Macoratti.net ASP .NET - Usando o NHibernate em uma aplicação Web I

Após muitos artigos dedicados ao **Entity Framework** estou voltando a falar do **NHibernate**, desta vez usando a linguagem C# em um projeto ASP .NET Web Forms.

O **NHibernate** é um porte do consagrado *framework* **Hibernate** para Java para a plataforma .NET, dessa forma o **NHibernate** surgiu a partir do **Hibernate** um framework muito usado na plataforma Java.

Podemos pensar no **NHibernate** como uma ferramenta **ORM** de persistência e ele não esta só pois existem muitas outras ferramentas que se propõe a fazer a mesma coisa. Veja a relação abaixo:

Lista das ferramentas ORM e/ou Frameworks de persistência mais conhecidos:

- ADO.NET Entity Framework, Microsoft's ORM, part of .Net 4.0
- AgileFx, open source
- Base One Foundation Component Library, free or commercial
- Devart LinqConnect, commercial, an ORM solution for Oracle, MySQL, PostgreSQL, and SQLite
- Castle ActiveRecord, ActiveRecord for .NET, open source
- Database Objects .NET, Open Source
- DataObjects.NET, GPL and commercial
- DevForce, commercial, N-Tier
- ECO, Commercial but free for use up to 12 classes
- EntitySpaces, commercial
- Habanero, Free open source Enterprise application framework with a Free Code Generation Tool
- iBATIS, Free open source
- LINQ to SQL, Free, .NET Framework component
- LLBLGen Pro, commercial
- Neo, open source
- NHibernate, open source
- nHydrate, open source
- ObjectMapper .NET, GPL and commercial license
- OpenAccess ORM, free or commercial
- Persistor.NET, free or commercial
- Quick Objects, free or commercial
- SubSonic, open source

A grande vantagem do **NHibernate** é que ela é uma ferramenta gratuita, madura e com uma grande aceitação e utilização na comunidade .NET.

Usando o NHibernate com a linguagem VB .NET em uma aplicação ASP .NET Web Forms

Neste artigo eu vou mostrar como criar uma aplicação simples que mostra como usar o **NHibernate** para fazer o mapeamento objeto relacional e persistir dados no SQL Server 2005 Express Edition.

Recursos necessários:

- **NHibernate 2.1.2.GA**
- **Visual Web Developer 2010 Express Edition**
- **SQL Server 2005 Express Edition**

Existem duas formas de usarmos o **NHibernate**:

1. A partir do modelo de dados já criado realizar o mapeamento ORM para as entidades;
2. Criar a estrutura das tabelas no banco de dados a partir das entidades definidas nos arquivos de configuração; **(Model First)**

Neste artigo eu vou usar a primeira opção onde a partir do modelo de banco de dados e tabela já existente irei realizar o mapeamento gerando as entidades, através dos arquivos de configuração XML (*poderíamos também usar anotações*) do **NHibernate**.

Roteiro de utilização:

- **Efetuar o download da última versão do NHibernate;**
- **Criação do banco de dados e da tabela no SQL Server 2005;**
- **Criação do projeto ASP .NET no Visual Web Developer 2010 Express;**
- **Definição da referência a API do NHibernate;**
- **Criação da classe que irá definir o domínio da aplicação;**
- **Criação e definição do arquivo de mapeamento que irá permitir mapear o objeto para a tabela do banco de dados;**
- **Utilização da API do NHibernate para mapear os objetos e persistir os dados na tabela;**

Definindo o modelo de dados: Banco de dados e tabela

Neste exemplo teremos um banco de dados criado no SQL Server 2005 de nome **Macoratti.mdf** e um única tabela chamada **Usuarios** com a seguinte estrutura:

