sem introduzir as informações de login. É um processo muito cómodo e, uma vez que é apresentado como uma opção para o utilizador, ele próprio assume os riscos.

Em muitos ambientes de desenvolvimento como o Visual Studio .NET, ao criarmos uma nova aplicação ASP.NET, o ficheiro web.*config* é criado automaticamente, pelo que nós só temos que alterar algumas tags que ele possui.

Mas, para iniciar, vamos criar um ficheiro *web.config* manualmente.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <authentication mode="Forms">
            <forms name="Restrito" loginUrl="login.aspx"
protection="All" timeout="30" />
        </authentication>
    </system.web>
    <location path="/">
        <system.web>
            <authorization>
                <deny users="?" />
            </authorization>
        </system.web>
    </location>
</configuration>
```

Vamos analisar este simples ficheiro de configuração *web.config*. Na primeira linha identificamos a versão XML usada. Com o elemento authentication estamos a determinar qual o tipo de autenticação que iremos utilizar, neste caso, autenticação de formulários `mode="Forms"`, a url do formulário de autenticação `loginUrl="login.aspx"` e o nome do cookie de autenticação `name="Restrito"`, `protection="All"` identifica quais as páginas que necessitam de autenticação, neste caso, todas, `timeout="30"`, identifica o tempo de vida do cookie, e a tag `<location path="/">`, identifica o directório onde se podem encontrar as páginas que necessitam de autenticação para serem visualizadas

Já com o elemento authorization determinamos quais os utilizadores que podem aceder à aplicação. O ? utilizado no elemento Deny `<deny users="?" />` indica que deve ser negado o acesso a utilizadores não identificados, ou seja, não autenticados.

Agora, iremos ver melhor os parâmetros aceites pelo *authentication mode*, que, como já vimos define o tipo de autenticação, pelo que podemos ter:

- None – Não existe nenhuma forma de autenticação do utilizador;
- Windows – Identifica e autoriza os utilizadores baseando-se numa lista de utilizadores no servidor;
- Passport – Este tipo de autenticação é utilizado através dos servidores da Microsoft, de forma única e centralizada. O utilizador e a palavra-

chave são as mesmas que o cliente utiliza para aceder ao Hotmail e a outros sites da Microsoft;

- Form – Esta forma é a mais utilizada e a que deixa o programador mais livre para poder implementar o seu código de autenticação. Muitas vezes, esta forma está directamente ligada com uma base de dados (SQL Server, Oracle, etc);

Com esta nova maneira de autenticação temos o método RedirectFromLogin page do objecto FormsAuthentication. Esse método cria o cookie, identificando que o utilizador está autenticado e redirecciona o utilizador para a página que ele solicitou. Assim temos:

```
FormsAuthentication.RedirectFromLoginPage("Utl", chkLembrar.Checked)
```

Observe o segundo parâmetro, que determina se o cookie será permanente ou temporário. Neste exemplo, estamos a configurar o segundo parâmetro com o checked de uma checkbox inserida na página de login, para dar a tal opção já falada ao utilizador. Desta forma, o utilizador pode optar por manter o seu login registado no cookie ou não. Se não tivéssemos criado a checkbox na página, deveríamos utilizar false como default.

Exercício proposto

Para terminar a aula, sugiro que crie o ficheiro *web.config* demonstrado e coloque-o no directório que está a usar para este curso, crie um ficheiro muito simples dizendo apenas “Página de login”, grave com o nome login.aspx, e chame uma página que esteja dentro desse directório, como por exemplo a calculadora. Qual o resultado?

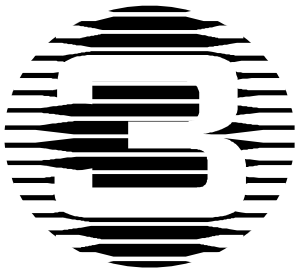
Auto-avaliação do módulo 2

Para avaliar os seus conhecimentos neste módulo proponho:

_ Crie um ficheiro login.aspx para autenticação de utilizadores e uma nova pasta no seu directório “Curso” chamada “Restrita”. Esta pasta passa agora a ser a nossa área restrita. Dentro dessa pasta crie um ficheiro default.aspx, que, para já, apresente as boas-vindas ao utilizador autenticado.

Para saber o nome do utilizador, após a autenticação, use HttpContext.Current.User.Identity.Name.

Curso\web.config
Curso\Restrita\login.aspx
Curso\Restrita\degault.aspx



Módulo 3 – Utilização do ADO.NET

Objectivos do módulo

O objectivo deste terceiro módulo é ensinar a utilizar a tecnologia ADO.NET para aceder a dados, dados estes, que se encontram em bases de dados, e outros, tornando as aplicações ASP.NET mais poderosas.

Sumário

Aula 1 – Visão geral do ADO.NET

Tal como o ASP.NET, também o ADO.NET é uma evolução natural para a nova plataforma da Microsoft do anterior ADO (ActiveX Data Objects). É através deste mecanismo que se processa o acesso aos registos em vários tipos de bases de dados nas aplicações .NET, como se de objectos se tratassem. O ADO.NET é a nova geração do conjunto de classes de acesso a dados feito sob medida para o universo .NET.

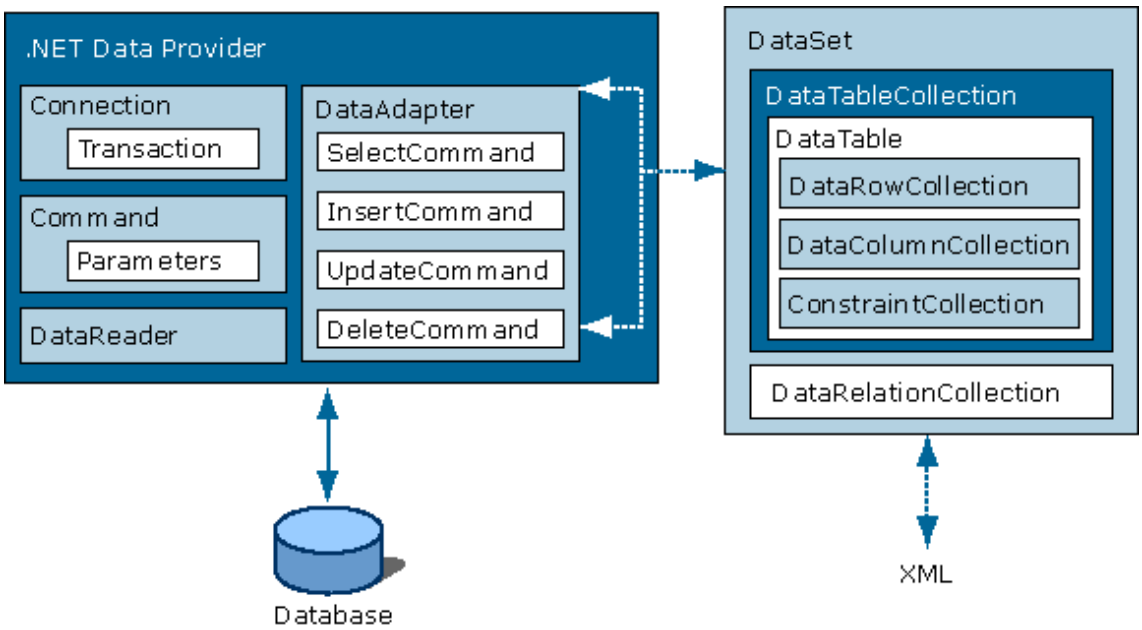


Imagem 15 – Arquitectura ADO.NET

O ADO.NET é composto de várias classes, a destacar:

- Connection – usado para criar a conexão entre o programa e a base de dados. Para estabelecer uma conexão com uma fonte de dados, o objecto Connection usa a propriedade ConnectionString que é a string de conexão que deverá ser informada para que a conexão seja efectivamente aberta;
- Command – usado para armazenar comandos SQL que são executados na base de dados (SELECT, INSERT, UPDATE, DELETE, Stored Procedures, etc). Os métodos usados para realizar estas tarefas são:

  - ExecuteReader – executa declarações SQL que retornam linhas de dados, tais como SELECT;
  - ExecuteNonQuery – executa declarações SQL que não retornam dados, tais como INSERT, UPDATE, DELETE e SET;
  - ExecuteScalar – retorna um valor único como resultado de uma função agregada: SUM, AVG, COUNT, MAX E MIN.
- DataReader – trata-se de um cursor read-only e forward-only com o objectivo de oferecer a máxima performance na busca de dados (útil e eficiente para preencher combos, listas e tabelas). Não oferece acesso desconectado e não permite alterar ou actualizar a fonte de dados original, sendo usado para obter, rapidamente, dados, apenas, de leitura. As propriedades e métodos mais usados dos objectos DataReader são:
  - FieldCount – informa o número de colunas da linha de dados actual;
  - IsClosed – Indica se o objeto DataReader está fechado;
  - RecordsAffected – especifica o número de linhas alteradas, excluídas ou incluídas na execução de uma declaração SQL;
  - Item (n) - obtêm o valor da n-ésima coluna no seu formato nativo;
  - Close – Método que fecha o objecto;
  - GetName – Método que retorna o nome da n-ésima coluna;
  - Read – método que permite ao DataReader avançar para o próximo registo;
  - IsDbNull – método que informa se a n-ésima coluna possui um valor nulo.
- DataAdapter – classe usada para buscar dados na base de dados e preencher uma instância DataSet ou DataTable;
- DataSet – Esta classe funciona como uma base de dados na memória da aplicação. Cria uma estrutura que permite armazenar dados de tabelas, colunas, registos e relacionamentos entre as tabelas. Os dados podem ser serializados em XML, alterados, excluídos, inseridos e sincronizados com a fonte original;
- DataTable – Destacam-se, ainda, as classes DataTable, hierarquicamente internas à classe DataSet. A classe DataTable armazena o resultado de um comando select ou stored procedure.

Aula 2 – Ligação a uma fonte de dados

Nesta aula, vamos aprender a estabelecer a ligação entre as nossas aplicações ASP.NET e uma fonte de dados, utilizando a tecnologia ADO.NET. As fontes de dados mais conhecidas são as bases de dados MS Access, MS Sql Server, Oracle entre outras, mas o ADO.NET também permite a ligação a uma folha de dados MS Excel e a outras fontes de dados que não sejam bases de dados. O .NET contém namespaces que nos permitem aceder a fontes de dados, nomeadamente:

· System.Data
· System.Data.ADO
· System.Data.Oledb
· System.Data.SQL
· System.Data.SQLTypes

Ligação a uma base de dados MS Access

Vamos então estabelecer uma conexão a uma base de dados Access. Criamos no MS Access uma base de dados simples, contendo apenas uma tabela, a tabela Pessoa, que guarda alguns dados relacionados com Pessoas e chamamos a esta base de dados pessoas.mdb:

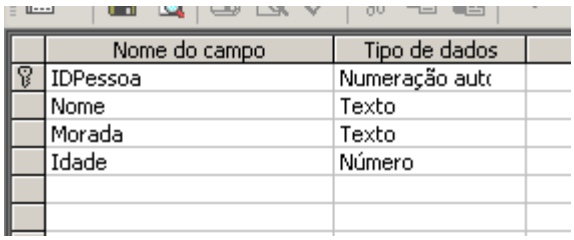


Imagem 16 – Tabela Pessoa da base de dados pessoas.mdb

Para fazer a conexão a esta base de dados MS Access temos de fazer duas importações de namespaces:

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<script language="VB" runat="server">

</script>
```

Após este primeiro passo, vamos então fazer a conexão propriamente dita. O código será criado no evento Page_Load, que é o primeiro evento que ocorre na página. O objecto que nos permite fazer a conexão com a base de dados é OleDbConnection. Além desse objecto, precisamos indicar onde se encontra a base de dados, de que tipo ela é, e mais algumas coisas. Para isso, usamos uma String de conexão que será passada ao objecto OleDbConnection. Depois, usamos o método Open do Objecto que instanciamos. Vejamos o código necessário para estabelecer uma conexão.

```
Sub Page_Load(sender as Object, e as EventArgs)

    Dim connString as String

    connString  =  "PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA  SOURCE="  &
Server.mappath(".\pessoas.mdb") & ";"

    Dim objConnection as OleDbConnection 'declara objecto para conexão
    objConnection = New OleDbConnection(connString) 'cria conexão
    objConnection.Open() 'abre a conexão
    objConnection.Close() 'Fecha a conexão à base de dados
End Sub
```

Criamos, assim, uma conexão a uma base de dados MS Access. Agora, só temos que fazer operações sobre os dados da base de dados, mas isso ficará para as próximas aulas deste módulo.

Ligação a uma base de dados Sql Server

A ligação a uma base de dados Sql Server segue o mecanismo anterior, mas com algumas diferenças, aqui também temos de importar dois namespaces:

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<script language="VB" runat="server">

</script>
```

Os namespaces a importar poderiam ser os mesmos que num acesso a uma base de dados MS Access mas é claro que não é possível ter acesso às funcionalidades do namespace System.Data.SqlClient.

Agora, tal como na conexão a uma base de dados MS Access, temos de criar um objecto que permita criar a conexão, neste caso, esse objecto é o SqlConnection, depois só temos de passar a esse objecto como parâmetro, o servidor Sql Server, a base de dados a conectar, o nome do utilizador e a palavra chave. Por fim, tal como no MS Access temos de usar o método Open do objecto que instanciamos.

```
Sub Page_Load(sender as Object, e as EventArgs)
    Dim connString as String

    connString = "server=localhost;database=bd;uid=xx; pwd=xx"

    Dim conexao As SqlConnection 'declara objecto para conexão
    conexao = New SqlConnection (connString) 'cria conexão
    conexao.Open()'abre a conexão
    objConnection.Close() 'Fecha a conexão à base de dados
End Sub
```

E assim terminamos esta aula dedicada à conexão a fontes de dados. Poderá pesquisar na Internet outras conexões, como por exemplo, a conexão a uma folha de dados MS Excel, entres outras.

Aula 3 – Operações sobre os dados de uma base de dados

Agora que já sabemos fazer uma conexão a uma base de dados, já podemos manipular os dados da base de dados conectada, ou seja, podemos ler, alterar, inserir e apagar os registos dessa base de dados.

Nesta aula, o aluno terá já de saber alguma coisa sobre base de dados e SQL, se tal não acontece, sugiro um breve estudo destas matérias antes de prosseguir.

No nosso curso, as bases de dados usadas são MS Access. Nesta aula, será usada a base de dados criada na aula anterior (pessoas.mdb) e será criada uma página basedados.aspx, onde serão acrescentadas funcionalidades ao longo da aula.

Ler dados de uma base de dados

Vamos agora ver como se lê dados de uma base de dados. O primeiro passo é fazer a conexão à base de dados como vimos na aula anterior, depois temos de definir o comando SQL (neste caso será um SELECT pois trata-se de uma leitura), e, por fim, executar esse comando e atribuir o resultado a um objecto DataReader. Vamos ler o Nome das pessoas que estão na base de dados e mostrar numa página.

Na página basedados.aspx, criamos o evento Page_Load e definimos os seguintes passos:

1. A conexão: é feita, como já vimos, a uma base de dados MS Access que está na raiz da aplicação e tem como nome pessoas.mdb.

