# Autenticação e Permissões de usuários em ASP.NET MVC 4

- Postado por Renan Cunha em 29 dezembro 2011 às 22:30
- Exibir blog

Neste artigo iremos ver uma maneira eficiente de autenticar usuários e definir permissões para os mesmos, ou seja, quais locais de uma aplicação (em ASP.NET MVC 4) o usuário poderá acessar. A autenticação e as permissões serão feitas a partir de informações contida em um banco de dados. Para desenvolver o exemplo do artigo estarei utilizando o Visual Web Developer 2010 (Visual Studio 2010), um banco de dados (SQL Server 2008) e o Entity Framework 4.1.

### Importante

Porém o foco do artigo não será ensinar o básico do MVC e nem a integração da aplicação com o banco de dados. O foco será totalmente voltado para a questão da autenticação. É interessante que o leitor já tenha um conhecimento prévio sobre o framework ASP.NET MVC para que não fique perdido.

Eu irei disponibilizar o download de uma aplicação básica, e nós iremos implementar nela um controle de login e depois definir os locais que um dado usuário poderá acessar.A estrutura dessa aplicação que iremos tomar como ponto de partida é a seguinte:

*(O download da aplicação que usaremos está no final do artigo)*

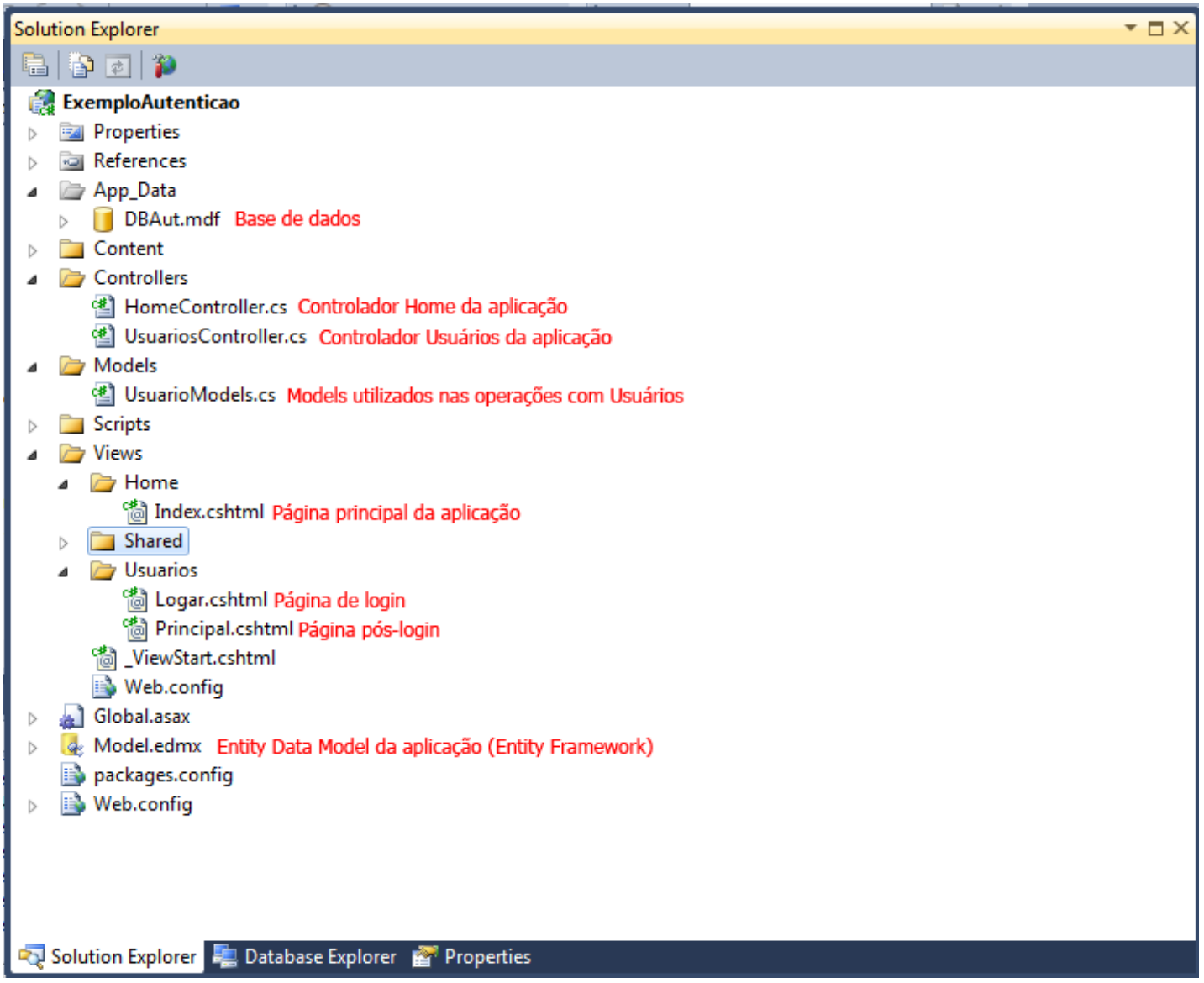



*Imagem 1 - Estrutura básica da aplicação*