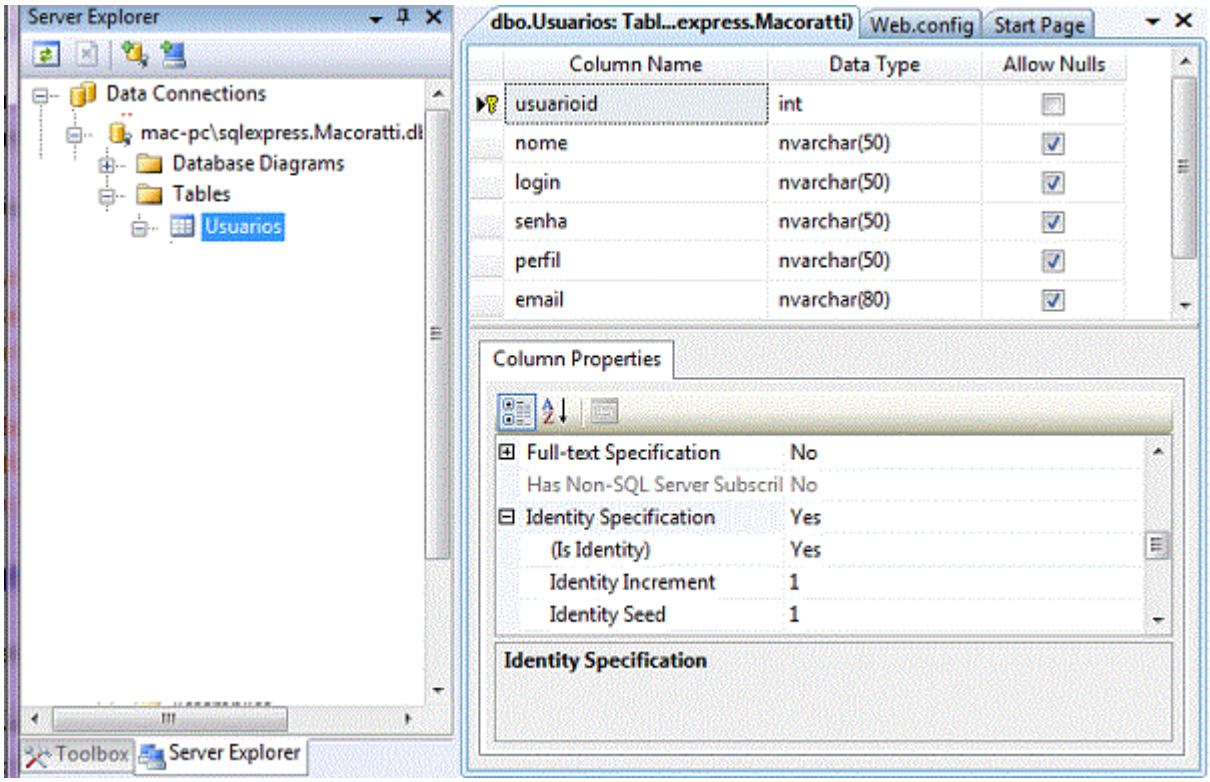


Tabela Usuarios

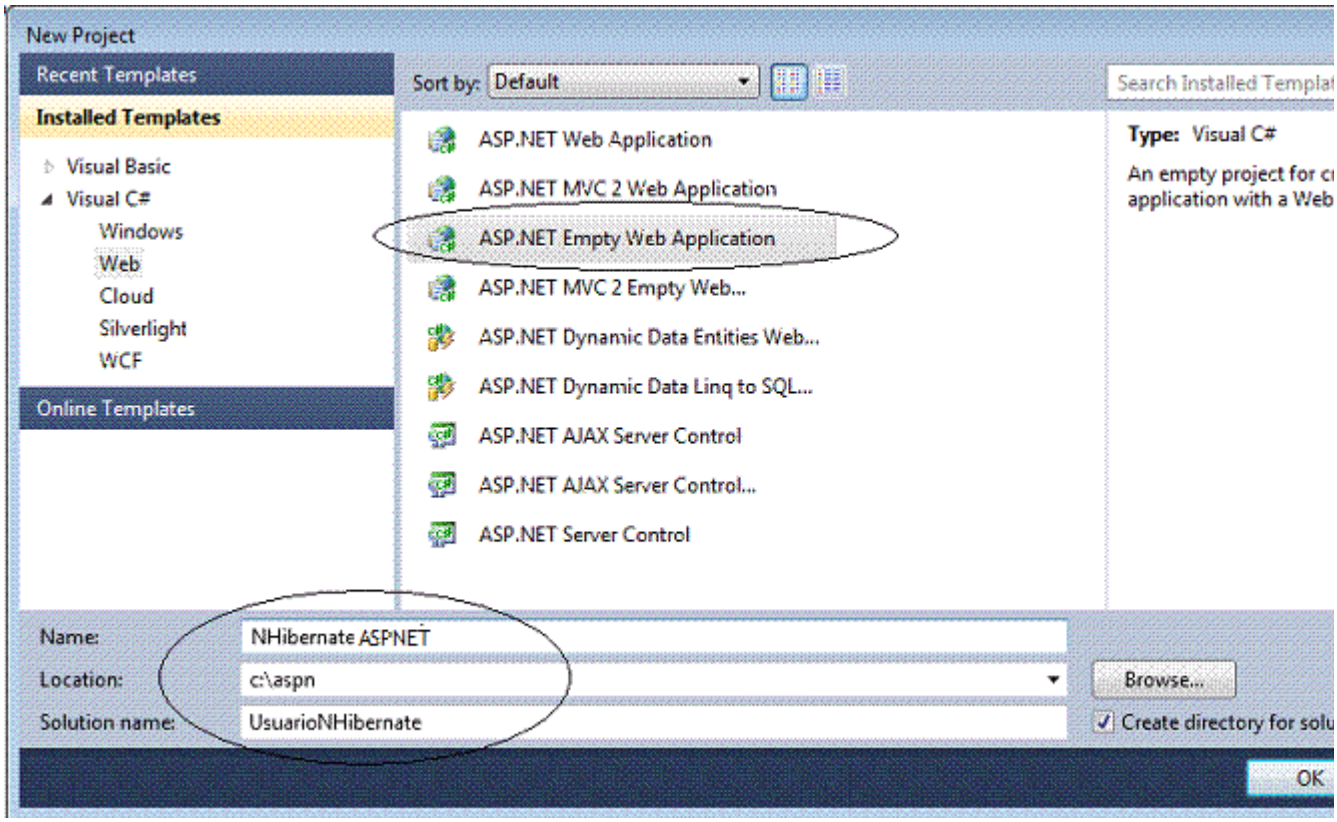
A nossa aplicação **ASP .NET Web Forms** irá gerenciar os usuários permitindo a inclusão , alteração e exclusão de usuários da tabela **Usuarios**.

Note que definimos uma chave primária - **usuarioid** - do tipo **identity**; dessa forma este campo passa a ser gerenciado pelo Banco de dados.