```
Sub Page_Load(sender as Object, e as EventArgs)

    Dim connString as String

    connString  =  "PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA  SOURCE="  &
Server.mappath(".\pessoas.mdb") & ";"

    Dim objConnection as OleDbConnection 'declara objecto para conexão
    objConnection = New OleDbConnection(connString) 'cria conexão
    objConnection.Open() 'abre a conexão
```

2. O comando SQL: é definido o comando SQL para ler da base de dados o Nome da tabela Pessoa, é criado um objecto command para executar o comando SQL definido, utilizando a conexão criada.

```
    Dim strQuery as String = "SELECT Nome FROM Pessoa"

    'Cria um objeto Command
    Dim objCommand as OleDbCommand
    objCommand = New OleDbCommand(strQuery, objConnection)
```

3. Leitura dos dados: agora temos de criar um objecto do tipo OLEDbDataReader que guardará o resultado do comando SQL, executamos o comando e atribuímos o valor ao objecto OLEDbDataReader, depois lemos os valores com o método Read e a propriedade Item do objecto.

Para além da propriedade Item("nome do campo da base dados") para ler dados, existe um método para leitura de dados, esse método é o GetValue(n) que lê a coluna n do registo actual, n inicia em 0 que corresponde à primeira coluna. Existem muitos outros métodos no objecto OLEDbDataReader que merecem uma pesquisa.

```
    Dim nomes as OLEDbDataReader
    nomes = objCommand.ExecuteReader

    While nomes.read
      Response.write (nomes.Item("Nome") & "<br>")
    End While
```

4. Fecho: fechamos o objecto de leitura e a conexão.

```
      nomes.Close() 'fecha o datareader

      objConnection.Close() 'Fecha a conexão à base de dados
End Sub
```

E assim finalizamos a nossa consulta à base de dados, pelo que, agora só temos que invocar a página basedados.aspx num browser e temos o seguinte resultado:

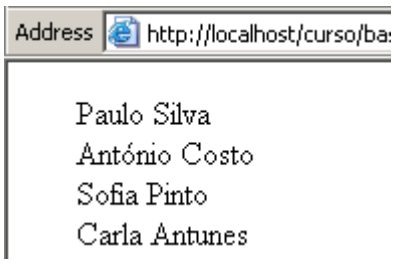


Imagem 17 – Resultado da consulta à base de dados

Inserir dados numa base de dados

Uma das operações importantes sobre uma base de dados é a inserção de novos dados nas aplicações WEB, o que poderá ser encontrado, quando um utilizador se regista, quando um visitante assina o livro de visitas, etc. Por isso, vamos agora ver como se inserem novos registos a uma base de dados MS Access. Vamos continuar a construir funcionalidades para a nossa página basedados.aspx.

1. A conexão: mais uma vez, tem que ser feita a conexão à base de dados. Como vamos utilizar a mesma base de dados, já não necessitamos de criar a conexão, pois já está criada no evento Page_Load, por isso, quando a página é aberta, a conexão é feita automaticamente, só temos de declarar o objecto da conexão como "global", para que possa ser acedido em qualquer parte da nossa página. Sugiro, também, criar um novo método que se responsabiliza pela conexão à base de dados, e um método que leia e visualize os dados, ou seja, vamos estruturar a nossa página para receber novas funcionalidades.

```
Dim objConnection as OleDbConnection 'objecto "GLOBAL"
```

```
Sub conexao(connString as String)

      objConnection = New OleDbConnection(connString)
      objConnection.Open() 'abre a conexao

End Sub
```

```
Sub consulta_nome()

      Dim strQuery as String = "SELECT Nome FROM Pessoa"

      'Cria um objeto Command
      Dim objCommand as OleDbCommand
      objCommand = New OleDbCommand(strQuery, objConnection)


      Dim nomes as OLEDbDataReader
      nomes = objCommand.ExecuteReader

      While nomes.read
         Response.write (nomes.Item("Nome") & "<br>")
      End While

      nomes.Close()'fecha o datareader

End Sub
```

```
Sub Page_Load(sender as Object, e as EventArgs)

      Dim connString as String

      connString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA SOURCE=" &
Server.mappath(".\pessoas.mdb") & ";"

      conexao(connString) 'Conexão

      If Not Page.IsPostBack Then

            consulta_nome() 'Consulta à base de dados

            objConnection.Close() 'Fecha a conexão à base de dados

      End If

End Sub
```

Como podemos constatar, a nossa página basedados.aspx ficou mais organizada, sendo, agora, mais fácil, adicionar novas funcionalidades e reaproveitar as já existentes.

Para aumentar um pouco o nível de dificuldade, vamos adicionar uma nova tabela à nossa base de dados. Essa tabela contém os grupos de sangue. Vamos adicionar um novo campo à tabela Pessoa, de modo a que esse campo referencie um grupo de sangue da tabela Grupo_Sangue.

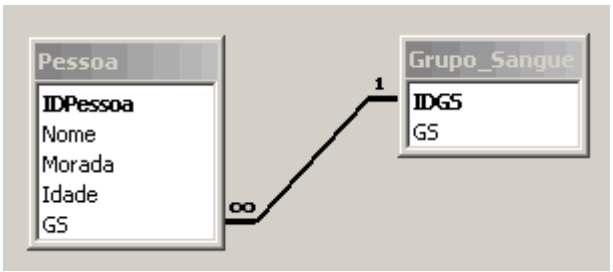


Imagem 18 – Relação entre tabelas da base de dados pessoas.mdb

2. O formulário para introduzir os dados: temos que criar um formulário para introduzir os dados que posteriormente serão adicionados à base de dados.

Temos que ter neste formulário, quatro campos, Nome, Morada, Idade e Grupo de Sangue. Para o campo Grupo de Sangue, use o controlo DropDownList.

Como este assunto já foi tratado, a construção do formulário ficará por sua conta, não se esqueça dos controles de validação para validar os dados introduzidos pelo utilizador.

3. Agora que já temos o formulário criado e validado, vamos criar uma função que reaja ao evento Click do botão “Inserir dados”:

```
Sub btInserir_Click(sender As Object, e As EventArgs)

End Sub
```

Aqui temos a função criada, agora só temos que adicionar o código para adicionar os dados do formulário à nossa base de dados:

```
Sub btInserir_Click(sender As Object, e As EventArgs)

    'Cria um objeto Command
    Dim objCommand as OleDbCommand

    Dim nome as String = txtNome.Text
    Dim morada as String = txtMorada.Text
    Dim idade as Byte = txtIdade.Text
    Dim gs as Byte = cint(DLGS.SelectedValue)

    Dim strQuery as String = "INSERT INTO Pessoa(Nome, Morada, Idade,
GS) VALUES ('"& Nome &"', '"& Morada &"'," & Idade &", " & gs &")"

    If Page.IsValid Then

       objCommand = New OleDbCommand(strQuery, objConnection)
       objCommand.ExecuteNonQuery()

       Response.Redirect("basedados.aspx") 'Para “Actualizar” a página

    End If

End Sub
```

Mais uma vez, temos que criar um objecto para executar o comando, depois instanciamos as variáveis nome, morada, idade e gs com os valores do formulário, criamos a String com o comando de inserção (INSERT) e, por fim, instanciamos o objecto de comando com o comando e o objecto de conexão e executamos. Se tudo correr bem, obtemos a função que insere dados na nossa base de dados.

filmes/insert.avi

Actualizar dados de uma base de dados

Agora, vamos construir a funcionalidade que permite actualizar os dados, para isso, em vez de listar os nomes das pessoas directamente na página vamos adicionar um controlo DropDownList:

```
<asp:DropDownList id="DLPessoas" runat="server"></asp:DropDownList>
```

E vamos alterar a função consulta_nome(), de modo a que preencha este controlo com o nome das pessoas:

```
While nomes.read

     Dim listaNomes As New ListItem()
     listaNomes.Text = nomes.Item("Nome")
     listaNomes.Value = nomes.Item("IDPessoa")
     DLPessoas.Items.Add(listaNomes)

End While
```

Para isso, definimos uma nova lista e atribuímos à propriedade Text o nome da pessoa e à propriedade Value o ID da pessoa, e adicionamos essa lista ao nosso controlo DropDownList.

Adicionamos um novo botão “Editar” e acrescentamos a propriedade `CausesValidation="False"` para que as regras de validação criadas não afectem este controlo e criamos a função que reage ao evento Click desse botão. Essa função vai identificar a pessoa seleccionada no controlo DropDownList e preencher os controlos já existentes com os dados dessa pessoa:

```
Sub btEditar_Click(sender As Object, e As EventArgs)

    Dim objCommand as OleDbCommand

    Dim IDPessoa as Long = cint(DLPessoas.SelectedValue)

    Dim strQuery as String = "SELECT * FROM Pessoa WHERE IDPessoa = "
& IDPessoa

    objCommand = New OleDbCommand(strQuery, objConnection)

    Dim pessoa as OLEDbDataReader
    pessoa = objCommand.ExecuteReader

    pessoa.read()

    txtNome.Text = pessoa.Item("Nome")
    txtMorada.Text = pessoa.Item("Morada")
```

```
    txtIdade.Text = pessoa.Item("Idade")

    DLGS.SelectedValue = pessoa.Item("GS")

End Sub
```

Agora temos de adicionar um novo botão e a função que vai actualizar os dados na base de dados:

```
Sub btActualizar_Click(sender As Object, e As EventArgs)

    Dim objCommand as OleDbCommand

    Dim IDPessoa as Long = cint(DLPessoas.SelectedValue)

    Dim nome as String = txtNome.Text
    Dim morada as String = txtMorada.Text
    Dim idade as Byte = txtIdade.Text
    Dim gs as Byte = cint(DLGS.SelectedValue)

    Dim strQuery as String = "UPDATE Pessoa SET Nome = '"& Nome &"',
Morada = '"& Morada &"', Idade = " & Idade &", GS = " & gs &" WHERE
IDPessoa = " & IDPessoa

    If Page.IsValid Then

       objCommand = New OleDbCommand(strQuery, objConnection)
       objCommand.ExecuteNonQuery()

       Response.Redirect("basedados.aspx")
    End If

End Sub
```

E assim já podemos actualizar os dados da nossa base de dados.

filmes/update.avi

Apagar dados de uma base de dados

Por fim, vamos adicionar a funcionalidade de apagar dados da base de dados, para isso, adicionamos um novo botão e utilizamos a DropDownList que contém os nomes das pessoas para identificar a pessoa a ser removida. Após esta explicação, penso que se torna simples implantar esta funcionalidade:

```
Sub btApagar_Click(sender As Object, e As EventArgs)

    Dim objCommand as OleDbCommand

    Dim IDPessoa as Long = cint(DLPessoas.SelectedValue)

    Dim strQuery as String = "DELETE * FROM Pessoa WHERE IDPessoa = "
& IDPessoa

    objCommand = New OleDbCommand(strQuery, objConnection)
    objCommand.ExecuteNonQuery()

    Response.Redirect("basedados.aspx")

End Sub
```

Assim, terminamos esta aula dedicada à manipulação de dados de uma base de dados. Poderá ver o funcionamento de todas as funcionalidades no ficheiro basedados.aspx.

Aula 4 – Usando o DataSet

Um factor importante para o sucesso da tecnologia ADO.NET é o uso de "datasets". Um dataset é uma cópia em memória dos dados da BD e contém tabelas tal como a BD real. Isto permite uma maior flexibilidade para a aplicação que utiliza esta tecnologia, pois só é efectuado o acesso ao servidor da base de dados para efeitos de leitura e escrita da BD.

Os datasets são sem dúvida poderosos, permitindo diminuir o código das aplicações ASP.NET.

A classe DataSet é membro do namespace System.Data e representa o primeiro dos dois maiores componentes da arquitectura ADO.NET (Imagem 14). O outro membro seriam os provedores Data .NET. Podemos resumir os atributos:

- É baseado em XML;
- É um conjunto de dados em cache que não está conectado ao banco de dados;
- É independente da fonte de dados;
- Pode armazenar dados em múltiplas tabelas que podem ser relacionadas;
- Armazena múltiplas versões de dados para coluna e para cada linha em cada tabela;

A ligação entre a fonte de dados e o DataSet é feita pelo objecto DataAdapter (Adaptador de dados). A imagem 19 representa uma ilustração que procura mostrar a interacção entre os objectos ADO.NET e o modelo desconectado com o DataSet.

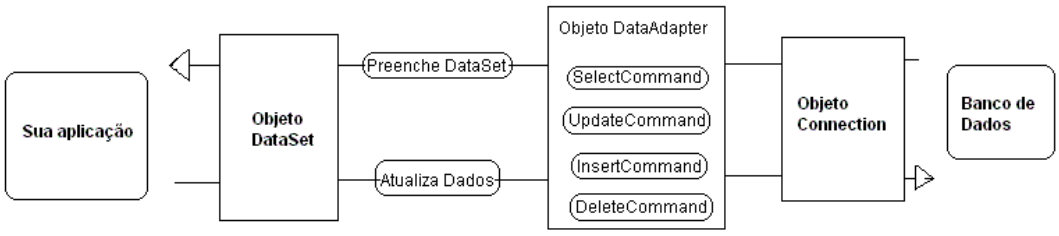


Imagem 19 – Objecto DataSet e objectos ADO.NET

Preencher um DataSet

Podemos preencher um DataSet, essencialmente, de duas formas: o modo desconectado e o modo conectado.

Vejamos o modo desconectado. Usando um DataAdapter através de seu método Fill, o método Fill invoca o objecto Command referenciado na propriedade SelectCommand do data adapter; a seguir os dados são carregados no DataSet via mapeamento através da propriedade TableMappings do data adapter. Além disso, devido à natureza desconectada do objeto DataSet, podemos carregar um DataSet via código através da sua colecção Table. Podemos também implementar a integridade entre os dados de um DataSet usando os objectos UniqueConstraint e ForeignKeyConstraint, além de criar relacionamentos entre as tabelas do DataSet.

Vamos usar a nossa base de dados pessoas.mdb e vamos preencher um DataSet com os dados da tabela Pessoa e Grupo_Sangue e criar o relacionamento entre as tabelas. Para iniciar, teremos que criar uma nova página, vamos dar o nome dataset.aspx, vamos criar a conexão à base de dados, tal como nos exemplos anteriores, vamos declarar um objecto DataSet como “variável global”:

```
Dim ds As DataSet
```

Vamos agora criar uma nova função que irá fazer a consulta à base de dados e preencher o DataSet com as duas tabelas da base de dados e respectivos relacionamentos:

```
Sub preencher()

        Dim dapessoas As OleDbDataAdapter = New OleDbDataAdapter()
        dapessoas.TableMappings.Add("Table", "Pessoa")

        Dim cmpessoas As New OleDbCommand("SELECT * FROM Pessoa",
objConnection)

        cmpessoas.CommandType = CommandType.Text

        dapessoas.SelectCommand = cmpessoas

        ds = New DataSet("Pessoas")

        dapessoas.Fill(ds)

        Dim dags As OleDbDataAdapter = New OleDbDataAdapter()

        dags.TableMappings.Add("Table", "Grupo_Sangue")

        Dim cmgs As OleDbCommand = New OleDbCommand("SELECT * FROM
Grupo_Sangue", objConnection)

        dags.SelectCommand = cmgs

        dags.Fill(ds)
```

```
        Dim dr As DataRelation

        Dim dc1 As DataColumn

        Dim dc2 As DataColumn

        dc1 = ds.Tables("Pessoa").Columns("GS")

        dc2 = ds.Tables("Grupo_Sangue").Columns("IDGS")

        dr = New System.Data.DataRelation("Grupo Sangue Pessoas", dc2,
dc1)

        ds.Relations.Add(dr)

End Sub
```

Nesta altura do curso, o aluno já deverá compreender o código sozinho, pelo que não será feita qualquer explicação, uma vez que o código é muito simples e lógico.

Agora, temos no nosso objecto DataSet uma cópia da nossa base de dados.

filmes/DataSet.avi

Inserir e actualizar dados com DataSet

Como, certamente, já reparou, o objecto DataSet tem um papel muito importante no acesso a dados com ADO.NET. Agora, vamos usar o DataSet para adicionar e alterar dados na nossa base de dados, ou seja, adicionar e alterar os dados no DataSet, pois é uma cópia da nossa base de dados, e depois reflectimos essa actualização na fonte de dados.

Para mostrar estas funcionalidades, vamos usar o exemplo anterior, que já temos um DataSet com uma cópia da nossa base de dados e vamos criar o código que permite adicionar dados e alterar registos.

```
Sub adicionar_alterar()

        Dim tblPessoas As DataTable
        tblPessoas = ds.Tables("Pessoa")

        ' Obtem um novo objeto DataRow do DataTable

        Dim drAtual As DataRow
        drAtual = tblPessoas.NewRow()
```

```
        ' Define os novos valores de DataRow

        drAtual("Nome") = "Joel Oliveira"
        drAtual("Morada") = "Amarante"
        drAtual("Idade") = "19"
        drAtual("GS") = "3"

        '  Insere  o  novo  objeto  via  método  Add  da  coleção
DataTable.Rows

        tblPessoas.Rows.Add(drAtual)

        drAtual.BeginEdit()

        drAtual("Nome") = "Joel Rocha"
        drAtual("Morada") = "Vila Real"

        drAtual.EndEdit()

        Dim comBuilder as OleDbCommandBuilder = new
OleDbCommandBuilder(dapessoas)

        dapessoas.Update(ds, "Pessoa")

End Sub
```

Definimos um objecto DataTable ao qual atribuímos a tabela Pessoa do DataSet, obtemos um objecto DataRow do objecto DataTable, ou seja, uma nova linha da tabela Pessoa, atribuímos os valores aos campos da tabela e adicionamos essa nova linha ao objecto DataTable. Nesta altura, podíamos adicionar esta nova linha à base de dados, mas para mostrar como se faz a alteração dos dados, alteramos os dados da nossa DataRow, criamos um objecto OleDbCommandBuilder, que cria dinamicamente o comando SQL para executar o pedido e por fim chamamos o método Update do DataAdapter para fazer a actualização da base de dados.

Se a operação for sobre duas ou mais tabelas, antes de chamar o método Update temos de definir as propriedades InsertCommand**,** UpdateCommand, e DeleteCommand do objecto DataAdapter, consoante as nossas necessidades. (Ver ficheiro DataSet.aspx)

Como vimos, o objecto DataSet é muito útil e permite-nos manipular os dados sem ligação à base de dados.

Para terminar esta aula, explore mais um pouco as propriedades deste objecto e as suas funcionalidades.

Aula 5 – DataGrid e DataList

Com o ASP.NET, apareceram novos controlos que facilitam a criação de aplicações, um desses controlos é o controlo DataGrid e o controlo DataList.

Nesta aula, vamos conhecer estes controlos e ver as suas potencialidades.

Controlo DataGrid

O controle DataGrid é, sem dúvida, o mais poderoso e versátil de todos os controlos do ASP.NET. Não só, tem muitas propriedades que o utilizador pode ajustar automaticamente, como também proporciona formas mais avançadas de customização. Vejamos, então, como criar e ajustar este controlo às nossas necessidades.

O controlo DataGrid é definido da seguinte forma:

```
<asp:DataGrid id="DataGrid" runat="server"></asp:DataGrid>
```

Para ver o funcionamento deste controle vamos utilizar algum código da página criada na aula anterior (dataset.aspx) e a base de dados pessoas.mdb.

Copie para um novo ficheiro chamado datagrid.aspx, as seguintes funções:

```
Sub conexao(connString as String)
Sub preencher()
Sub Page_Load(sender as Object, e as EventArgs)
```

Agora vamos adicionar um controle DataGrid à nossa página, esse controle vai ser responsável pela visualização dos dados:

```
<form runat="server">
      <asp:DataGrid id="DataGrid" runat="server"></asp:DataGrid>
</form>
```

Na função `preencher()`, vamos atribuir o DataSet à DataGrid criada:

```
DataGrid.DataSource = ds
```

```
DataGrid.DataBind()
```

Invoque a página datagrid.aspx.

IDPessoa	Nome	Morada	Idade
2	António Costo	Penafiel	36
3	Sofia Pinto	Porto	21
4	Carla Antunes	Valongo	29
6	Manuel Silva	VN Gaia	100
8	Sílvia Duarte	Bragança	20
10	Alberto Oliveira	Coimbra	45
11	Duarte Costa	Lisboa	16
12	Maria da Silva	Lisboa	46

Imagem 20 – Resultado DataGrid

Quem conhece o ASP clássico, nesta altura deve estar a pensar na quantidade de código e HTML que se gastava para ter este resultado. Agora tudo é muito mais simples.

Também deve estar a pensar, “mas o aspecto final da tabela é só este? “. A resposta é não, o controlo DataGrid pode ser configurado para exibir os dados de uma forma mais amigável, a seguir vamos reservar algum tempo para ver como isso é feito.

Vamos alterar algumas propriedades do controlo DataGrid:

```
<asp:DataGrid id="DataGrid" runat="server"
      BackColor="#FFFFC0"
      ForeColor="Navy"
      GridLines="Vertical"
      CellPadding="2">
</asp:DataGrid>
```

Para conhecer todas as propriedades do controlo DataGrid e modifica-las de uma forma simples, usamos o WebMatrix.

Se invocar a página com as alterações do objecto DataGrid, poderá ver que a apresentação dos dados é mais amigável.

Vamos tornar a nossa base de dados e consulta mais ricas, adicionando, para isso, um novo campo à tabela Pessoa da nossa base de dados. Esse campo guarda a

referência a uma imagem, essa imagem poderá ser a foto da pessoa, sendo assim a nossa tabela Pessoa fica:

Imagem 21 – Tabela Pessoa da base de dados pessoas.mdb

Agora, vamos mostrar a imagem a que o campo Foto faz referência, usando o controlo DataGrid. Para isso, temos que configurar as colunas do DataGrid, dentro das tags do objecto criamos o seguinte “código”:

```
<columns>
                <asp:BoundColumn HeaderStyle-HorizontalAlign="Center"
DataField="IDPessoa" HeaderText="ID"></asp:BoundColumn>
                <asp:BoundColumn HeaderStyle-HorizontalAlign="Center"
DataField="Nome" HeaderText="Nome"></asp:BoundColumn>
                <asp:BoundColumn HeaderStyle-HorizontalAlign="Center"
DataField="Morada" HeaderText="Morada"></asp:BoundColumn>

                <asp:BoundColumn HeaderStyle-HorizontalAlign="Center"
DataField="Idade" HeaderText="Idade"></asp:BoundColumn>
                <asp:TemplateColumn HeaderStyle
HorizontalAlign="Center">
                    <HeaderTemplate>Foto</HeaderTemplate>
                    <ItemTemplate>
                        <img src='<%# Container.DataItem("Foto") %>'
border="0" />
                    </ItemTemplate>
                </asp:TemplateColumn>
            </columns>
```

Apesar de, à primeira vista, parecer confuso, o código mostrado é muito simples e lógico. Então vamos analisar.

Entre as tags `<columns> </columns>`, é definida toda a configuração das colunas, depois temos quatro `asp:BoundColumn` que definem as colunas, neste caso uma para o Identificador, uma para o Nome, outra para a Morada e outra para a Idade, e por fim definimos um `asp:TemplateColumn`, onde personalizamos a nossa coluna, a instrução `Container.DataItem("Foto")` devolve o valor do campo Foto que o controlo DataGrid possui.

Temos de adicionar a propriedade `AutoGenerateColumns="False"` ao controlo DataGrid para que ele não gere automaticamente as colunas com os dados mas guie-se pela configuração criada. As colunas podem ter a ordem que deseja.

Vamos invocar a página datagrid.aspx:

ID	Nome	Morada	Idade	Foto
1	Paulo Alexandre Silva	Paredes	22	
2	António Costo	Penafiel	36	
3	Sofia Pinto	Porto	21	
4	Carla Antunes	Valongo	29	
8	Sílvia Liliana Duarte	Bragança	20	

Imagem 22 – Resultado da página datagrid.aspx

Se acha que este controlo é poderoso, ainda não viu o seu verdadeiro poder. Vejamos, então, como editar os dados do controlo DataGrid. Esta tarefa no ASP clássico era morosa e aborrecida, mas agora ficou tudo mais fácil.

Vamos então ver como editar os dados da tabela Pessoa da base de dados pessoas.mdb.

Os dados já foram exibidos no exemplo anterior, por isso, para os editar só temos de configurar algumas propriedades do controlo DataGrid.

Para permitir a edição no DataGrid vamos ter que incluir mais uma tag entre as tags `<columns> </columns>`, ou seja, temos de incluir uma coluna para editar, para isso, adicionamos o seguinte código:

```
<asp:EditCommandColumn ButtonType="LinkButton" CancelText="Cancelar"
EditText="Editar" UpdateText="Atualizar" />
```

Agora, temos que adicionar as seguintes propriedades ao controlo DataGrid:

```
OnEditCommand="DataGrid_Edit"
OnCancelCommand="DataGrid_Cancel"
OnUpdateCommand="DataGrid_Atualiza"
```

Estas propriedades definem as funções que respondem aos seguintes eventos, Editar, Cancelar e Actualizar. As funções são definidas da seguinte forma:

```
Sub DataGrid_Atualiza(Source As Object, E As DataGridCommandEventArgs)

End Sub

Sub DataGrid_Cancel(Source As Object, E As DataGridCommandEventArgs)

End Sub

Sub DataGrid_Edit(Source As Object, E As DataGridCommandEventArgs)

End Sub
```

Neste momento, já pode invocar a página datagrid.aspx. Aparecerá uma nova coluna com um link para editar, mas esse link não reage, pelo que teremos que inserir um código nas funções para que elas reajam aos eventos, mas primeiro, vamos acrescentar uma propriedade à coluna ID para que não possa ser editada, essa propriedade é `ReadOnly="True"`, ou seja, é uma coluna só de leitura.

```
Sub DataGrid_Atualiza(Source As Object, E As DataGridCommandEventArgs)
      Datagrid.EditItemIndex = E.Item.ItemIndex
      consulta_nome()
End Sub
```

Se invocar agora a página datagrid.aspx, o link Editar já funcionará, ele activará o link Actualizar e Cancelar e criará textboxs para editar os dados.

```
Sub DataGrid_Cancel(Source As Object, E As DataGridCommandEventArgs)
        Datagrid.EditItemIndex = -1
        consulta_nome()
End Sub
```

Se voltar a invocar a página datagrid.aspx, o link Cancelar já funcionará, ou seja, cancelará a edição dos dados.

Agora, vamos criar o código para a função que actualizará os dados na base de dados:

```
Sub DataGrid_Atualiza(Source As Object, E As DataGridCommandEventArgs)
        Dim myCommand As OleDbCommand

        Dim txtNome As TextBox = E.Item.Cells(1).Controls(0)
        Dim txtMorada As TextBox = E.Item.Cells(2).Controls(0)
        Dim txtIdade As TextBox = E.Item.Cells(3).Controls(0)

        Dim strActualiza as String = "UPDATE Pessoa SET Nome = '" &
txtNome.Text & "', Morada = '" & txtMorada.Text & "', Idade = " &
txtIdade.Text & " WHERE IDPessoa =" & E.Item.Cells(0).Text

        myCommand = New OleDbCommand(strActualiza, objConnection)

        myCommand.ExecuteNonQuery()

        Datagrid.EditItemIndex = -1
        consulta_nome()
End Sub
```

Pode reparar que o código é muito simples e curto em relação ao ASP clássico. Começamos por criar três objectos TextBox para guardar os dados editados, que são passados por argumento (E), bem como muitas outras informações, depois é o código "normal" para actualizar a base de dados, como já tivemos oportunidade de verificar. Sugiro que explore as propriedades do argumento do tipo `DataGridCommandEventArgs`.

Terminamos, assim, a actualização dos dados. Como reparou, é muito simples e rápido elaborar uma página deste género, o que não acontecia no ASP clássico. Para ver o resultado final, invoque a página datagrid.aspx.

filmes\data_grid_actualizar.avi

Para completar o nosso exemplo, vamos também fazer uma função para apagar um registo da base de dados, adicionando, para isso, a seguinte tag ao controlo DataGrid:

```
<asp:ButtonColumn Text="<img src='imagens\lixo.gif' border='0'
alt='Apagar'>" CommandName="Delete" />
```

Esta tag adiciona um novo botão a cada linha do controlo DataGrid. Este botão é representado por uma imagem definida na propriedade `Text`, mas poderia ser um simples texto. Este botão vai executar o comando Delete especificado na propriedade `CommandName`.

Agora, temos de adicionar uma propriedade ao controlo DataGrid, que vai definir a função que executará o comando deste botão, sendo que a propriedade a adicionar é `OnDeleteCommand="DataGrid_Delete"`.

Resta-nos, apenas, criar a função `DataGrid_Delete`, tal como fizemos na actualização, e adicionar o código para apagar o registo. Uma vez que, neste momento, estas matérias já serão dominadas, o código para apagar o registo não será explicado.

