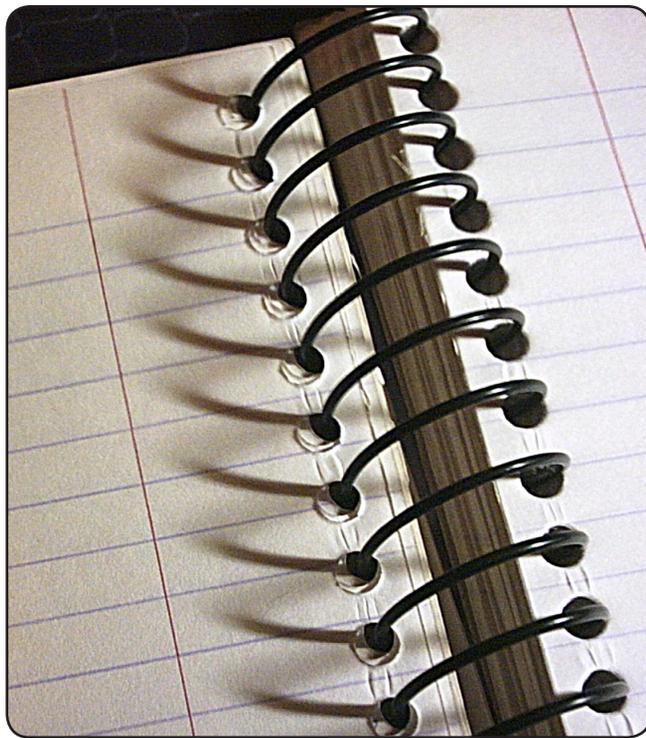


Nesta seção você encontra artigos para iniciantes em .net



#### De que se trata o artigo

- Introdução à aplicações ASP.NET;
- Principais características;
- Consumindo dados de bancos de dados ACCESS;
- Cuidados iniciais com segurança;
- Publicando a página no IIS.

#### Para que serve

- Demonstrar os principais conceitos para a criação de aplicações WEB baseadas em ASP.NET e também a integração destas com bancos de dados;
- Discutir alguns pontos relacionados com segurança ao desenvolver este tipo de programa (limitação de strings, cuidado com Script Injection e Sql Injection);
- Fazer a publicação da página no servidor IIS.

#### Em que situação o tema é útil

Páginas Web dinâmicas cujo conteúdo é elaborado a partir de informações passadas pelo usuário, que utilizem ou não consulta a bancos de dados, ou até que substituam programas Windows que irão precisar de pouca ou nenhuma configuração são candidatos naturais a utilização do ASP.NET.



#### Vladimir Rech

[vladimirrech@yahoo.com.br](mailto:vladimirrech@yahoo.com.br)

Tecnólogo em Desenvolvimento de Sistemas pelo CEFET-PR, palestrante; trabalha com desenvolvimento de sistemas em .NET destacando-se aplicações Windows, ASP e Web Services.

## Desenvolvendo aplicações Web com ASP.NET

Conceitos, consumindo dados de bancos de dados e cuidados com segurança



### Resumo do DevMan

Páginas da Web dinâmicas são muito comuns na maioria dos sites da Internet.

Com ASP.NET e Visual Studio, o desenvolvimento destas é bastante produtivo e rápido sendo que a maior complexidade fica por conta do Framework .NET que resolve questões de comunicação e protocolos.

O desenvolvedor deverá preocupar-se basicamente com as regras de negócio que se deseja implementar além de alguns cuidados com segurança que serão brevemente expostos neste artigo.

**A**SP.NET é a última versão da tecnologia *Active Server Pages (ASP)* da Microsoft fazendo parte do *Framework Microsoft .NET* sendo uma ferramenta poderosa para a criação de páginas Web dinâmicas. Antes de continuar você precisa ter um mínimo de conhecimento dos seguintes assuntos:

- WWW, HTML, XML e a noção básica para construção de páginas Web.
- Linguagens de script como *Java Script* ou *VB Script*.
- Conhecimento básico de scripts do lado do servidor como ASP ou PHP.
- Conhecimento de SQL (*Structured Query Language*) básico para poder executar consultas a bancos de dados.
- Noções de linguagem de programação como o C#.

## Nota

Embora Java Script e VB Script sejam também linguagens de script usadas em páginas Web, estas diferem basicamente de outras linguagens como o ASP ou PHP por serem executadas do lado do cliente, ou seja, na máquina que está visualizando a página. Estas linguagens não serão tratadas neste artigo ficando para outra oportunidade, porém é importante citá-las para saber que estão relacionadas com a criação de páginas dinâmicas e desenvolvimento de aplicações Web.

## ASP e ASP.NET

ASP é a sigla para *Active Server Pages*. É uma tecnologia para desenvolvimento de páginas Web baseadas em *scripts* executados no servidor. Estes *scripts* são interpretados e geram dinamicamente as marcações HTML (*tags*) para o *browser* visualizar a página solicitada.

ASP.NET é o próximo passo nesta tecnologia. Não se trata de uma nova versão de ASP ou uma atualização desta tecnologia, mas uma nova geração desta tecnologia.

Atualmente na versão 3.0 esta tecnologia possui pouquíssima compatibilidade com o ASP original. Neste artigo iremos tratar da versão 2.0.

Alguns outros fatos sobre ASP.NET e suas tecnologias relacionadas:

- ASP.NET é um programa que é executado dentro do servidor IIS – *Internet Information Server*.
- O IIS é um componente do sistema operacional *Windows* para as versões *Server (2000 e superiores)* e para as versões *desktop 2000, XP e Vista*. É o IIS que gerencia a execução do ASP.NET.
- Os arquivos ASP.NET são semelhantes a arquivos HTML e podem conter códigos em HTML, XML e *scripts*.
- Os arquivos ASP.NET devem ter a extensão “.aspx” e inicialmente podem existir *stand alone*, ou seja sem fazer parte de um projeto necessitando, entretanto, estarem em um site configurado no IIS.

Vamos criar uma página usando o *Visual Studio 2005* para demonstrar o código básico de um arquivo ASP.NET.

Abra o *Visual Studio* e execute os comandos *File -> New -> File*. Na janela que se abre, no campo *Categories* clique no “+” do lado de *Web* e escolha “C#” que será a linguagem de script que iremos usar no artigo. No campo *Templates* escolha *WebForm* para criar uma página ASP.NET. A **Figura 1** ilustra este processo.

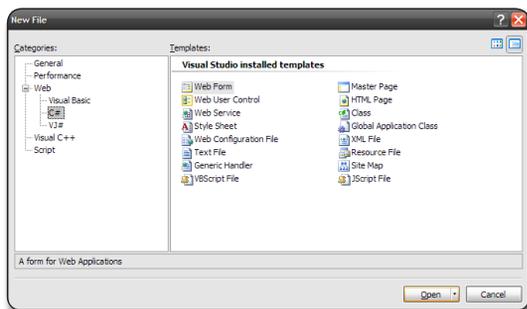


Figura 1. Criando um arquivo ASP.NET

Se ao criar a sua página for exibido o modo *design*, clique em “*source*” na base da janela para que possamos digitar o código ASP/HTML diretamente.

Embora seja mais complicado vamos iniciar digitando o código para a compreensão da estrutura da página ASP.NET.

Estes passos criam um arquivo que não está associado a nenhuma aplicação ainda. Digite o conteúdo como está exibido na **Listagem 1**.