2- Criação do projeto no Visual Web Developer e referência ao NHibernate 2.1.2

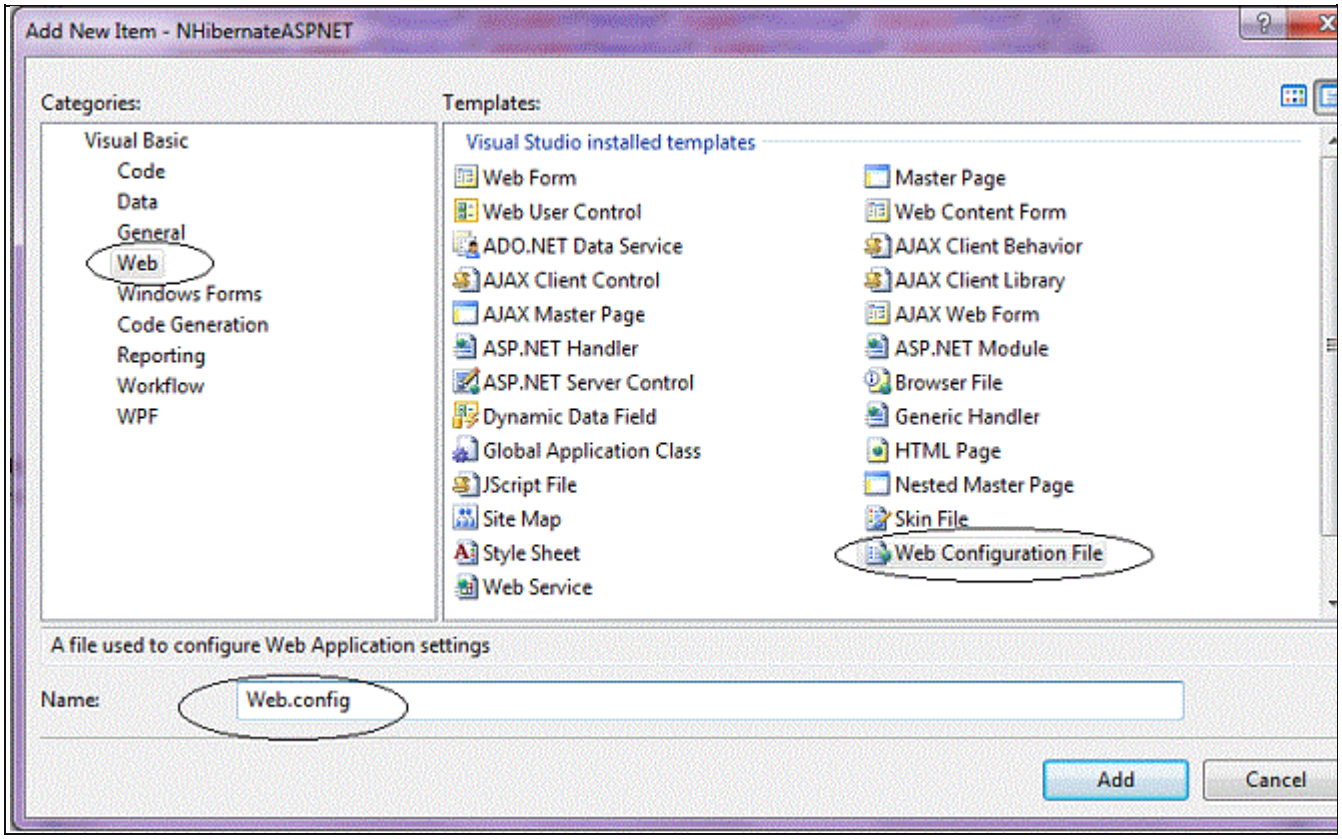
Abra o Visual Web Developer 2010 Express Edition e selecione a opção **New Project;**

A seguir selecione **Visual C# -> Web** e escolha o template **ASP .NET Empty Web Application** e informe o local e o nome do projeto que no nosso caso é **NHibernateASPNET**;

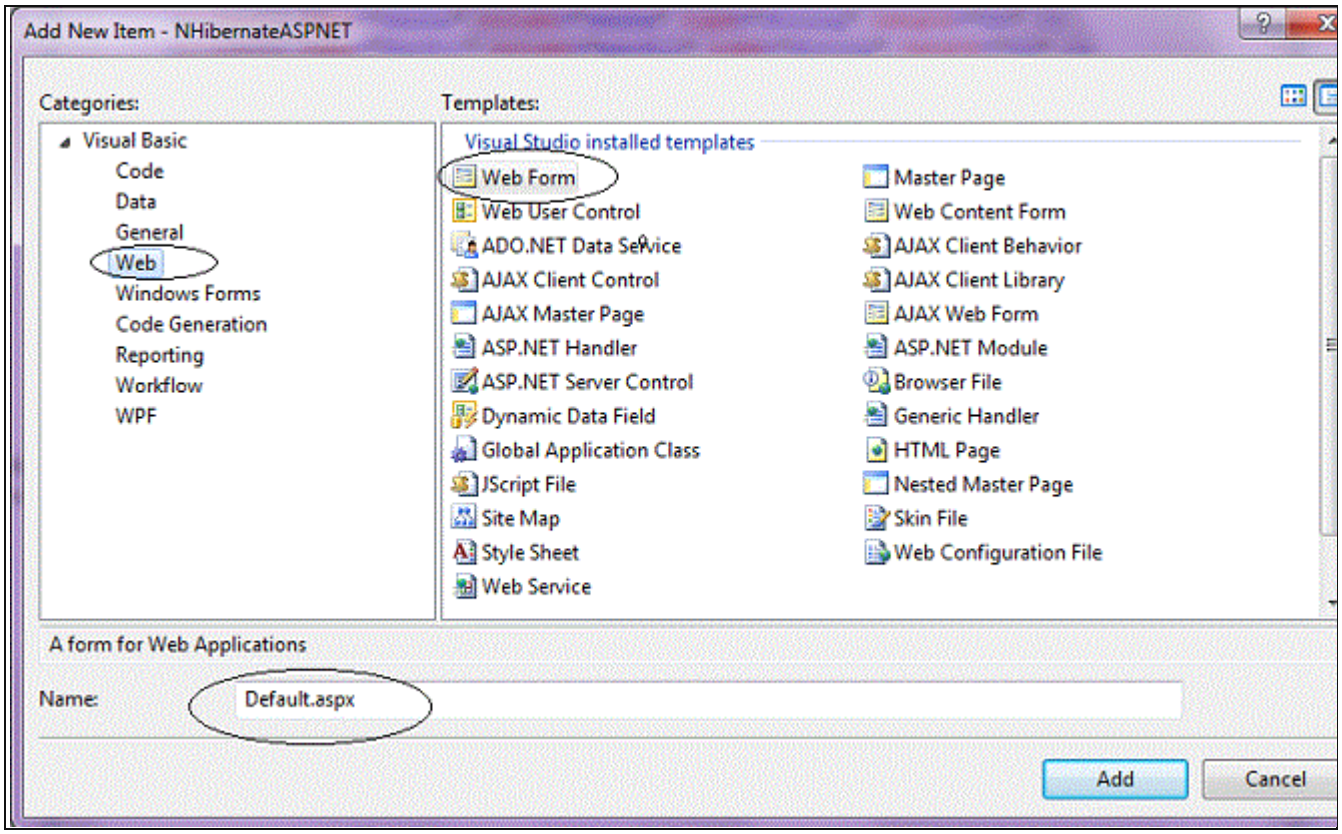


Vamos incluir dois arquivos ao projeto visto que o mesmo esta vazio.