```
Sub DataGrid_Delete(Source As Object, E As DataGridCommandEventArgs)
        Dim myCommand As OleDbCommand

        Dim strApaga as String = "DELETE * FROM Pessoa WHERE IDPessoa
=" & E.Item.Cells(0).Text

        myCommand = New OleDbCommand(strApaga, objConnection)

        myCommand.ExecuteNonQuery()

        Datagrid.EditItemIndex = -1
        consulta_nome()

End Sub
```

Como, certamente, reparou, o controlo DataGrid é poderoso e fácil de usar e configurar. Para finalizar, invoque de novo a página datagrid.aspx e verifique todas as operações.

Existe neste controlo uma propriedade chamada paginação, cujo objectivo consiste em mostrar os registos em várias páginas, em que cada página mostra um número determinado de registos. Esta propriedade é muito útil quando o número de registos é considerável, pelo que proponho que pesquise nas propriedades do controlo DataGrid que temos vindo a elaborar, de modo a que ele efectue a paginação dos registos.

Controlo DataList

O controlo DataList é muito semelhante ao controlo DataGrid, também tem os mesmos objectivos que o DataGrid, ou seja, poupar código e elaborar aplicações rapidamente e de uma forma simples.

O controlo DataList é definido da seguinte forma:

```
<asp:DataList id="DataList" runat="server"></asp:DataList>
```

Como o funcionamento deste controlo é muito idêntico ao controlo DataGrid não será mais aprofundado. (Ver ficheiro Curso/datalist.aspx)

Aula 6 – Associar a dados Extensible Markup Language

Ao falarmos em .NET temos, obrigatoriamente, que falar em XML (Extensible Markup Language).

Ora, o ASP.NET não é excepção, e temos XML em quase tudo, já vimos que o ficheiro de configuração web.config é escrito em XML, mas não ficamos por aqui.

Nesta aula, veremos como criar um ficheiro XML com os dados de uma base de dados, para isso, temos que usar o objecto DataSet, depois disso, é tudo muito simples.

Vamos usar o exemplo anterior e no final na função `preencher()`, adicione a seguinte linha de código, conforme o caminho da sua pasta virtual:

```
ds.WriteXml("D:\\Inetpub\\wwwroot\\Curso\\pessoas.xml",XmlWriteMode.Wr
iteSchema)
```

Invoque, agora, a página datagrid.aspx. Repare que foi criado um ficheiro XML na sua pasta Curso. Se abrir esse ficheiro, poderá verificar que lá constam os dados da tabela Pessoa da base de dados pessoas.mdb, que os dados estão organizados por tags, e que essa organização também é definida nesse ficheiro XML.

Invoque o ficheiro XML no browser, aí pode verificar de uma forma mais avançada a organização dos dados:

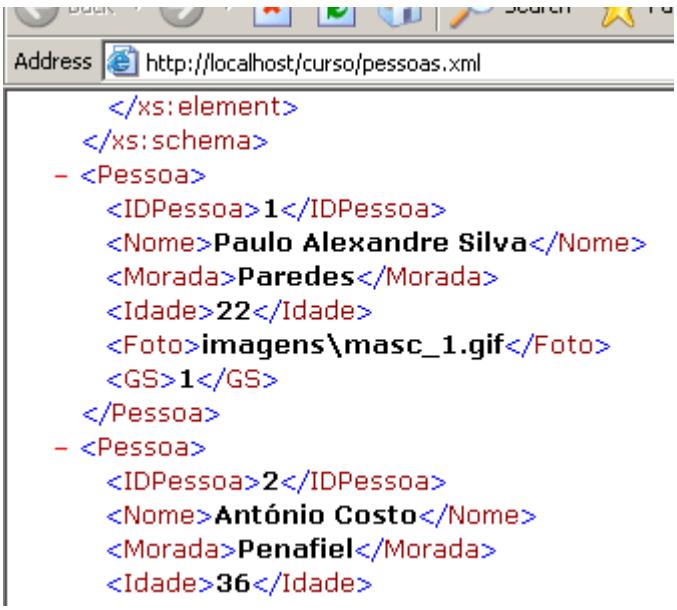


Imagem 23 – Ficheiro XML criado

Já pensou criar uma aplicação ASP.NET que guarde dados sem utilizar uma base de dados? Com o ASP.NET isso é possível, poderá utilizar um ficheiro XML como uma fonte de dados, vejamos como isso é feito.

Para mostrar como se utiliza um ficheiro XML como base de dados, vamos criar um livro de visitas.

Como vamos utilizar um ficheiro XML, temos de definir a sua estrutura, essa estrutura é definida num ficheiro XML Schema, que criaremos de seguida.

A estrutura que pretendemos para o ficheiro XML que vai guardar as visitas é a seguinte:

- `<LivrodeVisitas>` – tag principal, ou seja, é o livro de visitas em si;
- `<Mensagem>` – esta tag armazena cada mensagem incluída no Livro de visitas;
- `<id>` - identificação da mensagem ;
- `<nome>` - nome do autor da mensagem;
- `<data>` - data da mensagem;
- `<mensagem>` - a mensagem em si.

Com esta configuração, o ficheiro XML que guardará as mensagens ficará:

```
<?xml version="1.0" encoding="utf-8" ?>
<LivrodeVisitas>
    <Mensagem>
        <id>0</id>
        <nome>Roberto Rocha</nome>
        <data>2004-03-22 13:22</data>
        <mensagem>Esta é a primeira mensagem do GuestBook.</mensagem>
    </Mensagem>
</LivrodeVisitas>
```

Então, vamos definir o ficheiro XML Schema (guest.xsd):

```
<?xml version="1.0" standalone="yes" ?>
<xs:schema id="LivrodeVisitas" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
    <xs:element name="LivrodeVisitas" msdata:IsDataSet="true">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="Mensagem">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="ID"
msdata:AutoIncrement="true" type="xs:int" />
                            <xs:element name="Nome" type="xs:string"
minOccurs="0" />
                            <xs:element name="Data" type="xs:dateTime"
minOccurs="0" />
                            <xs:element name="Mensagem"
```

```
type="xs:string" minOccurs="0" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
        <xs:unique name="Constraint1" msdata:PrimaryKey="true">
            <xs:selector xpath=".//guest" />
            <xs:field xpath="ID" />
        </xs:unique>
    </xs:element>
</xs:schema>
```

Já temos definida a estrutura dos dados, agora vamos criar uma página chamada livrovisitas.aspx, para inserir e visualizar mensagens. Necessitamos de dois campos, Nome e Mensagem, de um botão para inserir as mensagens e de um controlo para visualizar as mensagens existentes, esse controlo poderá ser o Repeater que não falamos dele, mas que é muito semelhante ao controle DataGrid e DataList. Se tiver dificuldades, consulte o ficheiro LivroVisitas\livrovisitas.aspx.

Após a criação do formulário, vamos adicionar código para visualizar as mensagens e o código para adicionar novas mensagens. Vamos usar DataSet e DataTable para gerar o XML. O código é muito simples.

Vamos criar duas variáveis “globais” do tipo String que vão armazenar o caminho para o ficheiro guest.xml e guest.xsd já criados:

```
Dim strXML As String = Server.mappath("guest.xml")
Dim strXSD As String = Server.mappath("guest.xsd")
```

Agora, vamos criar o código para visualizar as mensagens existentes, para trabalhar com ficheiros XML temos de importar um Namespace chamado System.XML.

Agora já podemos trabalhar com ficheiros XML, a leitura das mensagens já existentes será feita quando a página for carregada:

```
Sub Page_Load()

    'Criar uma variável do tipo XmlDataDocument
    Dim xddGuest As New XmlDataDocument

    'Apontar o DataSetName definido no guest.xsd
    xddGuest.DataSet.DataSetName = "LivrodeVisitas"
```

```
    'Ler o ficheiro guest.xsd
    xddGuest.DataSet.ReadXmlSchema(strXSD)

    'Carregar o ficheiro guest.xml
    xddGuest.Load(strXML)

    'Aqui vamos usar o nosso DataSet criado e vinculá-lo ao Repeater.
    rptGuest.DataSource = xddGuest.DataSet.Tables("Mensagem")
    rptGuest.DataBind()
End Sub
```

Com esse código, carregamos o DataSet e associamos ao Repeater. Agora, temos que fazer com que o Repeater leia os campos do DataSet e apresente os resultados. Para isso, configure o Repeater da seguinte forma:

```
<asp:repeater id="rptGuest" runat="server">
    <HeaderTemplate>
        <div align="center">
            <font face="verdana" size="4"><b>Mensagens</b></font>
            <br />
            <br />
        </div>
    </HeaderTemplate>
    <ItemTemplate>
        <font face="verdana" size="2" color="#0053b9">
            <%# DataBinder.Eval(Container.DataItem, "Nome") %>
        </font>- <font face="verdana" size="2">
            <%# DataBinder.Eval(Container.DataItem, "Mensagem") %>
        </font>
        <br />
        <font face="verdana" size="2"><i>
                <%# DataBinder.Eval(Container.DataItem, "Data") %>
            </i></font>
        <br />
        <br />
    </ItemTemplate>
</asp:repeater>
```

Agora, já podemos visualizar as mensagens no nosso livro de visitas, crie algumas mensagens no ficheiro guest.xml e invoque a página livrovisitas.aspx:

Imagem 24 – Mensagens do Livro de visitas

Vamos criar o código para inserir novas mensagens no ficheiro guest.xml, esse código vai ser criado na função que reage ao evento Click do botão Enviar. Essa função cria uma linha, insere na tabela, grava no DataSet e mostra de novo as mensagens.

```
Sub cmdEnviar_Click(sender As Object, e As EventArgs)

    'Criar uma variável do tipo XmlDataDocument
    Dim xddGuest As New XmlDataDocument

    'Apontar o DataSetName definido no guest.xsd
    xddGuest.DataSet.DataSetName = "LivrodeVisitas"

    'Le o ficheiro guest.xsd
    xddGuest.DataSet.ReadXmlSchema(strXSD)

    'Carrega o ficheiro guest.xml
    xddGuest.Load(strXML)

    'Criar uma nova tabela
    Dim GuestTable As DataTable
    GuestTable = xddGuest.DataSet.Tables("Mensagem")

    'Criar uma nova linha
    Dim GuestRow As DataRow
    GuestRow = GuestTable.NewRow

    'Definer os valores da linha
    GuestRow.Item("Nome") = txtNome.Text

    Dim data As DateTime
    data = Now()
    GuestRow.Item("Data") = data
    GuestRow.Item("Mensagem") = txtMensagem.Text

    'Incluir a linha na tabela
    GuestTable.Rows.Add(GuestRow)

    'Escrever os dados na tabela
    GuestRow.AcceptChanges()

    'Escrever os dados no DataSet
    xddGuest.DataSet.AcceptChanges()

    'Escrever um novo ficheiro guest.xml
    xddGuest.DataSet.WriteXml(strXML, XmlWriteMode.IgnoreSchema)

    'Associa a fonte de dados ao Repeater novamente
    rptGuest.DataSource = GuestTable
    rptGuest.DataBind()
End Sub
```

E assim, terminamos o nosso livro de visitas. Ao fazer clique sobre o botão enviar, a mensagem será gravada do ficheiro guest.xml e mostrada na página. Como foi simples construir um livro de visitas!

Invoque a página livrovsitas.aspx e crie uma nova mensagem.

filmes\livrovisitas.avi

Terminamos, assim, esta aula dedicada aos dados via XML, e terminamos este módulo. Pode, agora, considerar-se preparado para desenvolver aplicações web mais robustas e complexas.

Auto-avaliação do módulo 3

Continue o exercício de auto-avaliação do módulo 2, crie uma base de dados que contém dados dos utilizadores, crie um formulário para registar novos utilizadores e altere o formulário de acesso, de modo a pesquisar se o utilizador existe ou não na base de dados. Cada acesso deverá ficar registado num ficheiro XML, com a data, hora e nome do utilizador.

Curso\Restrita\acessos.mdb
Curso\Restrita\login.aspx
Curso\registo.aspx
Curso\Restrita\degault.aspx
Curso\Restrita\acessos.xml
Cursos\Restrita\acessos.xsd

Módulo 4 – Separar código e conteúdo

Objectivos do módulo

O objectivo deste módulo é ensinar a criar aplicações web modulares, separando o código do conteúdo para reaproveitamento de código.

Sumário

Aula 1 – Separar código e conteúdo

Os programadores que conhecem o paradigma da programação orientada a objectos sabem e bem que deve-se sempre separar a parte de processamento da aplicação, com a parte de interface com o utilizador. São coisas distintas, por isso, faz todo o sentido estar separado. Com o *code behind,* as ASP.NET proporcionam-nos uma forma cómoda e elegante de o fazer.

Esta ideia consiste em ter um ficheiro template, em que temos a estrutura do site em HTML, e dizemos que esta página deriva de uma outra, que, por sua vez, num ficheiro à parte, realiza o processamento.

Como já reparou, o ASP.NET é muito organizado, separando o código do conteúdo, mas até agora fizemos tudo no mesmo ficheiro. A partir deste momento, vamos entrar na programação ASP.NET mais estruturada. Ao criarmos uma aplicação desta forma, teremos, essencialmente, dois ficheiros, como já vimos, o ficheiro com o HTML e outro com o processamento:

- Ficheiro com HTML

  Este ficheiro tem como extensão .aspx e na primeira linha deve identificar-se qual o ficheiro code behind que processa a página e qual a classe de que ela deriva:

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="pagina.aspx.vb" Inherits="WebApplication1.WebForm1"%>
```

  Neste exemplo, o código de processamento está no ficheiro pagina.aspx.vb e o ficheiro .aspx deriva da classe WebForm1 da aplicação WebApplication1.

- Ficheiro de processamento

  Este ficheiro tem como estensão .aspx.vb, no caso da linguagem usada ser o VB, neste ficheiro é definida a classe que vai processar o ficheiro com extensão .aspx.

Como já reparou, neste modelo de programação usam-se classes, pelo que uma classe define um conjunto de características que são comuns a uma série de objectos.

Desde o início do curso, temos vindo a usar classes para aceder às bases de dados, aos ficheiros XML, entre outros, mas essas classes já estão definidas no .NET, embora também possamos criar as nossas próprias classes. O corpo de uma classe é definido da seguinte forma:

```
Public Class XXXXX
      ‘Atributos
      ‘Métodos
End Class
```

Os atributos distinguem o objecto de um outro do mesmo tipo, guardam as variáveis internas do objecto.