### Listagem 1. Exemplo de código ASP.NET

```
1 <!-- Sintaxe inicial de uma página ASPX -->
2 <%@ Page Language="C#" %>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
4
5
6 <!-- Script a ser executado no servidor -->
7 <script runat="server">
8     private void submit(object sender, EventArgs e)
9     {
10         this.BtnExemplo.Text = "Olá! Você me clicou!";
11     }
12 </script>
13
14 <!-- TAGS Html para formatação -->
15 <html xmlns="http://www.w3.org/1999/xhtml" >
16 <head runat="server">
17     <title>Untitled Page</title>
18 </head>
19 <body>
20     <form id="form1" runat="server">
21     <div>
22         <asp:Label ID="lblTexto" runat="server"
23             Text="Clique no botão:" />
24
25         <!-- Executa a chamada para a função em C# no
26             SCRIPT no topo da página -->
27         <asp:Button ID="BtnExemplo" Text="Clique aqui!"
28             runat="server" OnClick="submit" />
29     </div>
30 </form>
31 </body>
32 </html>
```

O exemplo anterior é bastante simples. Consiste de dois controles ASP: um rótulo de texto que é usado para escrevermos uma mensagem para o usuário que é indicado pela tag `<asp:Label ...>` na linha 22 e um botão que irá disparar o script em C#, representado pela tag `<asp:Button>`. No topo da página a partir da linha 7 observamos o script escrito em C#.

Para ver a página em execução clique com o botão direito sobre o código e escolha a opção *View in Browser*. O resultado deve ser parecido com o que é apresentado na **Figura 2**.

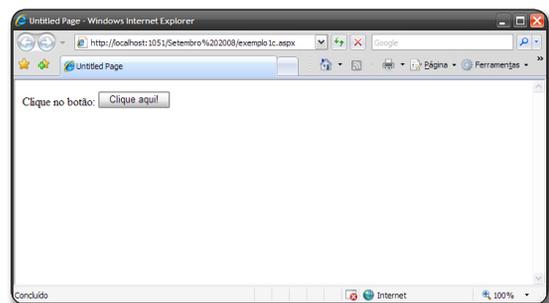


Figura 2. Página ASP.NET sendo executada

Uma questão a considerar nesta página é que tudo o que foi gerado para o *browser* foi código HTML. Observe este, clicando com o botão direito do mouse sobre a página e escolha *Exibir Código-Fonte*. Observe na **Listagem 2**, que nenhum detalhe do *script* é enviado para o *browser* bem como as *tags ASP.NET* foram substituídas por elementos HTML.

**Listagem 2.** Código HTML enviado para o browser

```

1 <!-- Sintaxe inicial de uma página ASPX -->
2
3 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
  transitional.dtd">
4
5
6 <!-- Script a ser executado no servidor -->
7
8
9 <!-- TAGS Html para formatação -->
10 <html xmlns="http://www.w3.org/1999/xhtml" >
11 <head><title>
12   Untitled Page
13 </title></head>
14 <body>
15   <form name="form1" method="post" action="exemplo1c.
  aspx" id="form1">
16 <div>
17 <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
  value="/wEPDwUKMTM5NjcyNzYyM2Rk5qfpDXpkM0HHfOckEft8yibW3o=" />
18 </div>
19
20 <div>
21 <span id="lblTexto">Clique no botão:</span>
22
23 <!-- Executa a chamada para a função em C# no
  SCRIPT no topo da página -->
24 <input type="submit" name="BtnExemplo"
  value="Clique aqui!" id="BtnExemplo" />
25 </div>
26
27 <div>
28
29 <input type="hidden" name="__EVENTVALIDATION"
  id="__EVENTVALIDATION" value="/
  wEWAgLmv4r1DALXiN1dAkjbmmTzEmdEBIRyUI5EJYjym0k" />
30 </div></form>
31 </body>
32 </html>

```

## Recursos necessários para desenvolver

O desenvolvimento de uma aplicação *ASP.NET* requer além dos conhecimentos citados no começo do artigo o uso de ferramentas próprias como um gerenciador de páginas *WEB*, no nosso caso o próprio *IIS* que vem instalado na maioria das instalações do *Windows* como vimos acima, do *Framework .NET* para a compilação das páginas e no nosso caso, do *Visual Studio 2005* ou *2008*. Como recursos adicionais para o desenvolvedor pode-se citar:

- Diversos *browsers* em diversas versões para poder realizar testes de compatibilidade das páginas;
- Se a aplicação for consumir dados de banco de dados, um gerenciador de bancos de dados compatível com o *.Net* como o *Microsoft Sql Server*, *Microsoft Office Access* ou mesmo bancos de dados *Open Source* como o *Firebird*;
- Conexão com a Internet para poder publicar o site no servidor destino da aplicação.

Ainda que tenhamos citado aqui o uso do *Visual Studio*, para quem deseja aprender a programar a *Microsoft* disponibiliza ferramentas gratuitas conhecidas como *Express*.

Verifique se o seu *Windows* tem o *IIS* instalado, para isso,

abra o *Painel de Controle*, *Ferramentas Administrativas* e verifique se existe o ícone *Internet Information Services*. Caso não exista você precisará instalar para poder fazer testes com sua aplicação. Para fazer isso, abra o *Painel de Controle*, a opção *Adicionar ou Remover programas*. Na janela que se abre, escolha *Adicionar/Remover componentes do Windows* e marque a opção *Internet Information Services (IIS)* como demonstrado na **Figura 3**. Pode ser necessário fornecer o disco de instalação do *Windows*.



**Figura 3.** Instalando o IIS

### Nota

a partir do VS2005, a MS disponibiliza um servidor de testes local que dispensa o uso do IIS para teste de Web Sites.

## Recursos necessários para hospedar

A "hospedagem" de uma aplicação *ASP.NET* consiste em copiá-la para um servidor onde este irá fazer o gerenciamento da aplicação tratando de como os usuários irão conectar-se com esta aplicação, gerenciar permissões, torná-la visível externamente (na Internet) e manter um registro dos eventos que ocorrerem com esta aplicação.

As versões *Server* do *Microsoft Windows* possuem um servidor nativo: o *Internet Information Service* onde todas as tarefas relacionadas com o gerenciamento da aplicação e sua instalação são feitos. É também através do *IIS* que definimos a localização da aplicação fisicamente no servidor e como ela será visível externamente ao servidor.

Nas versões *desktop* do *Windows* principalmente a partir da 2000 o *IIS* também pode ser instalado sendo isto indicado apenas para o desenvolvimento e testes uma vez que estas versões deste S.O. não possuem um gerenciamento avançado deste tipo de aplicação.

Também, se a aplicação *ASP.NET* for conectar-se com um banco de dados, pode ser interessante manter-se um gerenciador de banco de dados instalado no servidor que hospeda a página, embora, seja possível com o *ASP.NET* conectar-se com servidores de bancos de dados localizados remotamente, em outros servidores.

## Definindo a interface com HTML/ASP

No primeiro exemplo nós escrevemos a *interface* com o usuário da nossa aplicação manualmente, sem fazer uso de um design para isso.

Tal prática é muito útil para otimização do código HTML gerado já que muitas vezes os geradores automáticos de código – como o contido no *Visual Studio* – pode gerar código desnecessário, mas, sem dúvida é bem mais produtivo fazermos isso visualmente arrastando componentes para a nossa página.

Vamos ver como fazer isto criando um projeto de uma aplicação *ASP.NET* completa.