Nome **Project** selecione **Add New Item** e a seguir selecione o template **Web Configuration File** e defina as opções conforme a figura baixo para incluir o arquivo **Web.Config** no projeto clicando no botão **Add**;

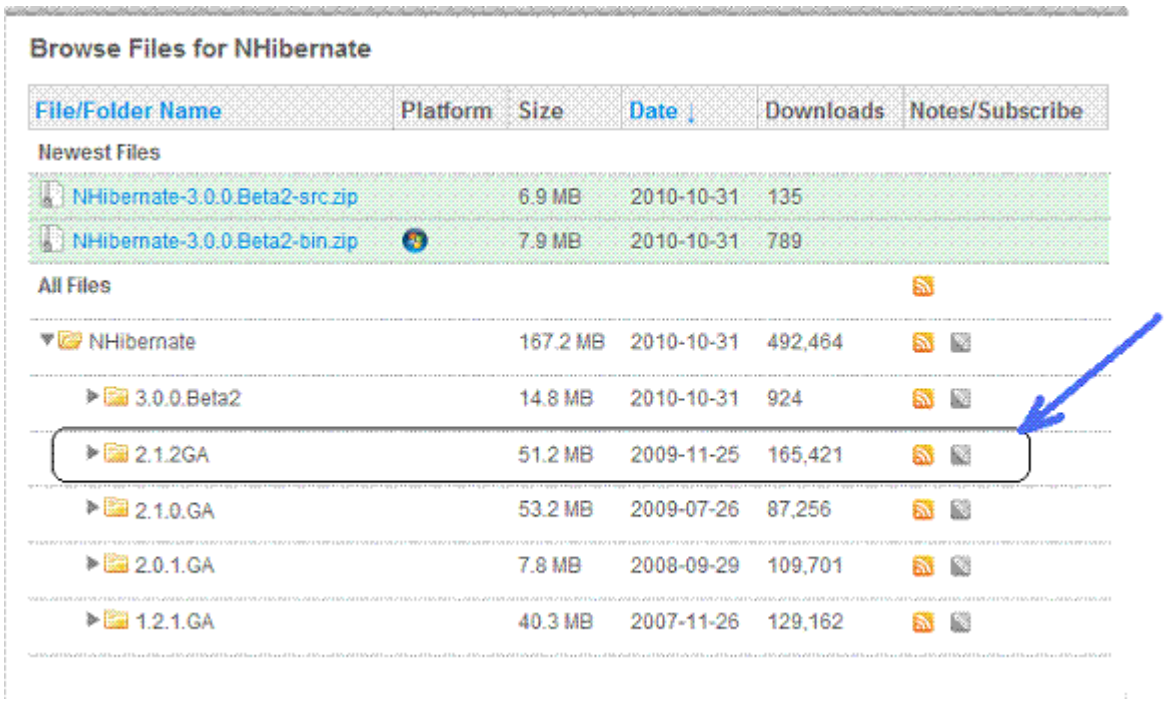


Repita o procedimento acima selecionando o template **Web Form** e informando o nome **Default.aspx** para incluir o um novo Web Form no projeto;

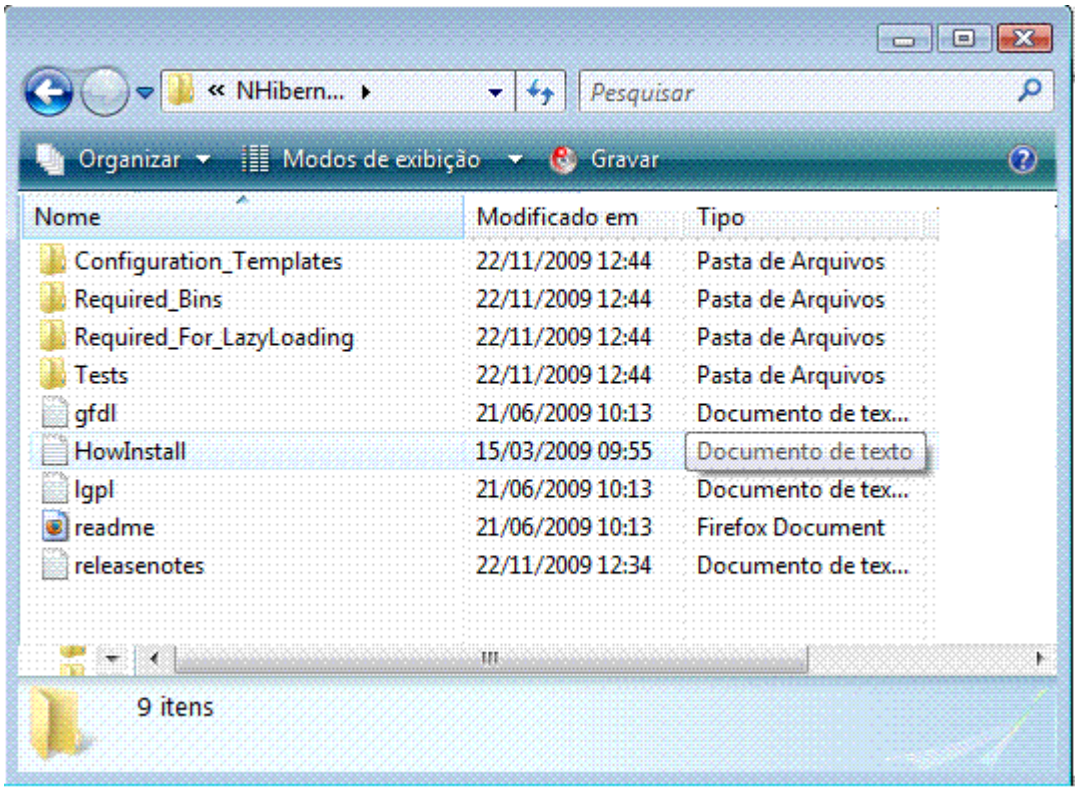


Iremos usar estes arquivos mais à frente.