Os comportamentos são normalmente implementados por intermédio dos métodos, podemos chamar a um método uma função, ou quase, ou seja, é uma entidade lógica que aceita determinados parâmetros de entrada e que realiza uma determinada acção, podendo ou não devolver uma resposta. A diferença é que os métodos, ao contrário das funções, são definidos no interior de uma classe e destinam-se a operar entre objectos dessa classe.

Nesta aula, já ficou com uma ideia de como se separa código de conteúdo e as suas vantagens. Para terminar a aula, sugiro que faça uma pesquisa na Internet para aprofundar mais este assunto.

Aula 2 – Criar, através da utilização de páginas code behind com o VS .NET

Após uma primeira aula mais teórica, passemos à prática. Vamos criar uma aplicação ASP.NET com o modelo de programação code behind, usando, para tal, um ambiente de desenvolvimento .NET bastante potente, e que nos facilita muito a criação de aplicações. Esse ambiente de desenvolvimento é o Visual Studio 2003 .NET da Microsoft®.

filmes\vs.avi

Como reparou, ao criarmos uma aplicação para web em VB, o Visual Studio .NET, criou, automaticamente, cinco ficheiros, nomeadamente:

- AssemblyInfo.vb – neste ficheiro encontram-se informações de assembly, como nome da aplicação, versão, companhia, etc;
- Globalasax – este ficheiro tem a mesma função do ficheiro Global.asa do ASP clássico, nele são definido métodos para quando a aplicação é iniciada, terminada, quando ocorre um erro, etc;
- Styles.css – ficheiro de estilos;
- Web.config – como já vimos, este ficheiro é utilizado para configurar a aplicação;
- WebForm1.aspx – o ficheiro ASP.NET propriamente dito, sendo neste ficheiro que é criado todo o código HTML.

Se fizer duplo clique sobre a área de desenho do ficheiro WebForm1.aspx abre o ficheiro WebForm1.aspx.vb, que, como já vimos, é o ficheiro onde é definida a classe que processa o ficheiro WebForm1.aspx:

```
Public Class WebForm1
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

End Class
```

É criado o método Page_Load, visto que fez duplo clique sobre a área de desenho, logo, se adicionar novos objectos à página e fizer duplo clique sobre ela, é criado um método desse objecto.

Agora, vamos criar um exemplo de uma aplicação ASP.NET neste modelo de programação e usando o Visual Studio .NET.

Abra o Visual Studio .NET e crie uma nova aplicação web, utilizando como linguagem o VB, tal como viu no filme.

Adicione uma etiqueta (label), altere as propriedades na janela de propriedades e adicione um texto de boas vindas à etiqueta quando a página é aberta.

Repare que quando chama uma propriedade ou um método de um objecto após fazer . aparece a lista de propriedades e métodos desse objecto, o que facilita, e muito, a criação de aplicações.

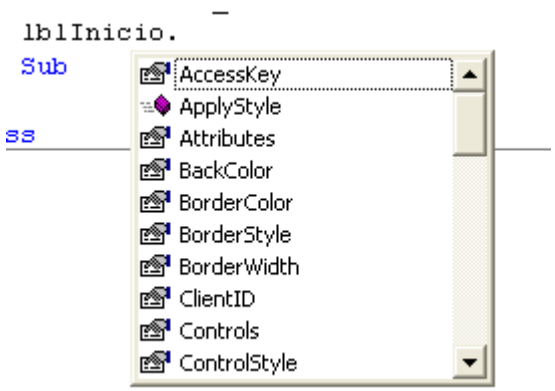


Imagem 25 – Propriedades e métodos de um objecto label

Agora, temos duas formas de correr a nossa aplicação, uma delas é como temos feito até aqui, invocamos num browser a página desejada e ela será compilada e mostrada, ou vamos ao menu Build e seleccionamos a opção Build Solution. Se a compilação correr bem, será mostrada uma mensagem de sucesso, senão, são mostrados os erros que a aplicação contém, depois da compilação ser executada com sucesso só temos que invocar a página num browser para a visualizar, mas agora já não será compilada, pois já o foi. Também podemos seleccionar a opção Start do menu Debug.

Vamos agora criar uma nova classe na nossa aplicação. Será uma classe muito simples, uma classe que contém um atributo do tipo String e quando é criado um objecto desse atributo é passado como argumento, uma frase que instancia esse atributo, depois criamos um método que retorna o valor do atributo.

Para criar uma nova classe no VS. NET, temos várias maneiras, uma delas é seleccionar a opção Add New Item do menu Project. Em templates seleccione Class e atribua o nome Texto.vb. Agora, já temos mais um ficheiro na nossa aplicação, o ficheiro que contém a classe Texto, que teremos que implementar:

```
Public Class Texto

    Private strTexto As String

    Public Sub New(ByVal str As String)
        strTexto = str
    End Sub

    Public Function getTexto() as String
        Return strTexto
    End Function
End Class
```

De notar que o método que retorna o atributo é uma `Function`, pois tem de retornar um valor com a instrução `Return`, os métodos são definidos como `Public`, pois têm de ser acedidos por outras classes, o atributo é definido como `Private`, visto que só pode ser acedido pela própria classe.

Após a implementação, temos de compilar a classe, para isso, seleccione a opção Build Solution do menu Build, tal como fizemos para compilar a aplicação anteriormente. Se a compilação decorreu com sucesso já temos a classe à nossa disposição. Vejamos, agora, como usa-la.

Voltamos ao nosso ficheiro WebForm1.aspx e dentro do método Page_Load, vamos criar um novo objecto da classe Texto com a frase de boas vindas, depois só temos de chamar o método `getTexto()` do objecto criado para mostrar a frase de boas vindas.

```
Dim objTexto As Texto
objTexto = New Texto("Ola esta página foi construida no VS .NET")

lblInicio.Text = objTexto.getTexto()
```

Como pode ver, a separação de código é vantajosa, pois podemos reaproveitar muito código já existente, podemos definir classes para fazer determinadas tarefas repetitivas. O uso do VS .NET como ambiente de desenvolvimento de aplicação ASP.NET facilita, e muito, a criação deste tipo de aplicações, ente outras, por isso, para terminar esta aula, sugiro que explore um pouco mais o VS. NET e que resolva o exercício proposto.

Exercício proposto

Crie uma nova classe na aplicação anterior, que faça as operações matemáticas, somar, subtrair, multiplicar, dividir. Crie um formulário onde será mostrado o funcionamento da classe.

Curso\VisualStudio\Operacoes.vb
Curso\VisualStudio\WebForm1.aspx
Curso\VisualStudio\WebForm1.aspx.vb

Aula 3 – Criar e usar User Controls

Os User Controls são uma nova característica do ASP.NET que proporciona uma forma rápida e simples de reutilizar código. Os User Controls são constituídos por código HTML e/ou código ASP.NET que é armazenado em ficheiros com extensão ascx.

Imagine que tem que construir uma aplicação para a web que, em todas as páginas, possua, por exemplo, uma caixa que contém o nome e a descrição de um utilizador autenticado. Construía essa caixa em todas as páginas da aplicação copiando o código de página para página. Imagine, ainda, que não gostou da cor de fundo da caixa, do tipo de letra, acha que a caixa está muito larga, e quer mudar estes aspectos e que, por isso, tinha que ir ao "código" de cada uma das páginas e alterar. Imagine, agora, se a aplicação contém dezenas de páginas …

Por sua vez, no ASP.NET, se usar User Controls, todo esse trabalho não existe, tudo é muito mais simples de modificar! Um User Control pode ser descrito como um pedaço de código que possui "vida própria". Cria o seu próprio controle, adiciona-o na(s) página(s) da aplicação que deseja, tal como se adiciona uma caixa de texto ou um botão, e todas as alterações que se fizer no controle reflectir-se-ão em todas as páginas que possuem o controle.

User Controls no ASP.NET trazem os benefícios do encapsulamento de código, uso optimizado de memória, código compilado e a capacidade de fazer mudanças rapidamente.

Então, agora, vamos criar um User Control, seguindo o exercício de login do nosso curso, o User Control mostrará a data actual e o nome do utilizador autenticado, e terá um botão que fará o logout do utilizador. Por fim, será adicionado à página default.aspx da nossa área restrita.

Poderá criar este controle com a ajuda do Visual Studio .NET ou com o Web Matrix, qualquer destes ambientes permite criar ficheiro ascx.

Criamos o ficheiro utilizador.ascx dentro da nossa área restrita. Vamos adicionar as duas etiquetas e o botão, não são necessárias as tags <html> e <body>, pois a(s) página(s) a onde o controle vai ser adicionado já as contém.

Após o desenho do controle, vamos dar-lhe inteligência, vamos criar o código que permite fazer o que pretendemos, como vamos ter a data e o nome do utilizador vamos criar duas variáveis “globais”, para guardar estes dados:

```
Private data as DateTime
Private nome as String
```

No evento Page_Load do controle, adicionamos o código para preencher o User Control:

```
Private Sub Page_Load()
    lblData.Text = data.Now.Date()
    lblNome.Text = HttpContext.Current.User.Identity.Name
End Sub
```

No método que responde ao evento click do botão de logout adicionamos o seguinte código:

```
Private Sub btLogout_Click(sender As Object, e As EventArgs)
     FormsAuthentication.SignOut()
End Sub
```

Neste momento, já temos o User Control pronto a ser adicionado a uma ou várias páginas, por isso, vamos adiciona-lo à página default.aspx da nossa área restrita.

Para usar um controle numa página, é preciso torná-lo disponível para a página e implementá-lo nesta. Se o controle for adicionado, declarativamente, é necessário usar a directiva @Register para configurar a sintaxe da sua tag.

Quando se usa a directiva @Register, pode declarar-se uma instância do User Control, usando uma sintaxe baseada em tags, similar ao HTML. Os atributos TagPrefix e TagName na directiva @Register determinam a sintaxe para a tag.

Na primeira linha da página default.aspx adicione a seguinte tag:

```
<%@ Register TagPrefix="UtlControl" TagName="UtilizadorControl"
Src="utilizador.ascx" %>
```

Repare que ela informa o prefixo da tag (TagPrefix) a ser utilizado, o nome do controle (TagName) e a sua localização (Src, no caso, o ficheiro do controle está no mesmo directório que a página aspx). Estas são informações básicas sem as quais o

controle não funcionará. Tendo o controle registado, vamos inserir a tag do nosso User Control na página:

```
<UtlControl:UtilizadorControl id="MeuControle1" runat="server" />
```

Como pode verificar, ela segue as informações que usamos ao registar o controle para a página (`UtlControl`:`UtilizadorControl`). Veja que também temos o id do controle, que usaremos para identificá-lo dentro do código, também podemos ter propriedades.

Agora, vamos aprender como colocar um controle na cache do ASP.NET, que é mais um novo recurso oferecido no ASP.NET, que permite fragmentar a área da página que vai ser armazenada dentro do servidor e recuperada de forma rápida sem necessidade de reprocessamento. Um exemplo simples poderia ser um DropDownlist que carrega dados de uma base de dados e mostra os seus itens, digamos que essa base de dados é actualizada uma vez por mês. Então, esse controle pode ficar na cache e definir um prazo de duração de 30 dias, por exemplo. Quando a página carregar a primeira vez, o controle vai consultar a base de dados, carregar os dados e armazenar na cache. Os próximos utilizadores a visitar a página não precisarão de perder tempo, pois o ASP.NET vai recuperar o controle da cache. Ao final do prazo o controle expira na cache e volta a ter necessidade de consultar a base de dados.

Para mostrar o funcionamento da cache adicione ao User Control desta aula a seguinte tag:

```
<%@OutputCache Duration="30" VaryByParam="none"%>
```

Esta tag determina que o controle vai ser armazenado na cache com uma duração de 30 segundos.

Invoque a página default.aspx, autentique-se e fique a actualizar a página (F5), frequentemente. Irá observar que a hora do controle só actualiza a cada 30 segundos.

Assim, terminamos esta aula dedicada aos User Controles, que, como teve oportunidade de constatar, é mais uma boa forma de criar aplicações web organizadas.

Aula 4 – Criar e usar componentes

A palavra-chave dos componentes é “Reaproveitamento de código”, como já vimos nos User Controls. Desenvolver uma aplicação bem estruturada, proporcionando o uso de componentes, é uma excelente maneira de criar aplicações de forma organizada, modular, simples e objectiva. Um bom planeamento de toda a aplicação antes de desenvolvê-la, facilita a sua criação. Toda e qualquer manutenção a ser feita sempre será mais rápida com o uso de componentes.

Se estivesse a desenvolver várias aplicações que, por exemplo, necessitavam de dados de uma mesma base de dados, copiava o código de aplicação para aplicação, e se as aplicações fossem implementadas em linguagens diferentes, a sintaxe do código já não era igual. Para um bom programador, o uso de componentes é fundamental, poderia desenvolver um componente para aceder à base de dados e retornar os dados que as aplicações necessitassem, depois só teria que referenciar o componente na aplicação e usa-lo, sendo o mesmo componente usado em todas as aplicações que necessitassem de aceder àqueles dados da base de dados.

Componentes são ficheiros com extensão dll, o componente poderá ser desenvolvido, usado e herdado por qualquer linguagem .NET. Poderá ser desenvolvido numa linguagem e ser usado em outra linguagem, o seu uso é transparente.

Para demonstrar o uso de componentes, vamos usar a nossa base de dados pessoas.mdb e criar um componente que retorne um DataSet da tabela Grupo_Sangue, e retorne também uma DataSet da tabela Pessoa de um determinado grupo de sangue.

Para criar o componente, vamos usar o Visual Studio .NET. Crie um novo projecto do tipo “Class Library” e atribua o nome de “Componente”.

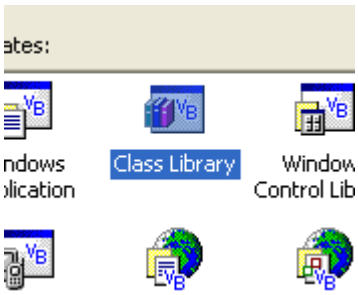


Imagem 26 – Criação de componente no VS .NET

Após criar o projecto “Componente”, altere o nome da classe criada automaticamente “Class1.vb” para “AcessoDados.vb”:

```
Public Class AcessoDados


End Class
```

Como vamos usar o acesso a uma base de dados MS Access, temos que importar o seguinte namespace:

```
Imports System.Data.OleDb
```

Vamos definir um atributo da classe, esse atributo será do tipo String e guardará a string de conexão à base de dados:

```
Private connString As String = "PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA
SOURCE=D:\Inetpub\wwwroot\Curso\pessoas.mdb;"
```

O caminho descrito em DATA SOURCE, terá de ser o caminho para a base de dados pessoas.mdb, que, neste caso, é o descrito, mas no seu caso poderá não o ser.