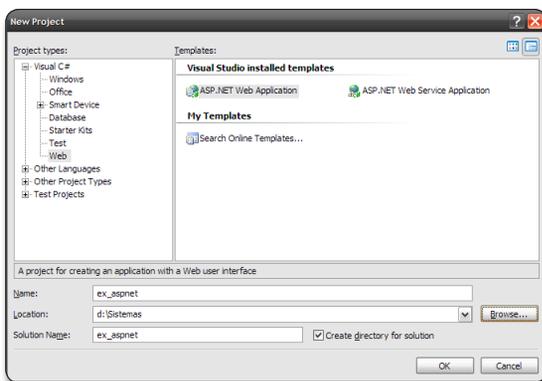
O objetivo desta aplicação será consultar dados de um banco de dados em *Access*. Desta forma veremos também como conectar-se com este tipo de banco de dados.

No *Access* crie uma base de dados (um arquivo) chamado *xopt.mdb*, e nele crie uma tabela chamada *Funcionarios* com a estrutura demonstrada na **Tabela 1**.

Nome do Campo	Tipo do Dado	Tamanho	Casas Decimais
ID	Inteiro – auto incremento, chave primária		
Nome	Texto	50	
Cargo	Texto	50	
Area	Texto	50	
Nascimento	Data/Hora		
Admissão	Data/Hora		
Salário	Número		2

**Tabela 1.** Estrutura da tabela Access “Funcionarios”

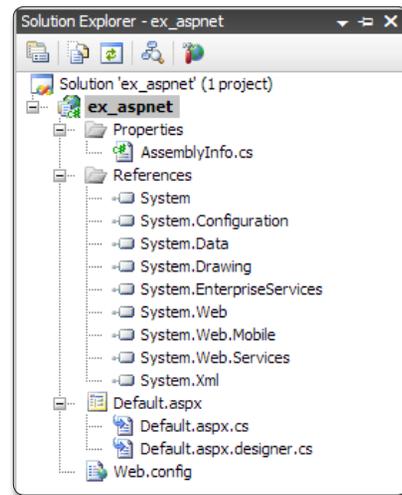
O próximo passo é criar o nosso projeto do site no *Visual Studio*. Abra-o e execute as opções de menu: *File – New – Project...* Na janela *New Project* em *Project Types* escolha *Visual C#* e *Web*. No campo *Templates* marque *Asp.net Web Application*. Digite um nome para seu projeto em *Name*, escolha a pasta onde o projeto se rá criado no campo *Location*, lembrando que esta pasta irá armazenar todos os arquivos necessários para desenvolver o projeto. Escolha um nome para a *Solution* que é o nome dado para o projeto pelo *Visual Studio* e pode conter vários projetos numa mesma *Solution*. Veja na **Figura 4** a tela para a criação do projeto.



**Figura 4.** Janela para criação do projeto ASP.NET

Escolhemos o *C#* mas em uma página *ASP.NET* poderíamos usar também o *Visual Basic*.

Quando criamos o novo projeto, vários outros arquivos são criados automaticamente. São arquivos e pastas que são usados pelo *Visual Studio* durante o projeto e um destaque para o arquivo *Web.Config* que armazena as configurações da aplicação. Para ver os arquivos contidos no projeto clique em *Solution Explorer* (**Figura 5**.)



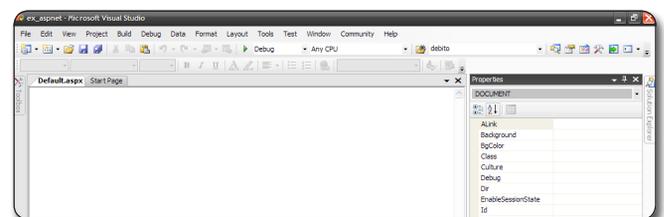
**Figura 5.** Solution Explorer

Um arquivo criado automaticamente é a página principal do projeto, geralmente *Default.aspx* ou *Index.aspx*.

Vamos trabalhar nesta página para definir a página para consulta de dados.

O *Visual Studio 2005* pode trabalhar com as páginas *ASP.NET* basicamente em dois modos: exibindo o código-fonte em HTML ou em modo *design*, onde uma prévia da página é exibida dentro do editor e podemos arrastar e soltar componentes. Para alternar entre uma visão e outra, clique nos botões *Design* e *Source* alternadamente que estão localizados na parte inferior da janela do *Visual Studio*.

Quando estamos trabalhando no modo *Design* (**Figura 6**) destaca-se além do espaço onde iremos definir a interface com o usuário a janela *Properties* onde ajustamos as propriedades dos elementos visuais arrastados para a página.

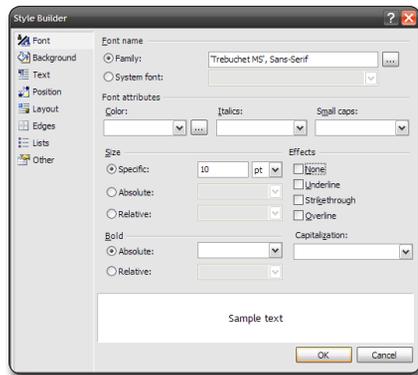


**Figura 6.** Exibindo a página em modo design no Visual Studio

Na janela *Properties* selecione o elemento *DOCUMENT* para definirmos o título da página. Clique no campo *Title* e escreva um título para sua página.

O próximo passo é criarmos uma formatação básica da nossa

página para definir por exemplo um tipo de fonte padrão. Novamente na janela *Properties* com o elemento *DOCUMENT* selecionado, clique no campo *Style* no botão “...” para exibir a janela *Style Builder*, como na **Figura 7**, defina o seu estilo e clique em OK.



**Figura 7.** Definindo estilos

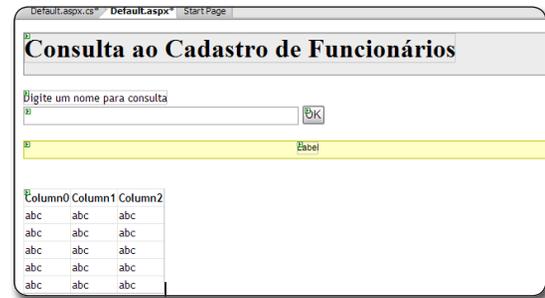
Nesta janela é possível definir vários elementos da formatação além da fonte como pode ser observado na figura anterior.

Vamos criar os primeiros elementos da nossa página. Para isso, arraste alguns componentes da *Toolbox* diretamente para a página e ajuste as propriedades como definido na **Tabela 2**.

Componente	Propriedade	Valor
Panel	ID	Panel1
	BackColor	#EAEAEA
	BorderColor	Gray
	BorderStyle	Solid
	BorderWidth	1px
	Height	50px
	Width	100%
Label	ID	Label1
	Font	Times New Roman, 24pt
	Text	Consulta ao Cadastro de Funcionários
Label	ID	Label2
	Text	Digite um nome para a consulta
TextBox	ID	txtConsulta
	MaxLength	50
	Columns	50
Button	ID	btnConsulta
	Text	OK
Panel	ID	pnAviso
	BackColor	#FFFFFF
	BorderColor	#C0C000
	BorderStyle	Solid
	BorderWidth	1px
	Font	Arial, 8pt
	Height	20px
	Width	100%
Label	ID	lblAviso
GridView	ID	GridView1
	AutoGenerateColumns	False

**Tabela 2.** Elementos da página

Com estes componentes arrastados para a página, ela deverá ter uma aparência parecida com a da **Figura 8**. Você pode ter que ajustar os elementos da página para que fiquem como a figura.