Bom, primeiro é importante entendermos a lógica utilizada para modelar o banco de dados utilizado.

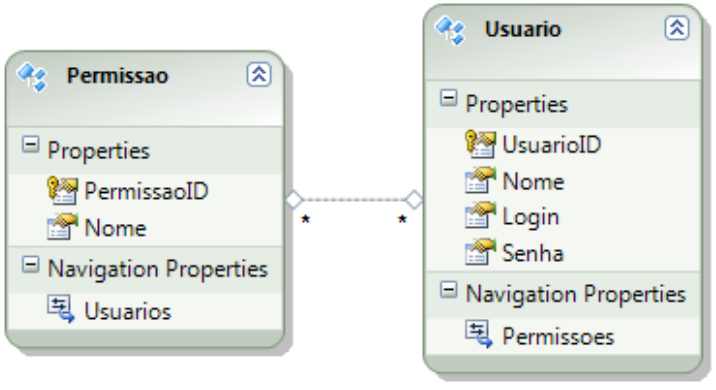


*Imagem 2 - Estrutura do banco de dados*

Podemos ver que a relação entre Usuários e Permissões é uma relação de muitos para muitos, pois um usuário pode ter várias permissões, e uma permissão pode pertencer a vários usuários. O Entity Framework já mapeou essa relação entre as entidades, como pode ser visto ali, o Usuario possui uma lista de Permissoes e vice-versa.

## Autenticação

Para iniciarmos, crie um novo diretório no nosso projeto, chamado Repositories. É nele que iremos guardar os repositórios da aplicação, que nada mais são do que classes que irão manipular a parte de dados da aplicação. Criado o diretório, crie dentro dele uma classe chamada UsuarioRepositorio, que será o repositório de Usuários, é nele que iremos implementar funcionalidades, como o método que irá tentar autenticar uma pessoa entre outros. (Que fique claro aqui que essa maneira como estou criando os diretórios, trabalhar com repositório, é uma opinião pessoal, cada um desenvolve da maneira que achar mais organizada, se você preferir organizar a estrutura de outra maneira não tem problema algum. Feito os passos acimas deveremos ter algo como a imagem abaixo:

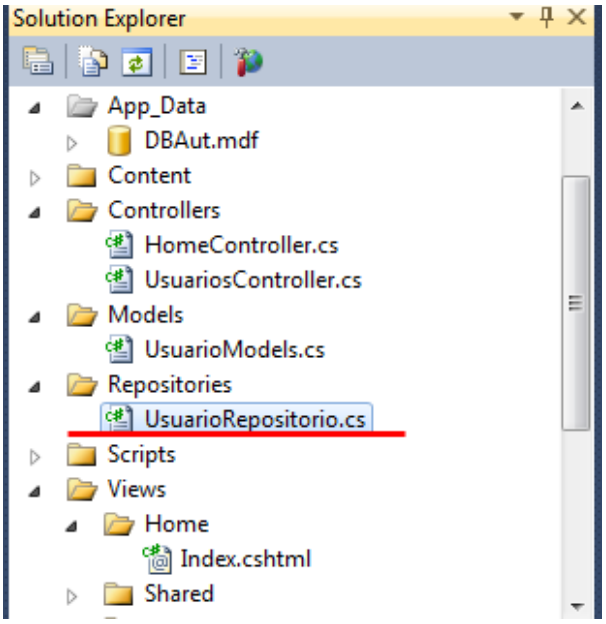



*Imagem 3 - Novo repositório*

Agora abra a nova classe criada "**UsuarioRepositorio**", e vamos implementar alguns métodos e propriedades nela. Primeiramente iremos criar um método estático chamado AutenticarUsuario que receberá como parâmetro duas strings, uma é o Login e a outra é a Senha. O resultado final será esse:

```
public static bool AutenticarUsuario(string Login, string Senha)
{

    DBAutEntities entities = new DBAutEntities();

    var Query = (from u in entities.Usuarios
                 where u.Login == Login &&
                     u.Senha == Senha
                 select u).SingleOrDefault();

    //Usuário não existe ou a senha está incorreta
    if (Query == null)
        return false;


    //Irá setar um cookie encriptado com o Login do usuário autenticado
    FormsAuthentication.SetAuthCookie(Query.Login, false);

    return true;

}
```

*Imagem 4 - Código para autenticar o usuário*

O código acima se auto-explica facilmente. Primeiro nós verificamos se os dados informados realmente existem no banco de dados e se pertencem, caso se obtenha sucesso, é setado o cookie de autenticação. O próximo passo agora é criamos um método estático que irá retornar o Usuario que está logado, caso existir um usuário logado é claro. O código final desse outro método será o seguinte:

```
public static Usuario GetUsuarioLogado()
{

    //Pegamos o login do usuário logado
    string _Login = HttpContext.Current.User.Identity.Name;

    //Não existe usuário logado, retornamos null
    if (_Login == "")
    {
        return null;
    }
    else
    {

        DBAutEntities entities = new DBAutEntities();

        //Buscamos no banco de dados o usuário que está logado
        Usuario user = (from u in entities.Usuarios
                        where u.Login == _Login
                        select u).SingleOrDefault();

        return user;

    }

}
```

*Imagem 5 - Código para buscar o usuário logado*

O código acima também é simples, ele apenas verifica se existe algum usuário logado (existência do cookie encriptado), caso exista ele faz uma busca no banco de dados e retorna um objeto Usuario.

Vamos implementar também no repositório Usuário, um método que irá desfazer a autenticação, ou seja, vai "Deslogar". Ele é bem simples:

```
public static void Deslogar()
{
    FormsAuthentication.SignOut();
}
```

A próxima etapa agora é implementar a autenticação na hora que usuário tentar logar. Se você verificar, no controlador "*UsuarioController*" já existe o ambiente preparado para exibir a página de login, e também para tratar o processamento lógico, que é quando o usuário envia o formulário preenchido. Na aplicação que estamos usando como base, já está criado o View para o usuário fazer o login. A URL dessa página de login é por convenção "*/Usuarios/Logar*", pois o nome do *controller* é Usuarios e o do método é Logar.

Abra a classe "*UsuarioController*", que se encontra no namespace Controllers da aplicação. Vá até o método "*public ActionResult Logar(UsuarioLogin_M model)*". É aqui que iremos *disparar* aquele método estático "*AutenticaUsuario*" que criamos lá no repositório do Usuario. Observe como ficará o código:

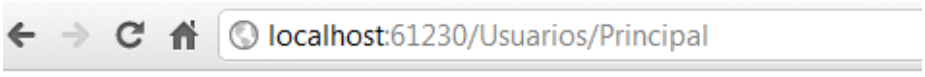
```
//Esse método abaixo será invocado quando o usuário tentar fazer login
[HttpPost]
public ActionResult Logar(UsuarioLogin_M model)
{

    if (UsuarioRepositorio.AutenticarUsuario(model.Login, model.Senha) == false)
    {
        ViewBag.msg_Error = "O nome de usuário ou a senha informada estão incorretos!";
        return View(model);
    }

    //Usuário foi autenticado com sucesso, redirecionamos ele para a página especial dos usuários.
    return RedirectToAction("Principal", "Usuarios");
}
```

*Imagem 6 - Código disparado quando o usuário tenta logar*

Se você executou todas etapas corretamente, rode a aplicação e vamos testar, para ver se o login está funcionando corretamente. Já existe um usuário de testes cadastrado, os dados para logar nele é teste/teste. Se você conseguiu efetuar o login e chegou na página abaixo, significa que a parte da autenticação está completa.