Agora baixe o **NHibernate 2.1.2** do site: http://sourceforge.net/projects/nhibernate/files/ e descompacte o arquivo em um local apropriado na sua máquina local;

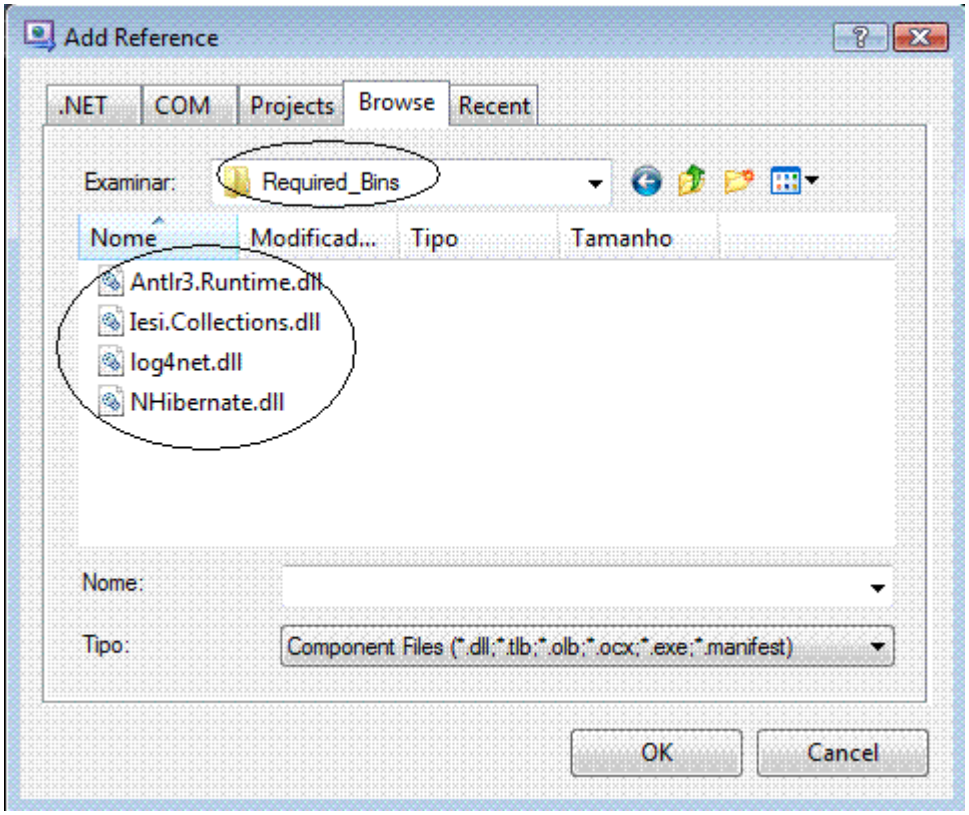


Eu estou descompactando o arquivo na pasta **C:\Program Files\NHibernate-2.1.2.GA-bin**; Abaixo vemos os arquivos desta pasta:

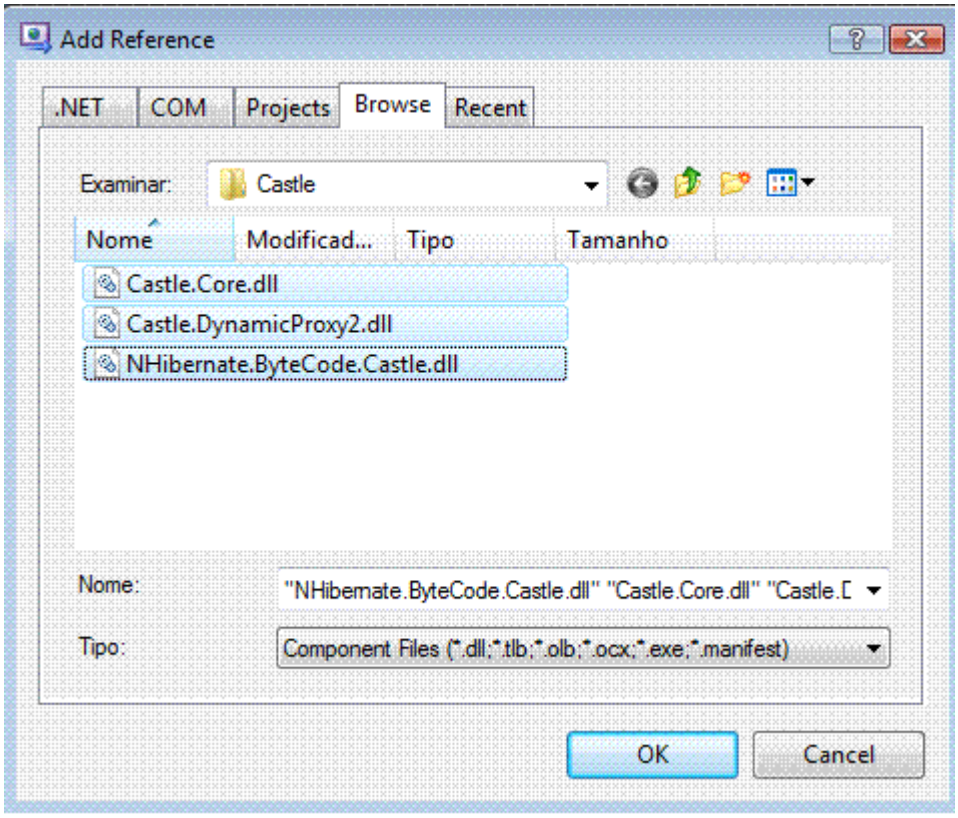


Em seguida no menu **Project -> Add Reference** e na janela **Add Reference** selecione a guia **Browse** e localize a pasta onde você descompactou os arquivos do NHibernate;