**Figura 8.** Visão da página em tempo de Design

Vejamos a função de alguns elementos começando pelo *Panel* que tem a função de agrupar outros componentes. Podemos ocultar este painel quando necessário ou ajustar sua aparência conforme o que desejarmos ter em nossa página. Os componentes arrastados para dentro deste componente ficam vinculados com este como podemos ver pelo código *ASP.NET* da **Listagem 3**.

**Listagem 3** – Demonstrando o código ASP para o componente *Panel* e componentes “filhos”

```

12 <asp:Panel ID="Panel1" runat="server"
    BackColor="#EAEAEA" Height="50px" Width="100%"
13     BorderColor="Gray" BorderStyle="Solid"
    BorderWidth="1px">
14     <asp:Label ID="Label1" runat="server" Font-
    Bold="True" Font-Names="Times New Roman"
15     Font-Size="24pt" Text="Consulta ao
    Cadastro de Funcionários"></asp:Label></asp:Panel>

```

O componente *TextBox* será usado para que digitemos o nome do funcionário a ser pesquisado pela nossa aplicação. Além de um *ID* estamos configurando também as propriedades *MaxLength* e *Columns*. A primeira propriedade deve ser configurada sempre que quisermos limitar o tamanho do texto que pode ser digitado. Este é um ponto que iremos tratar mais adiante, quando tratarmos de segurança da aplicação. O campo *Columns* vai afetar diretamente a aparência do controle.

Arrastamos também um componente *Button* que irá enviar a nossa consulta para o servidor executá-la no banco de dados e retornar os dados que encontrar.

O *pnAviso* é usado para exibirmos mensagens ao usuário, como por exemplo, quando os dados não foram encontrados, ou então, ocorreu algum erro na operação. A propriedade *Visible* é mantida como *False* sendo que o painel será exibido somente quando for necessário. Note que como definimos o formato da fonte e alinhamento do texto no próprio *Panel*, não precisamos fazer isto novamente ao arrastarmos o componente *Label* (*lblAviso*) dentro deste, sendo as propriedades de formatação herdadas deste.

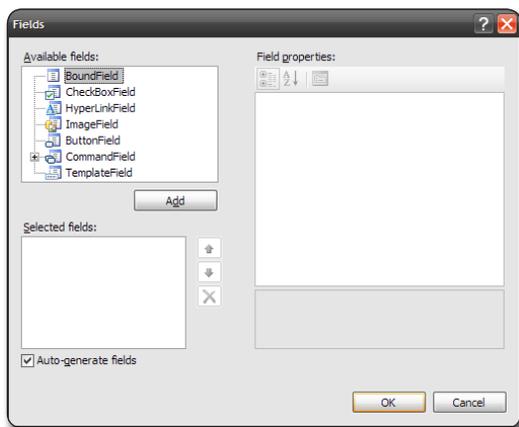
Por fim temos o componente *GridView* que é um dos principais componentes para visualização e edição de dados do *ASP.NET*.

Este componente é muito poderoso em recursos permitindo que mostremos os dados com pouco código ou ainda editar os dados exibidos, executemos alterações e exclusões e ainda personalizamos a exibição.

O componente *GridView* é muito flexível com relação a formatações e tratamento de eventos. No nosso exemplo vamos nos deter em poucos pontos principais deixando detalhes como resposta a eventos e personalização da aparência para outra oportunidade.

Quando trabalhamos com dados, podemos simplesmente deixar que o *ASP.NET* configure a exibição dos campos da tabela sem preocuparmo-nos com configurá-los no componente, porém, se quisermos uma aparência mais consistente, tratando formatos como números e datas precisamos configurar os campos e formatos.

Para fazer isto selecione o *GridView*, pressione F4 para visualizar o editor de propriedades (*Properties*) e clique no botão do campo *Columns*. Deve ser exibido uma janela como na **Figura 9**.



**Figura 9.** Editor de colunas do componente *GridView*

Nesta janela temos os campos:

- **Available Fields:** que reúne os tipos de campos (conteúdo) que podemos ter no componente sendo que o mais comum é o *BoundField* que representa os dados vindos diretamente do banco de dados. O campo *CheckBoxField* disponibiliza um controle do tipo *CheckBox* na *grid*; o *HyperLinkField* permite que tenhamos um *Hyperlink* em que o seu endereço seja o que está armazenado em uma tabela do banco de dados; *ImageField* é usado para exibirmos imagens; *ButtonField* mostra um botão que irá executar alguma ação no nosso *Form* e é muito semelhante a *CommandField* sendo este último usado para edição dos dados; por fim temos o componente *TemplateField* que permite que tenhamos componentes personalizados na nossa grade de registros.
- **Selected Fields:** quando escolhemos um componente e clicamos no botão *Add* as colunas que já foram definidas são listadas nesta caixa e suas propriedades devem ser

editadas na caixa de listagem *Field Properties*

- **Auto-generate Fields:** é usado para definir se os campos serão configurados dinamicamente pelo *ASP.NET* sendo exibidos conforme o resultado da consulta ou se devem seguir as definições que formos fazendo nesta janela. No nosso exemplo, desmarque este campo.

Para nossa aplicação vamos incluir todos os campos da tabela *Funcionarios* do *Access*. Para isso, selecione *BoundField* em *Available Fields*, clique no botão *Add* e configure as propriedades conforme definido na **Tabela 3**.

Bound Field	Propriedade	Valor
ID	HeaderText	ID
	DataField	ID
Nome	HeaderText	Nome
	DataField	Nome
Cargo	HeaderText	Cargo
	DataField	Cargo
Área	HeaderText	Área
	DataField	Área
Nascimento	HeaderText	Nascimento
	DataField	Nascimento
	HtmlEncode	False
	DataFormatString	{0:dd/MM/yyyy}
Admissão	HeaderText	Admissão
	HtmlEncode	False
	DataField	Admissao
	DataFormatString	{0:dd/MM/yyyy}
Salário	HeaderText	Salário
	HtmlEncode	False
	DataField	Salario
	DataFormatString	{0:n2}

**Tabela 3.** Definição dos campos do componente *GridView*

A propriedade *HeaderText* configura qual o cabeçalho que deve ser usado para a coluna enquanto *DataField* faz referência ao nome do campo na tabela.

*HtmlEncode* indica para o *ASP.NET* se o conteúdo do campo deve ser transformado em código HTML por razões de segurança convertendo caracteres especiais como "<>;&" e acentuações nos respectivos códigos HTML. Deixe como *False* para podermos definir os formatos de data e numérico para os campos *Nascimento* e *Admissão* (data) e *Salário* (numérico). Se deixarmos *HtmlEncode* como *True* o formato será desprezado.

Vamos formatar a *GridView* usando um dos formatos disponibilizados pelo *Visual Studio*. Para isto vamos usar um outro recurso interessante do *Visual Studio* que é o *Task Editor*, uma janela de diálogo para cada componente com as principais tarefas (*tasks*) que podem ser executadas para cada componente. Selecione a *GridView* e note a seta no canto superior direito. Clique nesta e veja quais as ações que são aplicáveis para este componente como na **Figura 10**.

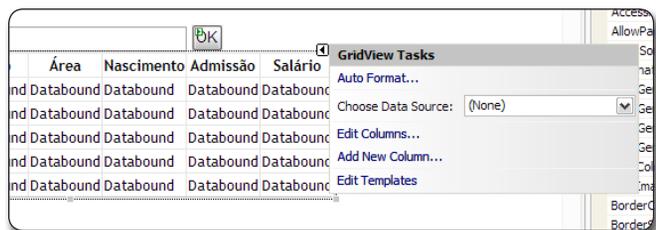


Figura 10. Editor de tarefas

Para formatarmos nossa *GridView* usando os formatos do *Visual Studio* escolha *Auto Format...* e escolha um estilo que seja da sua preferência. Desta forma, estamos com nossa página definida quanto a seu *Design*. O próximo passo é adicionar mais funcionalidade para a mesma, ou seja, fazê-la responder a eventos como o clique no botão para executarmos a consulta.