Página que só usuários logados podem ver

*Imagem 7 - Resultado da autenticação*

## Permissões

Bem, com a autenticação funcionando, vamos partir para as permissões. Se você reparar no controller "*UsuarioController*", no método que retorna o View da página que somente usuários logados podem acessar, existe uma anotação "*[Authorize]*" nele.

```
//Essa página será a pagina principal do usuário, só poderá ser acessada se o usuário estiver logado
[Authorize]
public ActionResult Principal()
{
    return View();
}
```

*Imagem 8 - Anotação Authorize*

A anotação ***Authorize*** é um recurso do modo que estamos utilizando para autenticar nossos usuários (*FormsAuthentication*). Todo método que tiver essa anotação, quando requisitado, passará por uma verificação para ver se existe um usuário logado ou não.

O problema é o seguinte, e se nós possuirmos dois tipos de usuários, por exemplo, os usuários comuns e um outro tipo, que seria os administradores. É obvio que os administradores teriam acessos a outras áreas do site. Conforme falado lá no inicio, o nosso banco de dados foi estruturado para funcionar desta maneira, com vários tipos de usuários.

A seguir iremos fazer algumas implementações para que a anotação *Authorize* receba as permissões por parâmetros. Quando o Authorize é executado, ele busca as "Roles" que seriam no nosso caso, as permissões disponíveis para o usuário. O que acontece é que como estamos fazendo isso de uma maneira alternativa, armazenando as permissões em nosso DB, precisamos sobrescrever o método que lista as "Roles" de um certo usuário.

Para iniciar, nós iremos criar um outro diretório em nossa aplicação, chamado "*Security*". Dentro desse diretório, crie uma nova classe em branco chamada "*PermissaoProvider*" e estenda essa classe a classe "*RoleProvider*" que faz parte do namespace "*System.Web.Security*". O próximo passo é selecionarmos a opção para que o Visual Studio implemente para nós todos métodos da classe abstrata "*RoleProvider*" como mostra a imagem abaixo:

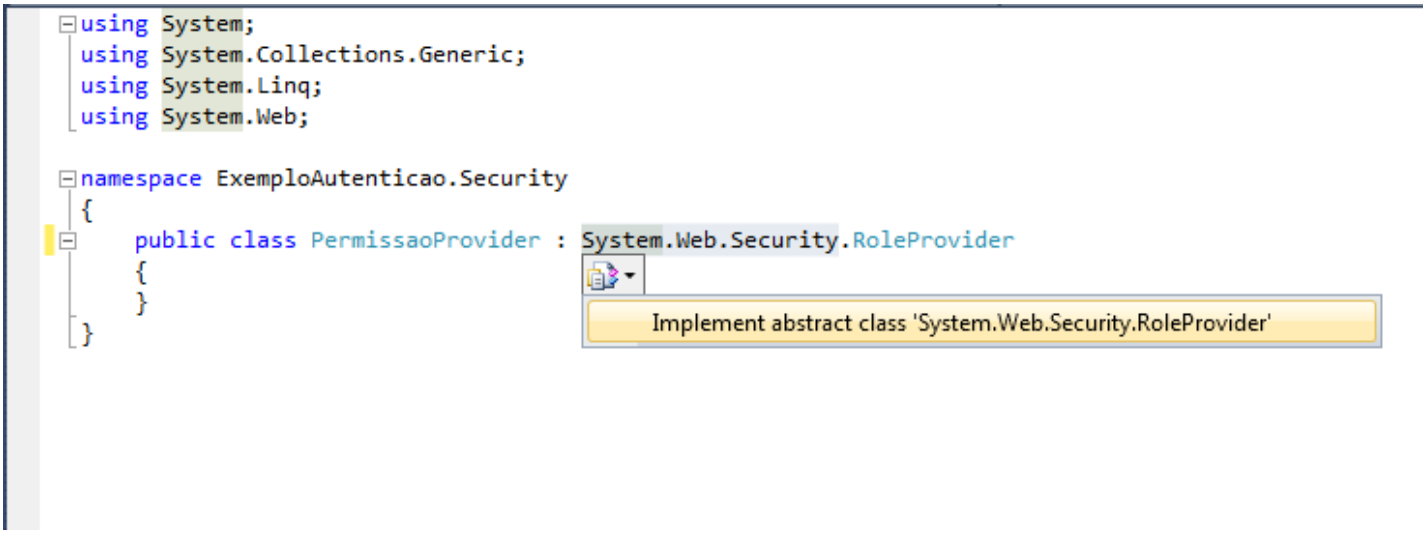



*Imagem 9 - Implementação automática dos métodos da RoleProvider*

Após clicar na opção mostrada na imagem, será criado vários métodos em nossa classe, mas apenas um nos interessa no momento. Esse que nos interessa se chama "*GetRolesForUser*". Nós iremos sobrescrever ele, esse método é o que foi falado acima, ele é responsável por retornar um array de strings que são as *Permissões* de um usuário. Veja como ficará o método:

```
public override string[] GetRolesForUser(string username)
{

    DBAutEntities entities = new DBAutEntities();


    Usuario user = entities.Usuarios.FirstOrDefault(u => u.Login == username);

    //No teoricamente improvável caso do usuário não existir...
    if (user == null)
        return new string[] { };


    //Aqui é onde pegamos a lista de permissões do usuário em questão...
    List<String> permissoes = user.Permissoes.Select(p => p.Nome).ToList();

    return permissoes.ToArray();

}
```

*Imagem 10 - Sobrescrevendo o método GetRolesForUser*

Bom amigos, agora nós precisamos informar a nossa aplicação, que nós alteramos a classe que fornece as Permissões de um usuário. Iremos fazer isso modificando ou inserindo uma tag no arquivo de configuração "*Web.config*".

Abra este arquivo, e procure pela tag *<roleManager>* e delete ela. Feito isso, insira ela novamente só que informando agora o local da nova classe que criamos (PermissoesProvider) conforme a imagem abaixo:

```
    </pages>
    <profile defaultProvider="DefaultProfileProvider">
      <providers>
        <add name="DefaultProfileProvider" type="System.Web.Providers.DefaultProfileProvider, System.Web.
      </providers>
    </profile>
    <membership defaultProvider="DefaultMembershipProvider">
      <providers>
        <add name="DefaultMembershipProvider" type="System.Web.Providers.DefaultMembershipProvider, Syste
      </providers>
    </membership>
    <roleManager enabled="true" defaultProvider="PermissaoProvider">
        <providers>
            <add name="PermissaoProvider" type="ExemploAutenticao.Security.PermissaoProvider"/>
        </providers>
    </roleManager>
    <sessionState mode="InProc" customProvider="DefaultSessionProvider">
      <providers>
        <add name="DefaultSessionProvider" type="System.Web.Providers.DefaultSessionStateProvider, System
      </providers>
    </sessionState>
  </system.web>
  <system.webServer>
```

*Imagem 11 - Alterando o Web.config*

O que nós fizemos acima, foi interceptar e alterar o jeito com que as permissões de um usuário são buscadas.

Agora é o ultimo passo para se concretizar o processo. Nós iremos sobrescrever um método da classe "*AuthorizeAttribute*" que é a classe responsável pela anotação "*[Authorize]*".

Crie uma nova classe no diretorio "*Security*" chamada "*PermissoesFiltro*" e estenda ela a classe "*AuthorizeAttribute*" que pertence ao namespace *System.Web.Mvc.* Feito isso, sobrescreva o método "*OnAuthorization(AuthorizationContext filterContext)*" conforme a imagem abaixo:

```
public override void OnAuthorization(AuthorizationContext filterContext)
{
    base.OnAuthorization(filterContext);


    //Caso o usuário não foi autorizado, ele será enviado para a página /Home/Negado/
    if (filterContext.Result is HttpUnauthorizedResult)
        filterContext.HttpContext.Response.Redirect("/Home/Negado");

}
```

*Imagem 12 - Sobrescrevendo o método OnAuthorization*

Nós sobrescrevemos esse método "*OnAuthorization*" para poder dizer para a aplicação o que fazer quando um

acesso for negado.