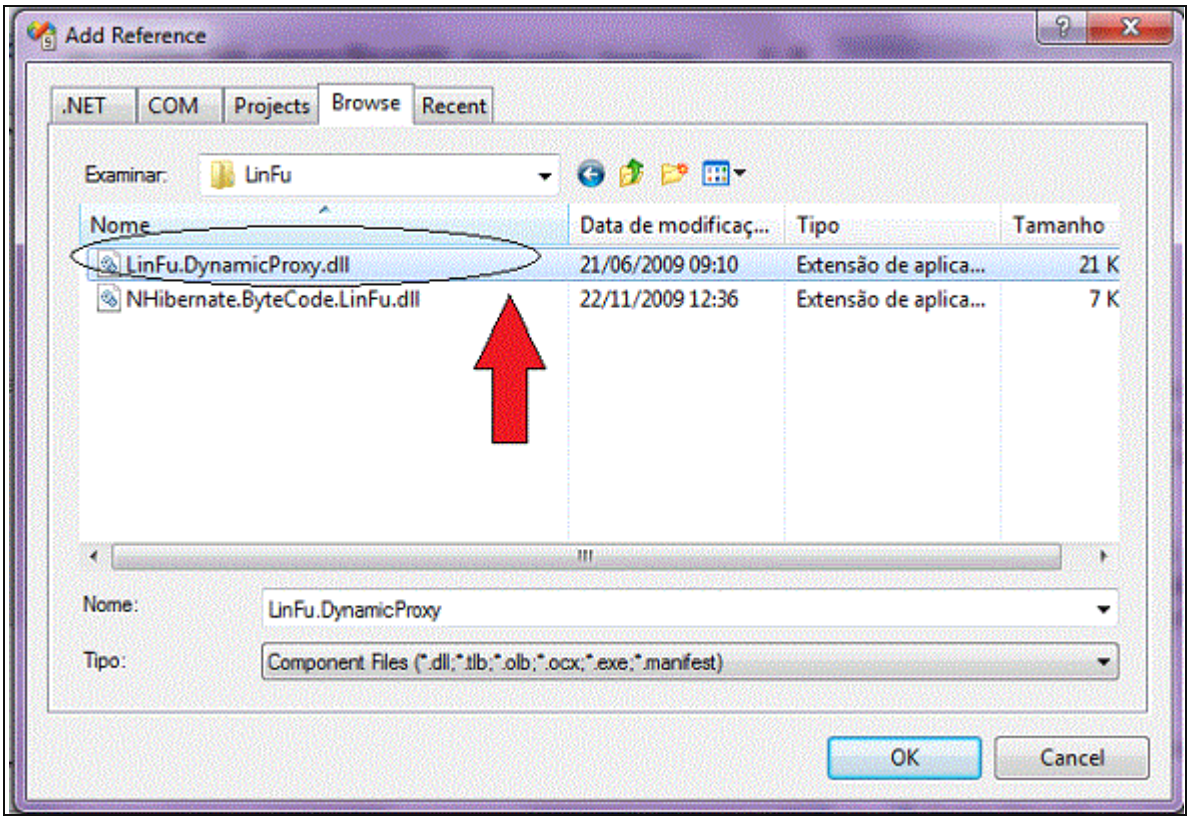
Abra a pasta **Required_Bins** e selecione todos os arquivos e clique em **OK**;



Repita o processo acima incluindo uma referência para a pasta **Required_For_LazyLoading -> Castle,** selecione todos os arquivos e clique em OK;



Após isso precisamos fazer mais uma referência no projeto; trata-se do **assembly** **NHibernate.ByteCode.LinFu.dll** que esta pasta **Required_For_LazyLoading,** subpasta - **LinFu:**



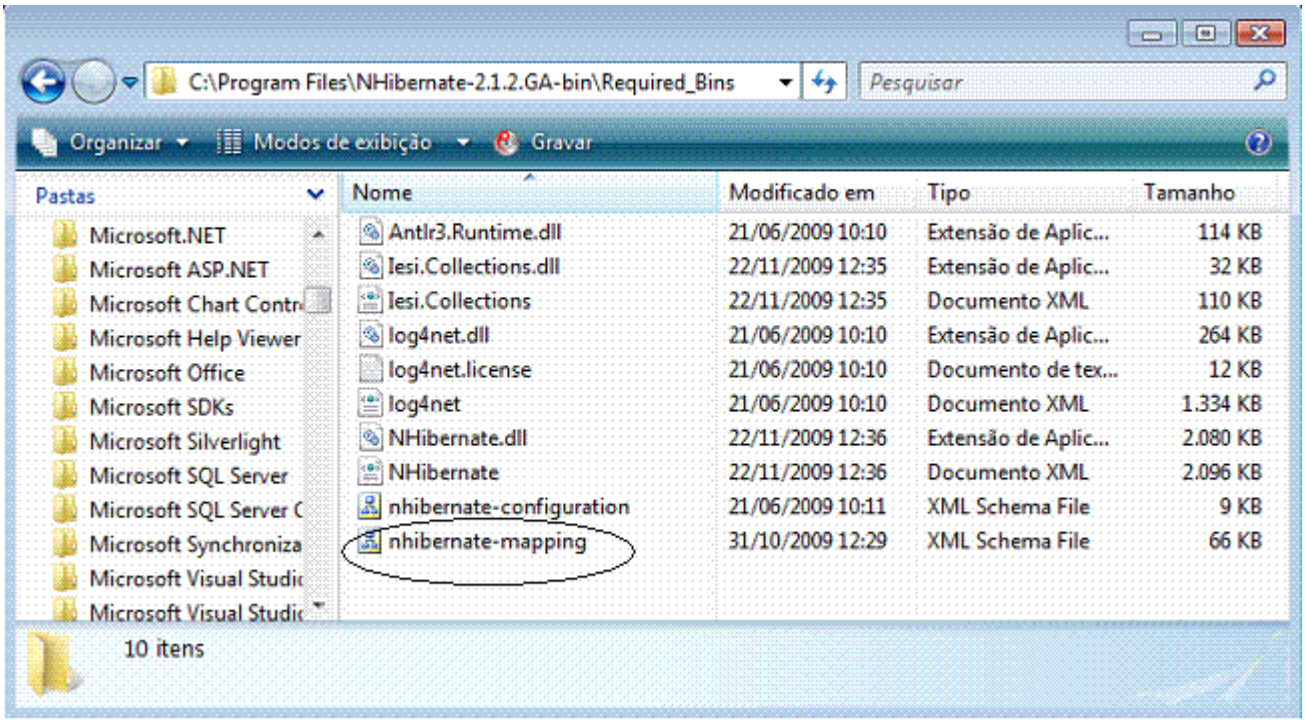
Se não fizermos isso iremos obter o erro: **The ProxyFactoryFactory was not configured. Initialize 'proxyfactory.factory_class' property of the session-factory configuration section with one of the available NHibernate.ByteCode providers.**