### Escrevendo código respondendo a eventos

Neste exemplo, vamos escrever o código que executa a consulta ao banco de dados *Access*. Para escrever um código respondendo a um evento como um clique em um botão, dê um duplo clique sobre este, e escreva o código como na **Listagem 4**. No começo do arquivo, escreva também *using System.Data* para importar este namespace para a página.

No código exposto encontramos alguns pontos a serem considerados. Primeiramente, além de cuidar da execução da consulta propriamente dita e envio dos dados para a página, é feito o tratamento de erros que a operação pode gerar, prevenindo de que a página exiba mensagens de erro do servidor para o *browser*.

A partir da linha 25 é ocultado o painel que exibe eventuais mensagens de erros anteriores (*pnAviso*). Da mesma forma, nas próximas duas linhas, estamos limpando os dados de uma eventual consulta anterior. O método *DataBind()* é usado para popular o componente *GridView*. Como anteriormente definimos que sua propriedade *DataSource* – responsável por prover os dados para o componente – como nula (*null*), nenhum dado será exibido no componente.

Como estamos usando C# e o *Framework .NET*, iremos usar as classes do *ADO.NET* aqui representadas pelo namespace *System.Data.OleDb* para conectarmos com a nossa base de dados *Access*.

#### Nota 3. JSR 172 e KSoap

Lembrando que como a aplicação está baseada no .NET podemos usar a maior parte das classes do *Framework* e, embora não estejamos usando no nosso exemplo, podemos nos conectar com diversos tipos de bancos de dados como por exemplo *Microsoft Sql Server*, *Oracle*, *Firebird*, *Postgree* entre outros.

A classe *OleDbConnection* é quem realiza a conexão com o banco de dados. Precisamos inicializá-la passando os parâmetros como o nome do arquivo e a localização destes e o tipo do *Provider* a ser usado. Esta configuração está em *hard code* no nosso exemplo (escrita diretamente no código-fonte

#### Listagem 4. Código para executar a consulta

```

21 protected void btnConsulta_Click(object sender, EventArgs e)w
22 {
23     // Executa a consulta ao banco de dados
24     // Oculta o painel de aviso
25     pnAviso.Visible = false;
26     // Limpa dados de uma consulta anterior
27     GridView1.DataSource = null;
28     GridView1.DataBind();
29     // Classe que realiza a conexão
30     System.Data.OleDb.OleDbConnection conexao =
31         new System.Data.OleDb.OleDbConnection
32         (@"Provider=Microsoft.Jet.OLEDB.4.0;Data
33         Source=d:\documentos\xopt.mdb");
34     // Classe para a consulta
35     System.Data.OleDb.OleDbDataAdapter daConsulta =
36         new System.Data.OleDb.OleDbDataAdapter(@"
37         select *
38         from funcionarios
39         where nome like ?
40         order by nome", conexao);
41     // Configuração do parâmetro de consulta
42     daConsulta.SelectCommand.Parameters.Add("@nome",
43         System.Data.OleDb.OleDbDbType.VarChar).Value = "%" +
44         txtConsulta.Text + "%";
45     DataTable dtConsulta = new DataTable
46         ("funcionario");
47
48     try
49     {
50         // abre a conexão com o banco de dados
51         conexao.Open();
52         // Executa a consulta preenchendo a tabela
53         // com os dados encontrados
54         daConsulta.Fill(dtConsulta);
55         // Envia dados encontrados para a
56         // GridView, caso tenha encontrado
57         if (dtConsulta.Rows.Count > 0)
58         {
59             GridView1.DataSource = dtConsulta;
60             GridView1.DataBind();
61         }
62         else
63         {
64             // Não encontrou os dados, avisa o
65             // usuário
66             lblAviso.Text = "Nenhum registro
67             encontrado.";
68             pnAviso.Visible = true;
69         }
70     }
71     catch (Exception ex)
72     {
73         // Se ocorrer algum erro, faz o tratamento
74         // e envia para o usuário
75         lblAviso.Text = "Ocorreu um erro durante a
76         consulta.\n" + ex.Message;
77         pnAviso.Visible = true;
78     }
79     finally
80     {
81         // Fecha a conexão com o banco
82         // Executa mesmo que ocorra erro
83         conexao.Close();
84     }
85 }

```

da aplicação), mas uma maneira mais correta de configurar é armazenar as informações sobre a conexão no arquivo *Web.Config* da aplicação.

A *string* de conexão para cada tipo de banco de dados irá variar. Para tirar dúvidas sobre qual o formato correto e para tipos diferentes de *string* de conexão (inclusive com arquivos texto ou do *Microsoft Excel*) visite o site <http://www.connections-trings.com>.

Para definirmos a instrução SQL que irá consultar os dados, usamos a classe *OleDbDataAdapter* que é usada quando desejamos preencher um *DataTable* com os dados trazidos da consulta. Uma outra maneira seria usarmos um *OleDbCommand* e um *DataReader*. Oportunamente poderemos tratar das vantagens de um e de outro. No nosso exemplo

vamos usar a *DataTable* para podermos verificar mais facilmente se a consulta obteve algum resultado ou não mais facilmente.

Como estamos consultando dados que foram passados por um usuário, é necessário enviarmos também os parâmetros de consulta. A nossa instrução SQL deve estar preparada para receber os parâmetros. Fazemos isto colocando o símbolo de interrogação “?” para cada parâmetro esperado e em seguida, enviando os parâmetros na ordem em que aparecem no SQL através do método *Parameters.Add()*.

A sintaxe para o envio dos parâmetros exibida aqui é típica para bancos de dados *Access* tendo algumas variações dependendo do tipo do banco de dados. Para uma consulta a um banco *Microsoft SQL Server* por exemplo, usamos o nome do parâmetro (“@<nome\_do\_parametro”) na instrução SQL.

Como estamos usando a palavra-chave *like* em nossa instrução SQL precisamos configurar os parâmetros colocando o símbolo “%” no início e no fim da expressão de consulta para que possamos comparar o que o usuário digitou em qualquer parte do nome do funcionário. Assim se for digitado “José” será listado “José Antonio”, “José da Silva”, “Antonio José” e etc.

Mais adiante, quando tratarmos de segurança iremos retornar ao assunto sobre passar parâmetros para consultas para bancos de dados e montagem da instrução SQL.

Em seguida estamos executando a consulta em um bloco protegido representado pelo *try...catch...finally*. Com isso, os erros que ocorrerem durante esta consulta poderão ser tratados e exibidos de uma maneira melhor do que enviarmos uma mensagem do servidor ou ainda, fazendo com que a aplicação trave.

O primeiro passo é abrir a conexão com o banco de dados através do método *Open()* do nosso objeto de conexão. É importante observar que isto consome recursos do servidor, logo, é preciso dimensionar apropriadamente o tamanho da consulta prevendo inclusive um número potencial de conexões que podem ser usadas simultaneamente. Além disso, após executar a consulta e obter os dados, esta conexão deve ser fechada com o método *Close()* do objeto de conexão.

Através do método *Fill* do *OleDbDataAdapter* preenchemos a *DataTable* com os dados que foram consultados. O *DataTable* possui um recurso para sabermos se a consulta retornou algum registro. Isto é feito através de *Rows.Count*. Desta forma podemos avisar ao usuário caso nenhuma informação tenha sido encontrada.