Agora finalmente, vamos voltar lá no Controller "*UsuarioController*". Delete a anotação "*[Authorize]*" dele, e vamos implementar nele agora as permissões do nosso modo.

Vamos supor que a página "*/Usuarios/Principal*" só poderá ser acessada por usuários que possuem a permissão de "Gerente". Veja como ficaria a anotação do método:

```
//Somente usuários que possuem a permissão com o nome "Gerente" poderão visualizar esta pagina
[PermissoesFiltro(Roles = "Gerente")]
public ActionResult Principal()
{
    return View();
}
```

*Imagem 13 - Configurando as permissões de uma localização*

E vamos supor novamente, que para o usuário acessar a página acima, ele precisa ter duas permissões (Gerente e Avançado). Veja como ficaria a definição:

```
//Somente usuários que possuem a permissão com o nome "Gerente" poderão visualizar esta pagina
[PermissoesFiltro(Roles = "Gerente, Avançado")]
public ActionResult Principal()
{
    return View();
}
```

*Imagem 14 - Duas permissões em uma localização*

Bom pessoal, é isso. O artigo ficou meio cansativo pois é muita coisa para se tratar, mais o MVC é uma forma muito inteligente para trabalhar com aplicações web. Espero que meu compartilhamento tenha sido útil. Obrigado.

**Para saber mais:**
Confira outras publicações sobre este assunto na comunidade ASP.NET
ExemploAutenticacao - Basica.rar
ExemploAutenticacao - Final.rar

Exibições: 7428

Tags: autenticar, autenticação, authentication, authorize, entrar, logar, login, mvc, permissions, permissões, Mais...

Curtir

10 membros curtem isto

Compartilhar Twitter Facebook

Curtir 3

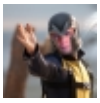
Comentar

## Você precisa ser um membro de DevBrasil para adicionar comentários!

Entrar em DevBrasil

 Comentário de Claudinei Rodrigues em 2 outubro 2013 às 16:22

Cara, muito legal. Parabéns.

 Comentário de Gustavo Adolfo Alencar em 27 setembro 2013 às 15:50

Excelente! Parabéns e obrigado.

 Comentário de Jonas C. Nobre em 19 setembro 2013 às 11:29

Parabéns! Eu gostaria de saber se alguém tem esse código aí? O link indicado está sem o arquivo... Vlw..

 Comentário de Aislan Miranda em 10 setembro 2013 às 15:52

Parabéns pelo post! muito boa a explicação!

 Comentário de Fábio Castro em 25 julho 2013 às 17:32

Muito bom tutorial, gostaria de obter os arquivos desse exemplo..Teria como me passar??Estou precisando muito..Vlw

 Comentário de Matheus Faria Sanches em 22 julho 2013 às 17:25

Excelente tutorial. Parabéns... Gostaria de obter o arquivo de exemplos final por favor.. Obrigado...

 Comentário de Alan Slivar Vieira em 21 junho 2013 às 14:00

Olá! Também gostaria de obter o arquivo de exemplos.

Por sinal, excelente tutorial! Parabens!

 Comentário de Plinio Rodrigues em 6 junho 2013 às 19:49

Alguem tem como me enviar os arquivos desse exemplo ? email pliniorodrigues@Live.com. Obrigado Pessoal.

 Comentário de Rômulo da Costa de Souza em 31 maio 2013 às 9:47

Bom dia Renan, eu estava dando uma olhada em seu artigo, para implementar em um projeto meu, porem não estou tendo sucesso na implementação, estou utilizando asp.net mvc 4, e banco de dados mysql, teria alguma rejeição com isso?

 Comentário de Hugo em 19 março 2013 às 15:55

Valeu Marco. Fico no aguardo de seu retorno, enquanto isso, se quiser, poderemos manter contato para trocar experiências sobre o .NET ok?

Fique na paz!


- ‹ Anterior
- 1
- 2
- 3
- Próximo ›
- Página 2 Ir


RSS

Bem-vindo a
DevBrasil

**Registre-se**
ou acesse

Or sign in with:

- Login



Badges | Relatar um incidente | Termos de serviço

Entrar no bate-papo