Assim já temos todas as principais referências no projeto e com essas referências garantimos o funcionamento do **NHibernate** na maioria dos projetos.

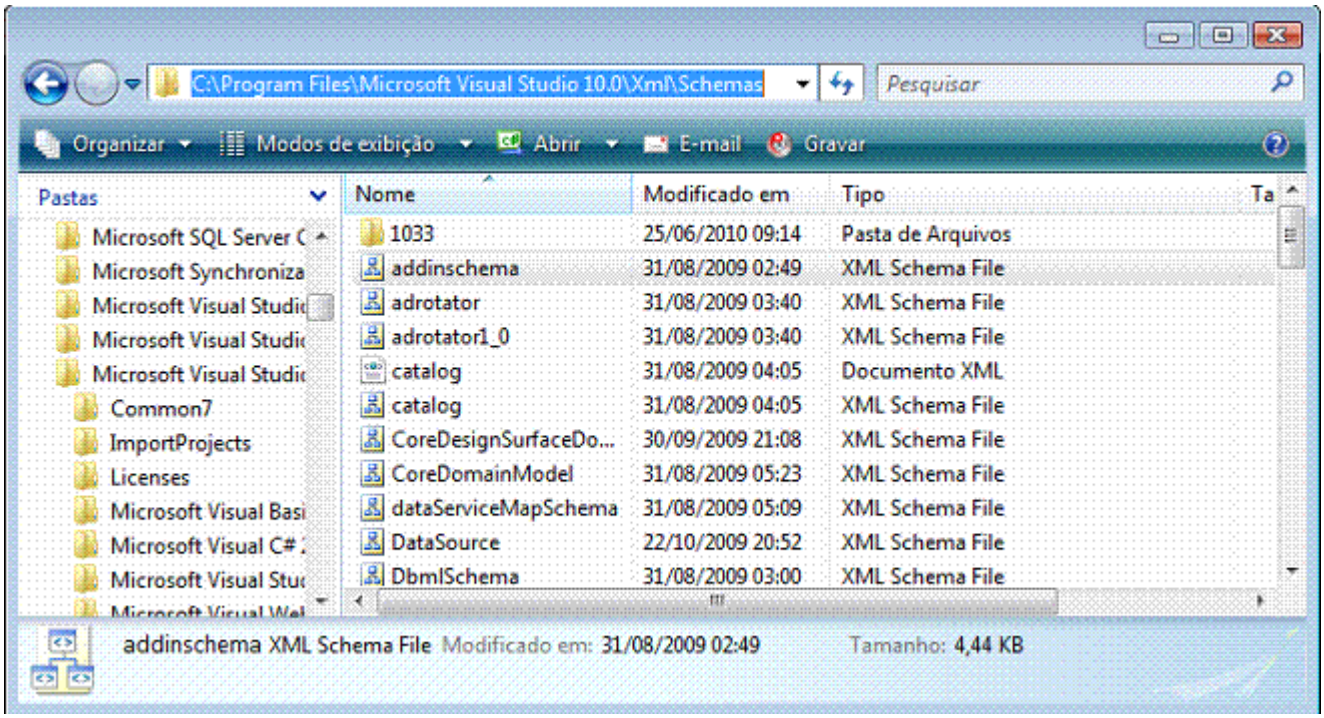
A próxima tarefa é definir os arquivos de mapeamento do **NHibernate**, a forma mais comum de fazer isso é usar arquivos XML.

Podemos obter o recurso do **IntelliSense** no Visual ao realizar a definição dos arquivos XML copiando o arquivo **nhibernate-mapping.xsd** da pasta **Required_Bins** para a pasta **c:\Arquivos de Programas\Microsoft\Visual Studio 10.0\Xml\Schemas**