Se a pesquisa retornar algum registro, passamos o *DataTable* como *DataSource* (origem dos dados) do componente *GridView* e o populamos através do método *DataBind()*.

Caso nenhum registro seja retornado, tornamos o componente *pnAviso* visível e colocamos uma mensagem para o usuário.

Por fim, fazemos o tratamento de erros com o bloco *Catch*. Se ocorrerem exibimos o painel de aviso para o usuário evitando que mensagens do servidor sejam enviadas para

o mesmo e ao final de tudo fechamos a conexão com o banco de dados no bloco *finally* que será sempre executado independentemente de ocorrer erros ou não na operação.

Para executar a página pressione a tecla F5. O resultado deve ser parecido com o das **Figuras 11 e 12**.



**Figura 11.** Aplicação ASP.NET sendo executada



**Figura 12.** Nenhum dado encontrado para a pesquisa

## Considerações sobre segurança

A Internet é um ambiente totalmente livre e aberta a todo o tipo de usuário quer seja bem intencionado ou não. Por mais que os *browsers* aperfeiçoem-se em segurança e controle de fraudes sempre vai ser possível encontrar brechas nas aplicações executadas por estes uma vez que não existe aplicação 100% à prova de falhas – mesmo que se tente chegar perto disto.

Assim o desenvolvedor deve cuidar de alguns aspectos para que o mínimo de segurança seja oferecido ao usuário. Os primeiros cuidados a serem tomados são validar as entradas feitas pelo usuário nos campos de formulário. Através de um componente *TextBox* por exemplo, usuários mal intencionados podem acessar detalhes do banco de dados usando uma forma de ataque conhecida como *Sql Injection*; escrever um código de *script* mal intencionado ou simplesmente desconfigurar a página através de código HTML injetado.

Estes são os tipos mais comuns de ataques. Uma página *ASP.NET* já possui um controle destes tipos de ataque definido por default. Faça o seguinte teste demonstrado na seqüência das **Figuras 13 e 14**.



Figura 13. Exemplo de digitação de código HTML em página ASP.NET

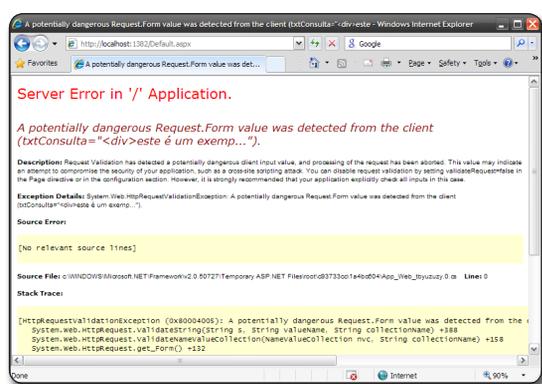


Figura 14. Demonstração do resultado do código digitado

O resultado do código digitado é o travamento da aplicação, pois o *ASP.NET* já faz uma validação dos dados digitados e detecta potenciais ataques. O efeito indesejado é que além do travamento da aplicação, parte do código é exibida para o usuário, o que, embora seja aparentemente inútil pode revelar detalhes da aplicação para potenciais invasores.

Uma forma de evitar que esta tela seja exibida é alterar o código HTML da página para que não faça nenhuma validação padrão do que foi digitado pelo usuário.

Isto é feito clicando-se no *Visual Studio* exibindo-se o código HTML clicando-se no botão *Source* do *design* da página. Logo na primeira linha temos a diretiva “<@ page” onde várias opções são configuradas automaticamente, no nosso caso, inclua a instrução `ValidateRequest="false"` assim, linha deve ficar como indicado a seguir:

```
1 <%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ex_aspnet._
Default" ValidateRequest="false" %>
```

Agora faça novamente o teste anterior e observe que nenhum erro acontece. Porém, isto tem um problema. Como estamos enviando a consulta para o banco de dados usando parâmetros em vez de escrever o que foi digitado na página, não há potencialmente nenhum perigo, mas, se por algum motivo quisermos exibir o que foi digitado podemos ter problemas.

Vamos demonstrar isto. Logicamente não é algo muito comum que se faça mas serve para demonstrar potenciais perigos. Altere o código que responde ao evento do botão *OK* da seguinte forma como aparece na **Listagem 5**.

Listagem 5. Código alterado para demonstrar o que o usuário digitou

```
41 try
42     {
43         // abre a conexão com o banco de dados
44         conexao.Open();
45         // Executa a consulta preenchendo a tabela
46         // com os dados encontrados
47         daConsulta.Fill(dtConsulta);
48         // Envia dados encontrados para a
49         // GridView, caso tenha encontrado
50         // Mostra na label de aviso o conteúdo do
51         // que o usuário digitou.
52         lblAviso.Text = "Resultado da pesquisa do
53         texto: " + txtConsulta.Text;
54         pnAviso.Visible = true;
55
56         if (dtConsulta.Rows.Count > 0)
57         {
58             GridView1.DataSource = dtConsulta;
59             GridView1.DataBind();
60         }
61         else
62         {
63             // Não encontrou os dados, avisa o usuário
64             lblAviso.Text += "Nenhum registro
65             encontrado.";
66             pnAviso.Visible = true;
67         }
68     }
69     catch (Exception ex)
70     {
71     }
```

As linhas 49 e 50 foram adicionadas para que seja escrito na página o conteúdo da consulta. A linha 60 foi alterada sendo inserido um “+=” para que a mensagem de que o registro não foi encontrado seja também exibida, ao lado do conteúdo digitado pelo usuário.

Digite o código acima, execute o programa pressionando F5 e no campo texto, digite o seguinte:

```
<script>alert('Olá! Estou inserindo um script.')</script>
```

O texto anterior se trata de um código em *JavaScript* e configura uma modalidade de ataque conhecida como *Script Attack*. Pressione o botão *OK* e provavelmente você terá o resultado da **Figura 15**.

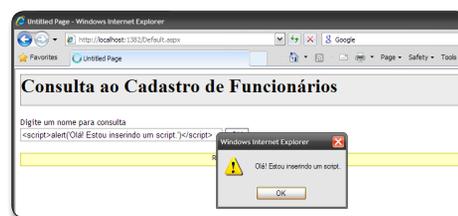


Figura 15. Exemplo de Script Attack

O exemplo é realmente simples e um tanto quanto ingênuo, mas, considere o potencial da linguagem *JavaScript* e então pense nas possibilidades do que pode ser feito.

Agora que conhecemos estes tipos de ataque vamos ver como defender nossa aplicação.

Em primeiro lugar nunca desative a validação dos dados digitados com `ValidateRequest="False"` como fizemos. Porém, quando for necessário, isto por algum motivo, como por exemplo armazenar em banco de dados algum código que tenha sido digitado como em uma entrada de *blog* por exemplo, antes de enviar o conteúdo para o banco converta qualquer símbolo de linguagem de marcação para seus códigos HTML de maneira segura.

Isto é feito usando o método `Server.HtmlEncode()` como demonstrado na **Listagem 6**.