Vamos criar o método (função), que após uma consulta à base de dados retorna um DataSet com os vários grupos de sangue:

```
Public Function GrupoSangue() As DataSet
        Dim objConnection As OleDbConnection
        Dim da As OleDbDataAdapter
        Dim ds As DataSet

        Dim sql As String = "Select * FROM Grupo_Sangue ORDER BY GS"

        objConnection = New OleDbConnection(connString)
        objConnection.Open()

        da = New OleDbDataAdapter(sql, objConnection)

        ds = New DataSet
        da.Fill(ds, "Grupo_Sangue")

        objConnection.Close()

        Return ds
End Function
```

Como pode reparar, não existe nada de novo, faz-se a conexão à base de dados, a consulta e associa-se a consulta a um DataSet que depois é retornado.

O método (função) que retorna uma DataSet com pessoas de um determinado grupo de sangue é muito semelhante e também não acarreta nenhuma novidade.

```
Public Function Pessoas(ByVal idGrupoSangue As Byte) As DataSet
        Dim objConnection As OleDbConnection
        Dim da As OleDbDataAdapter
        Dim ds As DataSet

        Dim sql As String = "SELECT * FROM Pessoa WHERE GS = " &
idGrupoSangue

        objConnection = New OleDbConnection(connString)
        objConnection.Open()

        da = New OleDbDataAdapter(sql, objConnection)

        ds = New DataSet
        da.Fill(ds, "Pessoa")

        objConnection.Close()

        Return ds
End Function
```

Após digitar o código, o último passo para criar um componente é fazer a sua compilação para criar o ficheiro dll. Para isso, seleccione a opção “Build Solution” do menu “Build”, se a compilação tiver sucesso é criado na pasta “Bin” do projecto o ficheiro Componente.dll, este ficheiro é o nosso componente pronto a ser usado.

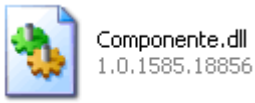


Imagem 27 – Ficheiro dll, componente

Agora, todas as aplicações que necessitam das funcionalidades do componente terão de referenciar este componente para o poderem usar.

Então, vamos ver como se usa um componente, para facilitar copie o ficheiro dll para a raiz do directório “Curso”.

Crie no VS .NET um novo projecto do tipo “ASP.NET Web Application” e dê o nome de “UsoComponente”, altere o nome do ficheiro WebForm1.aspx para UsoComponente.aspx, e adicione ao formulário uma ListBox para listar os vários grupos de sangue e uma DataGrid para listar as pessoas com o grupo de sangue seleccionado na ListBox. Para a ListBox, altere a propriedade “AutoPostBack” para

"True", para que quando seja seleccionado um grupo de sangue da ListBox, o código seja disparado automaticamente.

Para usar o componente criado, temos de o referenciar na nossa aplicação, para isso, seleccione a opção "Add Reference" do menu "Project", e localize e seleccione o componente que copiamos para a pasta "Curso":

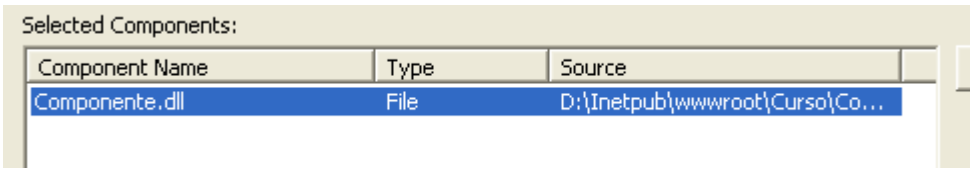


Imagem 28 – Selecção do componente

No evento Page_Load, digite o código para preencher a ListBox com os vários grupos de sangue:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        If Not IsPostBack Then
            'referencia o componente
            Dim comp As New Componente.AcessoDados
            'atribui as propriedades para a ListBox
            With Me.lbGS
                .DataTextField = "GS"
                .DataValueField = "IDGS"
                'preenche a ListBox
                .DataSource =
comp.GrupoSangue.Tables("Grupo_Sangue").DefaultView
                .DataBind()
            End With
            Me.dgPessoas.Visible = False
        End If
End Sub
```

O comando IsPostBack verifica se foi dado um Post na página, ou seja, esse código será carregado apenas na primeira vez que a página for carregada, mesmo estando no evento Page_Load.

Para referenciar um componente, atribui-se o mesmo a uma variável, utilizando o New NomeComponente.NomeClasse().

Para utilizar a função existente na classe do componente, use a variável, mais ponto, mais nome da função. Como esta classe retorna um Dataset que contém um DataAdapter chamado "Grupo_Sangue", então, a origem (DataSource) da ListBox é exactamente a tabela designada "Grupo_Sangue" gerada pelo DataAdapter.

Agora, vamos criar o código para o evento SelectedIndexChanged da ListBox para preencher o DataGrid com as pessoas do respectivo grupo de sangue. Para facilitar, dê um duplo clique sobre o objecto ListBox, que o evento é criado automaticamente.

```
Private Sub lbGS_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles lbGS.SelectedIndexChanged
        'referencia o componente
        Dim comp As New Componente.AcessoDados
        Me.dgPessoas.Visible = True
        'monta e popula o datagrid
        With Me.dgPessoas
            .DataSource =
comp.Pessoas(Me.lbGS.SelectedItem.Value).Tables("Pessoa")
            .DataBind()
        End With
End Sub
```

Aqui, também criamos uma variável para referenciar o componente, e depois é atribuído à origem do DataGrid o resultado do método Pessoas do componente que tem como parâmetro o identificador do grupo de sangue.

Por fim, só temos que compilar o projecto e invocar a página UsoComponete.aspx.

Se tudo correr da melhor forma, o DataGrid é preenchido com as pessoas que têm o grupo de sangue igual ao seleccionado na ListBox.

Terminamos, assim, esta aula dedicada aos componentes, e terminamos este módulo em que aprendemos a criar aplicações mais estruturadas e modulares, que permitem o reaproveitamento de código e uma fácil e rápida manutenção das nossas aplicações.

Auto-avaliação do módulo 4

Crie um componente em VS .NET que envie correio electrónico (E-mail) e crie também um User Control que contém um formulário que usará o componente criado.

Para enviar E-mail, tem de criar um objecto do tipo “MailMessage”. Para reconhecer este objecto, tem que importar o namespace “System.Web.Mail” e depois chamar as suas propriedades. As propriedades a usar estão descritas na tabela seguinte:

Propriedade	Descrição
To	Destino do E-mail
From	Origem do E-mail
Subject	Assunto do E-mail
Body	Corpo do E-mail
BodyFormat	Formato do E-mail (use MailFormat.Text)

Para enviar, tem de seleccionar um servidor de smtp válido, usando a instrução SmtpMail.SmtpServer = "smtp.servidor.pt", e, por fim, usar a instrução SmtpMail.Send(objEmail).

Projecto Curso\ComponenteEMail
Curso\Enviarmailaspx

Módulo 5 – Web Services

Objectivos do módulo

Este módulo tem como objectivo ensinar a criar e a usar Web Services, bem como entender o seu funcionamento e a sua utilidade.

Sumário

Aula 1 – O que é um serviço Web?

Services são "programas" que são executados em "background" no sistema operacional.

A grande vantagem desses "programas" é que não precisam de ser executados por nenhum utilizador, ou seja, quando se liga o computador, o serviço começa a ser executado e pode fazer qualquer tarefa como qualquer outra aplicação do sistema. O uso de XML também é uma grande vantagem.

Os Web Services são uma nova e inovadora técnica para a troca de dados via Web e muitas empresas estão a aderir ao fornecimento e consumo de serviços na web.

Anteriormente aos Web Services, a troca de dados entre entidades era normalmente realizada através de componentes COM.

Porém, a comunicação de componentes COM uns com os outros em locais distantes de uma rede ou até mesmo da internet sempre foi algo complexo. Os componentes COM utilizam o protocolo de rede DCOM, que, por sua vez, é baseado no RPC. Consequentemente, torna-se necessário ter a porta 135 e mais 20 portas aleatórias abertas para a comunicação entre os componentes. Isso sempre foi uma catástrofe para a configuração de Firewalls. Deixar muitas portas abertas é sempre um convite a invasões.

Claro que sempre foi possível determinar quais as 20 portas que seriam usadas, mas sempre foi uma configuração demasiadamente complexa, visto que a grande maioria dos administradores de rede não sabia realiza-la.

Devido a estes problemas surgiu a ideia de fazer a troca de dados entre componentes COM via XML, apenas pela porta 80.

Porém, como um componente COM pode chamar outro, via porta 80, como toda a comunicação dos componentes pode passar por HTTP? Às vezes, a resposta correcta acaba por ser a mais simples e óbvia: basta ter um servidor web com "algo" instalado que sirva como intermediário. Esse servidor Web recebe via POST um documento XML contendo a descrição do que tem que ser chamado (componente/método), executa o componente e devolve a resposta em XML. Durante muito tempo, foram utilizadas simples páginas ASP para realizar esse papel de intermediárias.

Foi neste ponto que começaram a surgir as necessidades de padronização desta tecnologia. Surgiu então o Soap (Simple Object Access Protocol), um padrão de documento XML para fazer o disparo de um método de um determinado componente existente remotamente e devolver o resultado da execução deste método.

Junto com o SOAP surgiu, praticamente, mais uma necessidade de padronização, ou seja, tornou-se necessário que a aplicação fosse capaz de identificar automaticamente os métodos existentes num serviço remoto e a forma de chama-los, para que assim pudesse, não só validar como até mesmo gerar automaticamente os pacotes SOAP para fazer a comunicação.

Surgiu então o padrão WSDL (WebServices Description Language), que fornece a descrição de tudo o que um WebService possui. Surgiram, então, Wizards e componentes capazes de utilizar o WSDL para gerar automaticamente as chamadas SOAP. Realizar a comunicação entre objectos remotos passou a ser semelhante a chamar um componente qualquer, permitindo até mesmo que o programador se esquecesse de que a comunicação ocorre via XML na porta 80.

Os web services são mais comparáveis a um protocolo de comunicação do que a um componente. Web Service é uma forma de comunicação padronizada entre dois componentes, mas o tipo de componentes não é importante. Quem vai executar o serviço em si, pode ser qualquer um: uma página ASP, um componente COM, enfim, qualquer coisa.

Mas o WSDL ainda não é suficiente para um bom funcionamento dos Web Services na Web. Isto, porque ainda é necessário que o utilizador do serviço saiba a localização exacta do serviço para poder utiliza-lo.

Para facilitar a descoberta da localização de serviços, foi criado o protocolo de Discovery. Através deste protocolo, podemos interrogar um servidor web para descobrirmos os serviços que o servidor possui.

Porém, com o protocolo de Discovery, ainda teríamos que interrogar servidor por servidor para localizarmos um serviço. Para evitar isso, foi criado o padrão UDDI, um padrão para catálogos públicos de Web Services. A própria Microsoft mantém um catálogo UDDI, que pode ser pesquisado através do próprio Visual Studio.

Aula 2 – Chamar um serviço Web existente na web

Existem neste momento vários Web Services disponíveis na Web e que poderão ser usados nas nossas aplicações, sendo um deles, o serviço de pesquisa disponibilizado pelo Google. Vamos ver um exemplo de utilização de Web Services, utilizando o Web Service do Google para realizar uma pesquisa.

Para usar o Web Service do Google é necessário fazer um registo gratuitamente e fazer o donwload de uma ferramenta de desenvolvimento no endereço http://www.google.com/apis/. Vamos, porém, fazer a demonstração, directamente, utilizando uma chave já gerada pelo google.

Crie um novo projecto ASP.NET Web Application no Visual Studio .NET e dê o nome de "WebServiceGoogle". Após a criação do projecto, vamos referenciar o Web Service do Google, para isso, seleccione a opção "Add Web Reference…" do menu Project. No campo URL insira o endereço do ficheiro WSDL que contém a descrição do Serviço, esse URL é http://api.google.com/GoogleSearch.wsdl, tendo em atenção que o servidor é case sensitive. Clique em GO e, se correr bem, a referência ao serviço é adicionada ao projecto e está pronto a ser usado.

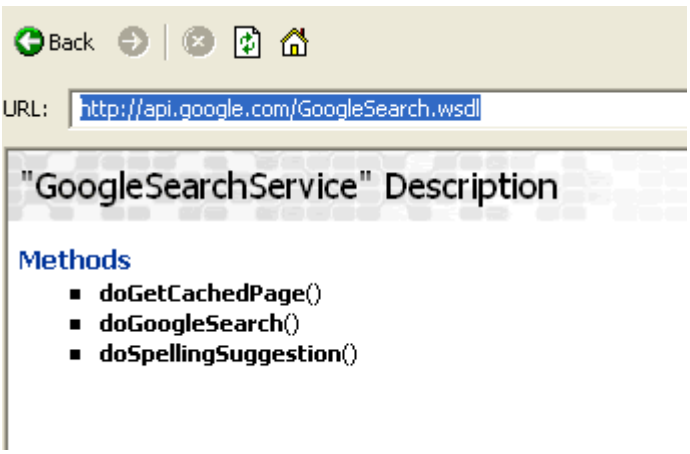


Imagem 29 – Web service Google

De notar que o nome do serviço é "GoogleSearchService" e contém três métodos que podemos usar.

Foi adicionado um ícon na pasta "Web References", um globo que representa o Web Service, para facilitar, no futuro, a alteração do nome para "Google".

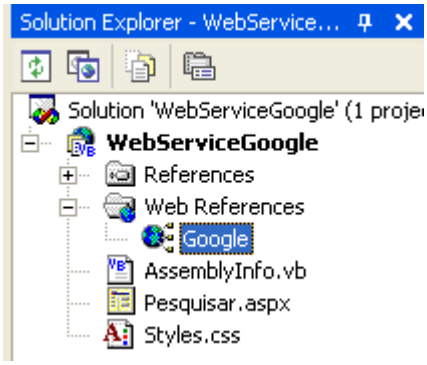


Imagem 30 – Referencia ao Web service

Agora, vamos construir a nossa página aspx para utilizar o serviço de pesquisa do Google, pelo que teremos que adicionar uma caixa de texto onde se digita o texto a pesquisar, uma etiqueta que mostrará o total de resultados encontrados, um botão que executará a pesquisa e uma ListBox que mostrará o resultado da pesquisa.

Após a criação do formulário, teremos que adicionar o código no evento Click do botão pesquisar:

```
Private Sub btPesquisar_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btPesquisar.Click

        ' Variável para guardar a chave de acesso
        Dim ChaveLicenca As String

        ' Variável para acesso ao Web Service do Google
        Dim ServGoogle As Google.GoogleSearchService = New
Google.GoogleSearchService