#### Listagem 6. Codificando os dados

```
37 // Configuração do parâmetro de consulta
38 daConsulta.SelectCommand.Parameters.Add("@
    nome", System.Data.OleDb.OleDbType.VarChar).
    Value = "%" + Server.
    HtmlEncode(txtConsulta.Text) + "%";
39 DataTable dtConsulta = new DataTable
    ("funcionario");
40
41 try
42 {
43     // abre a conexão com o banco de dados
44     conexao.Open();
45     // Executa a consulta preenchendo a tabela
46     com os dados encontrados
47     daConsulta.Fill(dtConsulta);
48     // Envia dados encontrados para a GridView,
49     caso tenha encontrado
50     // Mostra na label de aviso o conteúdo do
51     que o usuário digitou.
52     lblAviso.Text = "Resultado da pesquisa do
53     texto: " + Server.
54     HtmlEncode(txtConsulta.Text);
55     pnlAviso.Visible = true;
```

Observe que na linha 38 e 49 incluímos a chamada para a função citada. Agora, observe o resultado disto na **Figura 16**.



**Figura 16.** Demonstrando resultado da codificação da entrada do usuário

O segundo passo é limitar o tamanho do texto que pode ser digitado. Além de restringir as possibilidades de escrever, por exemplo, um script complexo ou ainda evita provocar um estouro de *buffer*. Suponha por exemplo uma tela para entrada de dados que são gravados em um banco de dados. Se tivermos um campo com tamanho limitado na tabela, mas no lado do *browser* não seja feito nenhum tratamento para limitar o tamanho dos dados que pode-se digitar. Isto é feito através da propriedade `MaxLength` que deve receber um número que corresponda ao tamanho do campo na tabela do banco de dados por exemplo.

Finalmente ao construir suas instruções SQL evite usar concatenação de string como forma de passar os parâmetros para o banco. A **Listagem 7** abaixo demonstra como isto seria feito no caso da nossa aplicação.

#### Listagem 7. Montando a instrução SQL concatenando string

```
31 // Classe para a consulta
32 System.Data.OleDb.OleDbDataAdapter daConsulta =
    new System.Data.OleDb.OleDbDataAdapter(@"
33     select *
34     from funcionarios
35     where nome like '%" + txtConsulta.Text + @"%'
36     order by nome", conexao);
```

Com o código anterior – concatenando-se a instrução SQL com o que foi digitado pelo usuário, abre-se uma brecha para que o usuário mal intencionado construa uma instrução SQL para, por exemplo, obter dados do banco como nome de tabelas, de campos etc.

Ao usarmos parâmetros (como está demonstrado na **Listagem 4**), isto fica bem difícil de acontecer. Outra medida é configurar os componentes que serão populados com os dados do banco apropriadamente, ou seja, definindo nomes dos campos para que somente sejam exibidos os dados esperados.

No caso do nosso componente *GridView* nós fizemos isto nos primeiros passos e marcando a propriedade `AutoGenerateColumns` com `false`.

Neste caso, mesmo que dados diferentes sejam trazidos pela consulta, somente os campos configurados é que serão exibidos.



## Resumo do DevMan

Os aspectos de segurança com as páginas ASP.NET vão muito além dos expostos aqui e precisam ser muito bem estudados, busque informar-se sobre o assunto para que possa desenvolver aplicações Web seguras e confiáveis.

Além das medidas de segurança que devem ser implementadas na aplicação WEB deve-se constantemente verificar por brechas de segurança nos browsers mais comuns e como sua aplicação pode ser afetada.

Uma boa fonte de consulta às brechas em aplicativos e de sistema operacional é o site do Secunia (<http://secunia.com>). Além de relatar as vulnerabilidades encontradas em diversos softwares e sistemas operacionais de várias plataformas dispõe de ferramentas que verificam as aplicações no pc do usuário.

Outra fonte de informação é o fórum de segurança em <http://forums.asp.net/25.aspx>, do site oficial do projeto ASP.NET.

## Publicando a página no IIS

Por fim, um aspecto importante na criação das páginas ASP.NET é sua configuração em um servidor WEB como no nosso caso o IIS. Com o *Visual Studio 2005* isto é bastante facilitado. Nas figuras a seguir o processo é demonstrado.

O primeiro passo é copiar o conteúdo da nossa aplicação para o local onde será disponibilizado para o IIS gerenciar o acesso dos usuários. Com o botão direito do mouse clique na aba *Solution Explorer* que irá exibir os arquivos. Clique no ícone correspondente ao projeto com o botão direito do mouse e clique em *Publish* no menu que é exibido como na **Figura 17**.

Ao fazer isto é aberta uma janela (**Figura 18**) para que seja digitado o local no computador atual ou remoto onde a aplicação será armazenada. No nosso exemplo, vamos usar a pasta usada como padrão pelo IIS que é `c:\inetpub\wwwroot`. Ainda estamos apenas copiando os arquivos necessários observando que somente os que são necessários não sendo incluídos os arquivos correspondentes ao código em C# uma vez que a aplicação é enviada compilada.

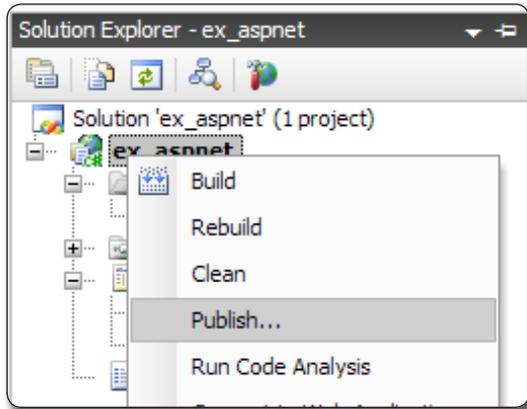


Figura 17. Enviando a aplicação para a pasta onde será armazenada

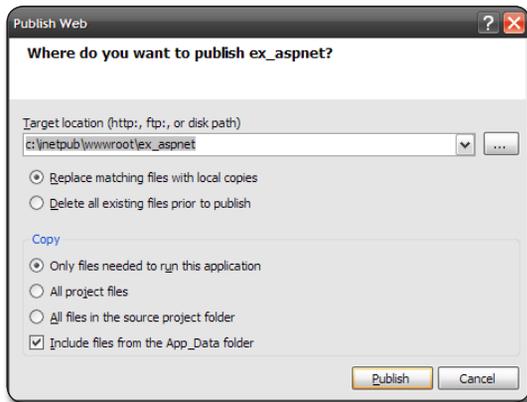


Figura 18. Janela para cópia dos arquivos

A Figura 19 mostra no Windows Explorer o conteúdo da pasta da aplicação no sistema de arquivos do servidor ou do computador.

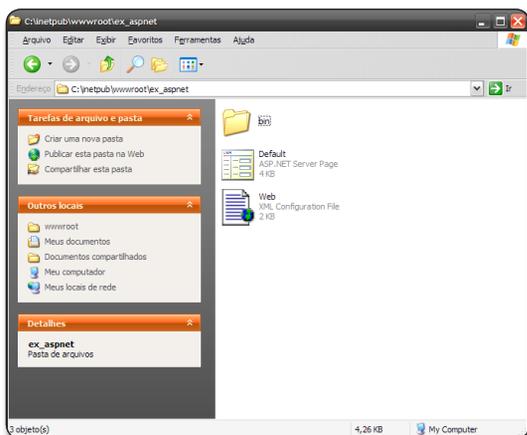


Figura 19. Exibindo os arquivos da aplicação

Após a cópia dos arquivos precisamos criar e configurar a aplicação no IIS. Abra o *Painel de Controle*, em seguida, *Ferramentas Administrativas* e por fim, o ícone *Internet Information Services* (IIS). A Figura 20 mostra a tela principal deste aplicativo.

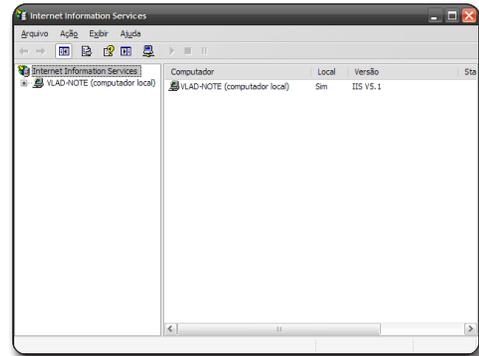


Figura 20. Tela Principal do IIS

Para configurar sua aplicação expanda os nós da árvore “(computador local)” até encontrar o ícone *Site da Web Padrão*. Clique neste com o botão direito do mouse e escolha a opção *Novo -> Diretório Virtual* como na Figura 21.

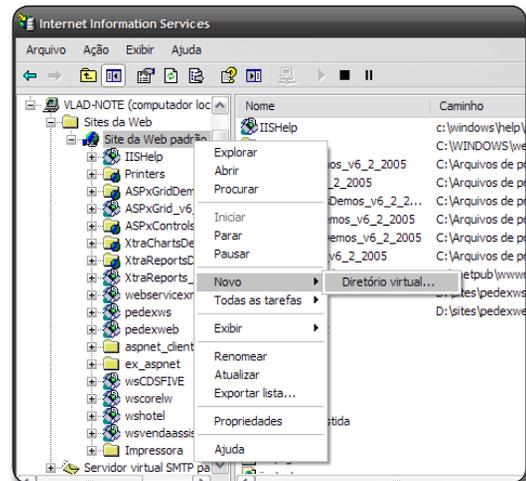


Figura 21. Criando o diretório virtual para a aplicação

Este processo irá indicar para o IIS onde estão localizados os arquivos da aplicação WEB e o nome pelo qual a aplicação será invocada no campo *Endereço*: dos browsers. As Figuras 22 a 26 mostram os passos desta configuração.

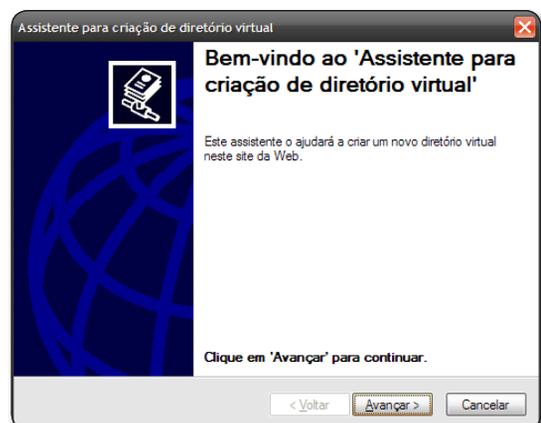


Figura 22. Início do assistente para configuração da pasta

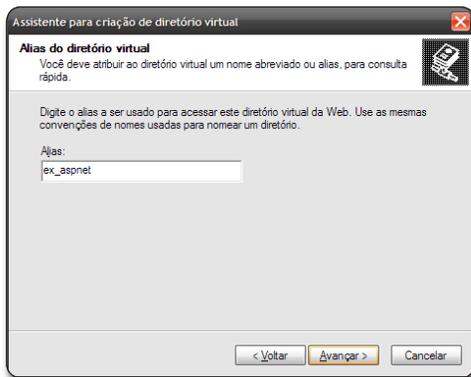


Figura 23. Definindo o nome para a aplicação

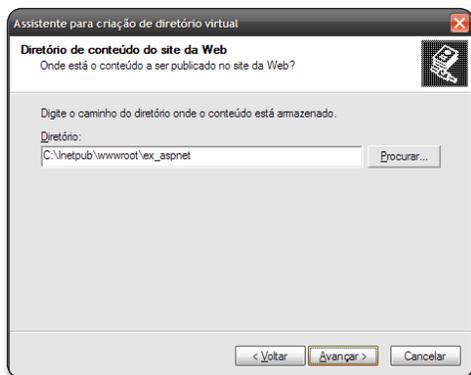


Figura 24. Indicando o diretório onde está localizada a aplicação

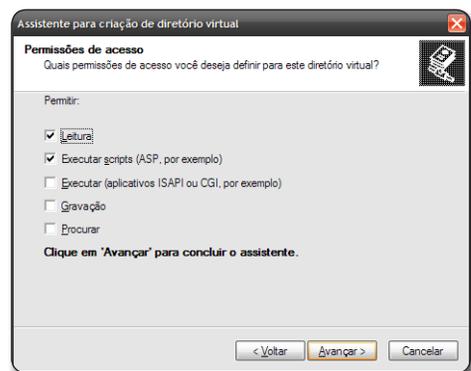


Figura 25. Definindo permissões

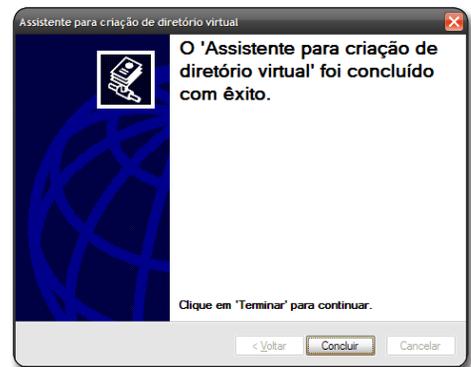


Figura 26. Conclusão

Com os passos expostos aqui a aplicação está quase pronta para ser acessada por outros computadores, observe antes se a versão do *ASP.NET* está configurada corretamente. Isto é necessário caso o servidor tenha instalado outras versões do *Framework .NET*. Clique no ícone correspondente ao nome da aplicação recém criada com o botão direito do mouse e escolha *propriedades*. Na janela que se abre (Figura 27), escolha a aba *ASP.NET* e verifique se o campo *ASP.NET Version* está correto (2.0.50727).

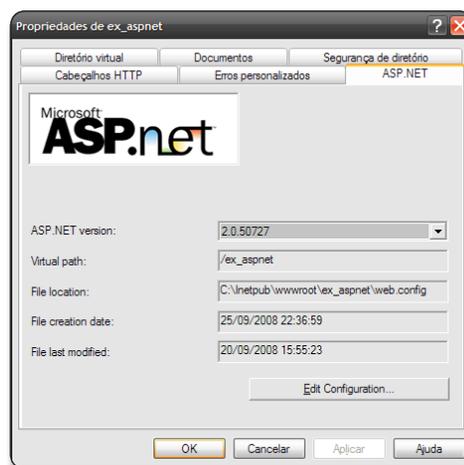


Figura 27. Verificando a versão correta do ASP.NET

Teste a execução da aplicação a partir do IIS. Abra um browser e digite `http://localhost/ex_aspnet`.

#### Nota

Usamos `http://localhost` por estarmos acessando localmente, se formos fazer isto a partir de uma estação da rede devemos inserir o nome do servidor ou o endereço IP. Lembrando que se houver um firewall este deve permitir o acesso à porta http.

## Conclusão

*ASP.NET* e o desenvolvimento de aplicações WEB com o *Visual Studio 2005* torna-se uma tarefa produtiva já que com o exposto aqui pudemos conhecer as facilidades oferecidas por esta ferramenta.

Procure aprofundar seus conhecimentos buscando fontes de dados como os sites <http://www.asp.net> - site oficial da plataforma e <http://www.w3schools.com/aspnet/default.asp>, tutorial desenvolvido pelo W3C para início. Existem várias comunidades na Internet sobre este assunto. ●

#### Dê seu feedback sobre esta edição!

A *.NET Magazine* tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

[www.devmedia.com.br/netmagazine/feedback](http://www.devmedia.com.br/netmagazine/feedback)