        ' Variável para receber o resultado da pesquisa
        Dim ResultadoPesquisa As Google.GoogleSearchResult

        ' licença  de acesso
        ChaveLicenca = "tGCTJkYos3YItLYzI9Hg5quBRY8bGqiM"

        ' Executa a pesquisa no Google
        ResultadoPesquisa = ServGoogle.doGoogleSearch(ChaveLicenca,
txtPesquisar.Text, 0, 10, False, "", False, "", "", "")

        ' Exibe o total de itens encontrados
        lblTotal.Text = ResultadoPesquisa.estimatedTotalResultsCount

        'tratar cada um dos resultados obtidos
        Dim UmResultado As Google.ResultElement

        For Each UmResultado In ResultadoPesquisa.resultElements
            lbResultados.Items.Add(UmResultado.title)
        Next
End Sub
```

Observe que a pasta Google passou a ser reconhecida pela aplicação de forma semelhante a um Namespace. Passamos a ver as classes expostas pelo Web Service do Google como se fossem classes locais. Desta forma, podemos instanciar a classe GoogleSearchService e utilizar o método do GoogleSearch para realizarmos a pesquisa.

Os principais parâmetros do método do GoogleSearch são a chave de licença de acesso, o texto a ser pesquisado no google e o valor 10, que indica o número máximo de respostas que desejamos obter. Observe que a limitação deste número não limita a resposta da propriedade estimatedTotalResultsCount, que continua a mostrar o número total de resultados encontrados pelo Google.

Observe que as classes e métodos utilizados a partir da pasta Google são classes e métodos específicos do serviço do Google. Cada Web Service terá classes e métodos personalizados.

Com esse exemplo podemos ter uma boa noção dos recursos que a utilização de WebServices nos oferece. São milhares de novas possibilidades em termos de desenvolvimento de software.

Aula 3 – Criar e chamar um serviço Web usando o VS .NET

Como reparou, o uso de um Web Serviço é muito idêntico ao uso de componentes e a sua criação também é muito semelhante.

Vamos criar um Web Service que disponibiliza os mesmos métodos que o componente criado na aula 4 do módulo passado, ou seja, retorna um DataSet da tabela “Grupo_Sangue”, após uma consulta à base de dados e um outro método que retorna uma DataSet de pessoas de um respectivo grupo de sangue.

Crie um novo projecto do tipo “ASP.NET Web Service” no Visual Studio .NET e dê o nome de “WebService”. O VS .NET cria automaticamente um serviço web muito básico:

```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace :=
"http://tempuri.org/WebService/Service")> _
Public Class Service
    Inherits System.Web.Services.WebService

    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World.
    ' To build, uncomment the following lines then save and build the
project.
    ' To test this web service, ensure that the .asmx file is the
start page
    ' and press F5.
    '
    '<WebMethod()> _
    'Public Function HelloWorld() As String
    '   Return "Hello World"
    'End Function

End Class
```

O Web Service criado pelo VS. NET contém um método que retorna uma frase “Hello Word”.

De notar que é identificado o caminho para o Web Service que, por defeito, é o computador local http://tempuri.org/WebService/Service, que terá que ser alterado caso disponibilize este serviço em outro servidor.

Copie o código do componente criado na aula 4 do módulo passado. Para este serviço web, a única alteração a fazer é adicionar a tag `<WebMethod()> _` antes de cada método:

```
Imports System.Web.Services
```

```
Imports System.Data.OleDb

<System.Web.Services.WebService(Namespace:="http://tempuri.org/WebServ
ice/Service")> _
Public Class Service
    Inherits System.Web.Services.WebService

    Private connString As String =
"PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA
SOURCE=D:\Inetpub\wwwroot\Curso\pessoas.mdb;"

    <WebMethod()> _
    Public Function GrupoSangue() As DataSet
        Dim objConnection As OleDbConnection
        Dim da As OleDbDataAdapter
        Dim ds As DataSet

        Dim sql As String = "Select * FROM Grupo_Sangue ORDER BY GS"

        objConnection = New OleDbConnection(connString)
        objConnection.Open()

        da = New OleDbDataAdapter(sql, objConnection)

        ds = New DataSet
        da.Fill(ds, "Grupo_Sangue")

        objConnection.Close()

        Return ds
    End Function

    <WebMethod()> _
    Public Function Pessoas(ByVal idGrupoSangue As Byte) As DataSet
        Dim objConnection As OleDbConnection
        Dim da As OleDbDataAdapter
        Dim ds As DataSet

        Dim sql As String = "SELECT * FROM Pessoa WHERE GS = " &
idGrupoSangue

        objConnection = New OleDbConnection(connString)
        objConnection.Open()

        da = New OleDbDataAdapter(sql, objConnection)

        ds = New DataSet
        da.Fill(ds, "Pessoa")

        objConnection.Close()

        Return ds
    End Function

End Class
```

Compile o Web Service e se tal for feito com sucesso já temos criado o Web Service. Agora, vamos criar uma aplicação Web para utilizar as funcionalidades deste

serviço, mais uma vez a aplicação será muito idêntica à aplicação criada para usar o componente.

Crie um novo projecto do tipo “ASP.NET Web Application” no VS .NET e dê o nome de “UsarWebService”, adicione uma ListBox e uma DataGrid, tal como no exemplo do uso do componente.

Referencie o Web Service criado tal como na aula 2, e altere o nome para “Service”.

Adicione o seguinte código à nossa aplicação:

```
Dim Servico As WebService.Service

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Variável para acesso ao Web Service
        Servico = New WebService.Service

        ' Executa o método do Web Service para preencher a ListBox
        With Me.lbGS
            .DataTextField = "GS"
            .DataValueField = "IDGS"
            lbGS.DataSource =
Servico.GrupoSangue().Tables("Grupo_Sangue").DefaultView
            lbGS.DataBind()
        End With
End Sub

Private Sub lbGS_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles lbGS.SelectedIndexChanged
        Me.dgPessoas.Visible = True
        'preenche o datagrid
        With Me.dgPessoas
            .DataSource =
Servico.Pessoas(Me.lbGS.SelectedItem.Value).Tables("Pessoa")
            .DataBind()
        End With
End Sub
```

Como pode ver, no código não há muitas diferenças entre o uso de componentes e Web Services.

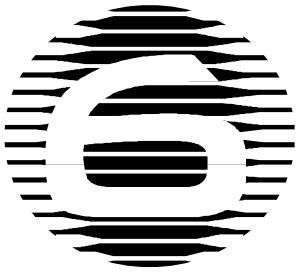
Agora, compile a aplicação e execute. Se tudo funcionar da forma desejada, terá o mesmo resultado do uso do componente, só que desta vez será usado um serviço web.

Auto-avaliação do módulo 5

Elabore um Web Service que disponibilize as temperaturas e o estado do tempo nas principais cidades Portuguesas. Deve usar a base de dados meteo.mdb e o Web Service deverá disponibilizar os seguintes métodos:

- Retornar um Data Set com todas as temperaturas e estado do tempo;
- Retornar a temperatura de uma determinada cidade;
- Retornar o estado do tempo numa determinada cidade;
- Retornar a temperatura mais elevada no momento e respectiva cidade;
- Retornar a temperatura mais baixa no momento e respectiva cidade;

Curso\meteo.mdb
Projecto Curso\WSMeteo



Módulo 6 – Criação de uma aplicação Web ASP.NET

Objectivos do módulo

Este módulo final tem como objectivo fortalecer os conhecimentos na criação de aplicações web mais elaboradas em ASP.NET, quais os seus requisitos e configurações.

Sumário

Aula 1 – Segurança em ASP.NET

Nesta aula, o aluno aprenderá algumas noções de segurança em aplicações ASP.NET.

A segurança deverá ser uma das primeiras preocupações no desenvolvimento de aplicações Web, pois, se deixar para o final, a aplicação poderá não estar apta a receber o tipo de segurança que é desejado e levar a uma reestruturação completa da aplicação. Nesta aula, vamos ver qual a segurança que se deve aplicar às aplicações Web.

Três componentes diferentes compõem a arquitectura de segurança de uma aplicação: integridade de dados, autenticação e autorização. A integridade dos dados poderá ser assegurada utilizando algumas medidas de criptografia para impedir que outros possam interceptar, ler ou alterar os dados trocados entre o browser e o servidor Web. Poderá usar-se HTTP pelo protocolo Secure Sockets Layer (SSL) para codificar os dados. A utilização do SSL é bastante directa e não requer nenhuma codificação, mas tem de se obter primeiro um certificado digital de uma autoridade em certificados. O browser usa este certificado para determinar se o servidor é o que reivindica ser. Por outras palavras, o browser usa o certificado para autenticar o servidor. O SSL deverá ser usado em sites Web que trocam informações sigilosas como números de cartão de crédito. O SSL também é útil com certos mecanismos de autenticação que necessitam de criptografia.

O objectivo da autenticação é assegurar que os utilizadores sejam quem eles reivindicam ser. As técnicas de autenticação envolvem um prompt, solicitando ao utilizador um ID e uma senha, enquanto conferem com a lista de userIDs e senhas. Comunicar o userID e a senha (ou informação equivalente) do cliente para o servidor sem comprometer essa informação, é a questão principal sobre autenticação em Aplicações Web.

O IIS5 Oferece Opções de Autenticação

O IIS 5.0 oferece três opções de autenticação além de anonymous (anónimo) (nenhuma autenticação). As duas primeiras, Básica e Sumária, são consideradas padrões da Internet. Porém, há alguns problemas com a autenticação Básica. Ela,

simplesmente, envia o userID e a senha ao servidor em texto codificado em Base64. Um hacker não só pode interceptar esta mensagem e obter o userID e a senha, como o servidor obtém a sua senha em texto não codificado. Este é um caso claro de utilização do SSL para complementar o mecanismo de autenticação, oferecendo autenticação de servidor e criptografia necessários para impedir que outros interceptem a senha do utilizador. O Netscape e o Internet Explorer (IE) suportam a autenticação Básica. Isto torna a combinação da autenticação Básica com SSL, uma boa estratégia para aplicações de Internet, nos quais não se pode requerer um browser específico.

A autenticação Sumária é nova no HTTP 1.1, e só o IE5 o suporta. Isto significa que a autenticação Sumária não é prática para a utilização na Internet, neste momento. A autenticação Sumária é considerada mais segura que a Básica, porque não envia a senha através de conexão. Ao contrário, o cliente prova que sabe a senha, transmitindo a soma de verificação (checksum) ao servidor. (Uma soma de verificação - checksum - é um valor calculado, usando a informação original: o userID, a senha, o método HTTP, a URL solicitada e uma chave gerada no servidor.) O servidor usa esta informação para calcular a soma de verificação (checksum) e compara-la à soma de verificação (checksum) enviada pelo cliente. Isto, porque o servidor precisa ter acesso à informação original. O servidor calcula uma soma de verificação (checksum) semelhante, utilizando os mesmos dados, e então compara com aquela que foi enviada pelo cliente.

Para que este processo funcione, o servidor tem que ter acesso à senha do utilizador em texto não codificado. A maneira como o IIS5 implementa a autenticação Sumária exige que o servidor Web também aja como seu controlador de domínio, e que você verifique a opção Active Directory que permite armazenar senhas de utilizador como texto não codificado. Dadas as exigências e o facto de que a autenticação Básica com criptografia SSL é uma alternativa prática para a maioria dos browsers, a autenticação Sumária não será usada.

A terceira opção, autenticação Integrada Windows, permite ao servidor autenticar o cliente sem ter de solicitar um userID e senha. Ao invés, o servidor usa as actuais credenciais do utilizador para a autenticação.

Escolha a Melhor Opção

Para aplicações Internet, a melhor opção é usar a autenticação Básica combinada com SSL.

Todos os mecanismos de autenticação exigem que se definam os utilizadores da aplicação como utilizadores Windows no domínio ou no servidor Web.

Pode criar-se o próprio esquema de autenticação, usando um formulário que solicita userID e a contra-senha, comparando-os a uma lista de userIDs e senhas conhecidas mantidas numa base de dados no servidor (como fizemos nas aulas).

O ASP.NET possui o recurso embutido de autenticação baseada em cookie, que oferece uma estrutura para fazer exactamente o que queremos. Quando um browser solicita uma página numa aplicação que utiliza autenticação cookie ASP.NET, o servidor verifica o cookie que indica se o utilizador foi autenticado. O servidor redirecciona o browser à página de login, se não encontrar o cookie (como já vimos). Ao contrário dos outros métodos de autenticação, a página de login é uma página ASP.NET que contém um formulário em HTML simples com dois campos de entrada para o userID e senha, como também qualquer outra coisa). O utilizador digita um userID e uma senha e envia o formulário de volta ao servidor. Deve utilizar-se sempre SSL na página de login para prevenir a transmissão dos dados do utilizador como texto não codificado. No servidor, utilize a classe CookieAuthentication.Authenticate para verificar se os dados fornecidos são válidos. A classe CookieAuthentication é parte do.NET framework.

Aula 2 – Configurar aplicações ASP.NET

Quando criamos uma aplicação WEB ASP.NET, existem dois ficheiros que merecem a nossa atenção: *Global.asax* e *Web.config,* dos quais já falamos nas nossas aulas.

Com o ficheiro *Global.asax*, pode executar um determinado código quando a aplicação é iniciada/finalizada ou até mesmo quando qualquer página da Aplicação for requisitada. Já o ficheiro *Web.Config* armazena as configurações da aplicação, tais como: Estado de Sessão, Segurança, Globalização, etc.

Quando cria aplicações com o Visual Studio .NET, ao criar uma nova aplicação do tipo “ASP.NET WebApplication”, o ficheiro *Global.asax* e o ficheiro *Web.Config* são adicionados automaticamente à aplicação.

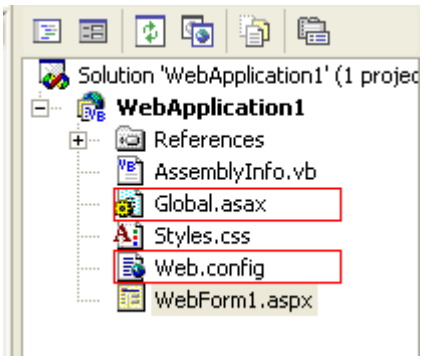


Imagem 31 – Ficheiros Global.asax e Web.Config

O Directório \bin

Dentro da raiz da aplicação existe um directório chamado "\bin". Dentro desse directório estão armazenados ficheiros com formato binário compilados e utilizados pela aplicação (um exemplo são os ficheiros com extensão *.dll).

Os assemblies colocados aqui estão automaticamente disponíveis para serem usados nos ficheiros *.aspx, e com isso não precisamos de nos preocupar com os procedimentos complexos de criação.

O ficheiro Global.asax

O directório Virtual no IIS é grande parte da aplicação ASP.NET. Independentemente da página, a aplicação é iniciada na primeira vez que é solicitada. Enquanto os utilizadores navegam pelas páginas, o processamento ocorre em segundo plano.

A aplicação poderá ser reiniciada da mesma forma que qualquer aplicação tradicional, com a seguinte diferença: enquanto uma aplicação tradicional é iniciada e executada num computador desktop, permitindo a interacção directa com o utilizador, uma aplicação ASP.NET é iniciada e executada num servidor Web, e o utilizador utiliza um browser para aceder à aplicação.

Lembrando que tudo em .NET Framework são objectos, temos um objecto chamado *HttpApplication*, que nos fornece métodos e eventos. Com isso, podemos deduzir que sempre que uma página do directório virtual for solicitada pela primeira vez, um objecto do tipo *HttpApplication* é instanciado.

Como as páginas ASP.NET, realizamos uma ou mais tarefas individualmente, elas não controlam a aplicação de modo geral, não podendo uma página afectar directamente a outra. Deve existir uma localização central que controla a execução da aplicação. E é neste contexto que entra o ficheiro *Global.asax*.

Conhecido como um ficheiro de aplicação ASP.NET, o *Global.asax* permite-nos programar no lugar do objecto *HttpApplication* e com isso pode-se controlar a aplicação ASP.NET como se faz com qualquer outro objecto por meio de métodos e eventos.

Apesar do Visual Studio .NET incluir o arquivo *Global.asax* por default, ele é totalmente opcional. Se a aplicação não contiver um ficheiro desse tipo, a aplicação opera de forma padrão. Desejando adicionar funcionalidades, a sua utilização torna-se essencial.

O ficheiro *Global.asax* é colocado no directório raiz da aplicação (Exemplo: http://servidor/site/ - c:\inetpub\wwwwroot\site\). O ASP.NET controla o acesso a esse ficheiro, de modo que não seja acessível através do browser, o que garante a segurança.

Programar o ficheiro Global.asax

O ficheiro *Global.asax* opera de forma semelhante às páginas *.aspx. O ficheiro *Global.asax* é utilizado para sincronizar qualquer evento exposto pela classe *HttpApplication*.

Eventos que estão no quadro abaixo:

Evento	Descrição
AcquireRequestState	Accionado quando a aplicação obtém a cache para a solicitação.
AuthenticateRequest	Accionado quando a aplicação tenta autenticar a solicitação de HTTP.
AuthorizeRequest	Accionado quando a aplicação tenta autorizar a solicitação de HTTP.
BeginRequest	Accionado quando a solicitação de HTTP é iniciada.
EndRequest	Accionado quando a solicitação de HTTP é concluída.
Error	Accionado quando surge um erro.
PostRequestHandlerExecute	Accionado imediatamente depois do handler de HTTP processar a solicitação.
PreRequestHandlerExecute	Accionado imediatamente antes do handler de HTTP processar a solicitação.
PreSenderRequestContent	Se a solicitação tiver conteúdo adicional (QueryString, Variáveis de Formulário, etc.), este evento é accionado imediatamente antes daquele conteúdo ser recebido.
PreSenderRequestHeaders	Accionado imediatamente antes de os cabeçalhos de solicitação serem recebidos.
ReleaseRequestState	Accionado quando o Aplicativo libera o estado de sessão para a solicitação.
ResolveRequestCache	Accionado quando o Aplicativo determina a cache para a solicitação.
UpdateRequestCache	Accionado quando o Aplicativo actualiza e liberta a cache para a solicitação.

O ficheiro *Global.asax* padrão criado pelo VS .NET:

```
Imports System.Web
Imports System.Web.SessionState

Public Class Global
    Inherits System.Web.HttpApplication


    Sub Application_Start(ByVal sender As Object, ByVal e As
EventArgs)
        ' Fires when the application is started
    End Sub
```

```
    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Fires when the session is started
    End Sub

    Sub Application_BeginRequest(ByVal sender As Object, ByVal e As
EventArgs)
        ' Fires at the beginning of each request
    End Sub

    Sub Application_AuthenticateRequest(ByVal sender As Object, ByVal
e As EventArgs)
        ' Fires upon attempting to authenticate the use
    End Sub

    Sub Application_Error(ByVal sender As Object, ByVal e As
EventArgs)
        ' Fires when an error occurs
    End Sub

    Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
        ' Fires when the session ends
    End Sub

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
        ' Fires when the application ends
    End Sub

End Class
```

O código mostrado tem apenas os Eventos que o próprio Visual Studio .NET cria, mas poderão ser usados mais Eventos.

A ordem de execução dos eventos é a seguinte:

1. Application_Start
2. Application_BeginRequest
3. Application_AuthenticateRequest
4. Application_AuthorizeRequest
5. Application_ResolveRequestCache
6. Session_Start
7. Application_AcquireRequestState
8. Application_PreRequestHandlerExecute
9. Page_Load (arquivo *.aspx) ou qualquer outra saída de página
10. Application_PostRequestHandlerExecute
11. Application_ReleaseRequestState
12. Application_UpdateRequestCache
13. Application_EndRequest
14. Application_PreSendRequestHeaders

Vale a pena lembrar que alguns eventos são executados de acordo com alguma circunstância. Um exemplo disso é o caso do Evento Session_Start, que somente é executado quando qualquer página é solicitada por um utilizador, a partir da segunda vez/página, o Evento não ocorre novamente para esse utilizador.

Da mesma forma que é importante controlar o processamento da aplicação, é necessário configurá-la. Controle de acesso, Segurança, Estado de sessão e até mesmo configurações personalizadas. Para isso, o ASP.NET fornece um ficheiro baseado em texto, que nos dá extensibilidade e fácil configuração.

Além disso, a configuração é hierárquica, ou seja, as informações de configuração de aplicações são aplicadas de acordo com a estrutura de directórios Virtuais do seu site. Os sub-directórios podem herdar ou anular opções de configuração dos seus directórios-pai.

Por padrão, todos os directórios são herdados de um ficheiro de configuração padrão de sistema chamado de *machine.config*, (localizado em: "WinNT\Microsoft.NET\Framework**Versão**\CONFIG).

O ficheiro que é responsável pela configuração é o *Web.Config*, que é um ficheiro XML.

O ficheiro Web.config

Dentro deste ficheiro não existe nada de muito significativo, a não ser que ele contém chaves e valores que são reconhecidos pelo ASP.NET. Tais valores são facilmente modificáveis, podendo-se adicionar chaves próprias para controlar outras operações que o ASP.NET não conhece.

A estrutura básica deste ficheiro é a seguinte:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

      <configSections>
      </configSections>

      <system.web>
      </system.web>

      <system.net>
      </system.net>

</configuration>
```

Como vemos, é somente XML. Entre as tags <configuration> existem dois elementos diferentes: handlers de secção de configuração e configurações da secção de configurações.

As configurações que configuram a aplicação são pares de chave/valor. Há dois tipos destas secções: *system.net* e *system.web*. A primeira secção é para configurar o tempo de execução da .NET em si. Por sua vez, a segunda é para controlar o ASP.NET. As configurações personalizadas serão colocadas nas tags <system.web>..

O ficheiro *Web.Config* estabelece a distinção entre letras maiúsculas e minúsculas, e sem o formato correcto, a aplicação poderá gerar erros.

Abaixo, uma tabela com as configurações disponíveis para a utilização no ficheiro *Web.Config*:

Seção	**Descrição**
<appSettings>	Utilizada para armazenar as suas próprias configurações personalizadas da aplicação.
<authentication>	Configura como o ASP.NET autentica os seus utilizadores.
<authorization>	Configura a autorização de recursos no ASP.NET.
<browsercaps>	Responsável por controlar as configurações do componente de capacidades do navegador.
<compilation>	Responsável por todas as configurações de compilação.
<customErrors>	Indica como exibir erros no navegador.
<globalization>	Responsável por configurar as opções de globalização.
<httpHandlers>	Responsável pelo mapeamento de URLs de entrada em classes *IHttpHandler*.
<httpModules>	Responsável por configurar Módulos de HTTP dentro de um aplicativo.
<identity>	Controla como o ASP.NET acede aos seus recursos.
<location>	Controla como as configurações se aplicam a um directório.
<pages>	Controla configurações de páginas.
<processModel>	Configura as configurações de modelo de processo do ASP.NET em Sistemas de Servidor da Web do IIS.
<sessionState>	Configura o Estado de Sessão.
<trace>	Configura o Trace (Rastreamento).
<webServices>	Controla as configurações dos Serviços da Web.

Como podem ver, o arquivo *Web.Config* ajuda a tornar a aplicação bastante flexível, ou seja, podemos definir funcionalidades globais num único lugar. Além disso, uma das vantagens é que se houver a necessidade de mudar algo dentro do ficheiro *Web.Config*, não há a necessidade de recompilar a aplicação.

Vimos que o arquivo *Global.asax* permite-nos controlar quase todos os aspectos do processamento de uma página ASP.NET. Podem utilizar-se os Eventos do objecto *HttpApplication* para realizar operações imperceptíveis para o utilizador, tornando a aplicação muito mais robusta e eficiente. Além disso, podemos tornar a nossa aplicação bastante flexível, utilizando o arquivo de configuração *Web.Config*, fazendo com que a mesma possa reagir rapidamente a qualquer mudança.

Aula 3 – Exemplo prático "Loja on-line"

Para finalizar o nosso curso, vamos criar uma aplicação Web ASP.NET.

Assim, a nossa aplicação será uma "Loja on-line", tipo supermercado. Nesta aula, o aluno não terá ajuda, terá de solucionar os problemas sozinho.

A base de dados a utilizar já está criada em loja.mdb, analise-a bem para iniciar a criação da aplicação.

A aplicação vai ser criada na pasta Restrita e o controlo de acessos já foi criado em aulas anteriores, com a base de dados acessos.mdb e o ficheiro login.aspx.

O que vamos fazer neste exemplo não é nada de novo. Vamos, apenas, juntar o que aprendemos para criar uma aplicação mais elaborada.

A aplicação vai ser criada com a ajuda do VS .NET, pelo que vamos abri-lo e criar um novo projecto "ASP.NET Web Application" na pasta restrita. A esse projecto daremos o nome de Loja.

O código a ser criado não vai ser mostrado, apenas serão dadas indicações para a criação da Loja, uma vez que o aluno, neste momento, já terá capacidade de desenvolver aplicações mais elaboradas.

Ao analisar a base de dados, reparou que os produtos estão divididos por categorias, as quais terão que ser listadas para, posteriormente, visualizar os produtos dessa categoria. Para listar as categorias, deve usar-se uma DataList que permite que, ao ser selecionado um item dessa DataList, seja preenchida uma DataGrid com os produtos dessa categoria.

A DataGrid terá que ser presonalizada para que se possa comprar uma determinada quantidade de um produto.

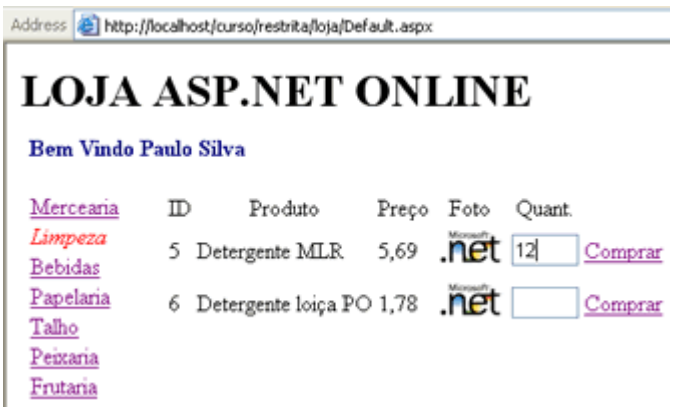


Imagem 32 – Loja On-Line

Depois, temos que criar o carrinho de compras. A criação do carrinho de compras é muito simples, uma vez que os produtos comprados estão guardados na base de dados, por isso só temos que fazer uma consulta à base de dados e preencher uma DataGrid com os produtos comprados até ao momento.

ID	Produto	Foto	Preço	Quantidade	Valor	
1	Arroz AZZ		1,1	3	3,3	Editar
2	Azeite OZT		3,25	2	6,5	Editar
13	Lápis DRG		1,45	3	4,35	Editar
14	Marcadores SSS		4,12	1	4,12	Editar

Imagem 33 – Carrinho de compras

E se o utilizador quiser alterar as quantidades do carrinho? Temos que implementar o comando actualizar na DataGrid carrinho, tal como fizemos na aula em que falámos de DataGrids.

Por fim, só temos que adicionar uma label, que informa o valor total do carrinho.

Projecto Curso\Restrita\Loja

Como reparou, a criação de aplicações mais elaboradas em ASP.NET torna-se simples com o uso dos componentes existentes no ASP.NET.

Agora que terminou este curso, já poderá criar aplicações elaboradas em ASP.NET, e se já usou os ASP clássicos, reparou que esta é uma nova forma de criar aplicações web.

A partir deste momento, tudo está nas suas mãos…

Auto-avaliação do módulo 6

Usando o exercício da Loja online, adicione mais funcionalidades a seu gosto, conferindo um aspecto mais profissional à loja.

3. Conclusão

Ao longo deste trabalho, dediquei-me à elaboração de um curso ASP.NET em modo de e-Learning, apresentando vários módulos, que, por sua vez, se encontravam divididos por aulas; alguns exercícios e a respectiva correcção.

O objectivo primordial centrava-se na apreensão, a curto prazo, da matéria leccionada, tendo sido alcançado através do planeamento das aulas e dos temas, de modo a apresentarem uma sequência lógica e um nível crescente de dificuldade pouco perceptível para o aluno e que, por esse motivo, não funciona como elemento desmotivador.

Quando iniciei este projecto não possuía conhecimentos relevantes de ASP.NET.

Porém, após a conclusão do trabalho, considero que domino a tecnologia, o que pode ser um factor positivo de avaliação do curso.

Uma vez que a minha experiência na área de e-Learning era nula, este não é um curso perfeito, visto que isso só seria possível com a experiência e com o tempo.

O e-Learning que muito certamente vai ser uma ferramenta valiosa nos tempos que vem e quem sabe depois de ter ganho experiência possa construir um curso ainda mais aprofundado de ASP.NET que poderia ser usado por colegas que estejam interessados em conhecer e dominar a tecnologia.

Assim, considero positivo o balanço deste projecto e considero atingidos os objectivos iniciais.

4. Bibliografia

- www.asp.net – site dedicado ao ASP.NET
- www.msdnbrasil.com.br – site de ajuda ao desenvolvimento
- www.linhadecodigo.com.br – site de ajuda ao desenvolvimento