1- C:\Program Files\NHibernate-2.1.2.GA-bin\Required_Bins



2- C:\Program Files\Microsoft Visual Studio 10.0\Xml\Schemas

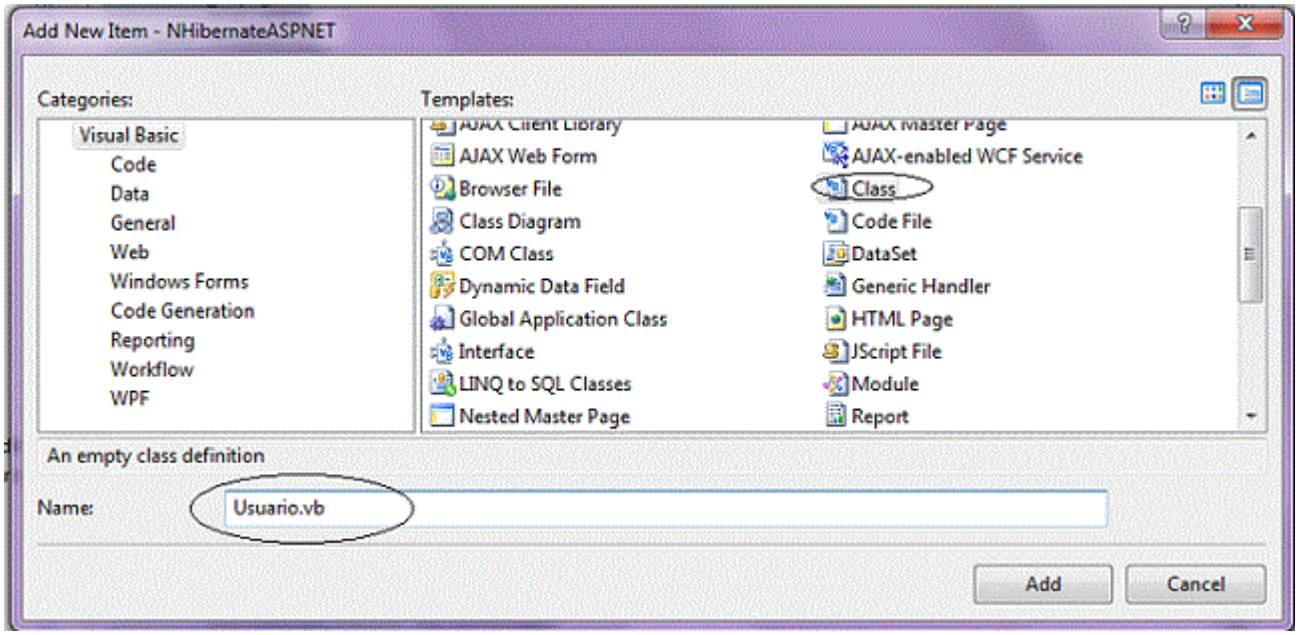


Após isso já podemos criar os arquivos de mapeamento tendo o recurso IntelliSense a disposição o que nos ajuda muito.

Criando a classe de Negócio

Como estamos partindo do modelo de dados já criado agora vamos definir a classe de negócio que representa a tabela **Usuarios**.

No menu **Project** selecione **Add Class** , selecione o template **Class** e informe o nome **Usuario.cs** e clique em **Add;**



A seguir vamos definir o seguinte código no arquivo **Usuario.vb:**

```
Public Class Usuario

Private _usuarioid As Int32
Private _nome As String
Private _login As String
Private _senha As String
Private _perfil As String
Private _email As String

Public Sub New()
End Sub

Public Overridable Property Usuarioid() As Int32
Get
Return _usuarioid
End Get
Set(ByVal value As Int32)
_usuarioid = value
End Set
End Property

Public Overridable Property Nome() As String
Get
Return _nome
End Get
Set(ByVal value As String)
_nome = value
End Set
End Property

Public Overridable Property Login() As String
Get
Return _login
End Get
Set(ByVal value As String)
_login = value
End Set
End Property

Public Overridable Property Senha() As String
Get
Return _senha
End Get
Set(ByVal value As String)
_senha = value
End Set
End Property

Public Overridable Property Perfil() As String
Get
Return _perfil
End Get
Set(ByVal value As String)
_perfil = value
End Set
End Property

Public Overridable Property Email() As String
Get
Return _email
```

```
End Get
Set(ByVal value As String)
_email = value
End Set
End Property
End Class
```

Neste código estamos declarando as propriedades na classe **Usuario**, onde cada propriedade será mapeada para o respectivo campo da tabela **Usuarios**.

No VB.NET indicamos que um método é passível de sobreposição usando a palavra-chave **Overridable** na classe pai (classe base) e a seguir na classe filha declaramos novamente o método com a palavra-chave **Overrides**. Assim temos que :

- **Overridable** - declara que o método pode ser sobreposto nas classes que herdarem da classe base
- **Overrides** - indica que o método da classe filha irá sobrepor o método da classe pai.

Para que um método da classe pai não possa ser sobreposto usamos o modificador - **NotOverridable**. Já, para definir que um método seja obrigatoriamente sobreposto em uma classe filha usamos a palavra-chave - **MustOverride.**

Com a classe **Usuario** definida podemos passar para a etapa de configuração e mapeamento.

Definindo os arquivos de Mapeamento e Configuração

Creio que a parte de configuração é que apresenta maiores problemas para os usuários iniciantes, portanto preste atenção em cada detalhe usado na configuração do NHibernate.

Temos de que definir dois tipos de configuração:

- O arquivo de configuração do NHibernate: Onde indicamos o tipo de banco de dados, drivers, conectores e a string de conexão usada;
- O arquivo de mapeamento do NHibernate: Onde definimos a tabela e o mapeamento entre a(s) classe(s) de negócio e a(s) tabela(s);

- Definindo o arquivo de configuração

A primeira coisa que vamos fazer é definir o arquivo de configuração para que o NHibernate possa funcionar corretamente.

Podemos definir essas configurações no arquivo **Web.Config** ou em um arquivo a parte chamado de **hibernate.cfg.xml**;

Como já temos o arquivo **Web.Config** vamos usá-lo para criar essas configurações.

Abra o arquivo **Web.Config** e inclua o código que esta em azul conforme mostrado a seguir:

```
<?xml version="1.0"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
```

```
   <system.web>
      <compilation debug="true" targetFramework="4.0" />
   </system.web>

  <!--tag definindo seção de configuração do NHibernate -->
  <configSections>
    <section name="hibernate-configuration"
type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
  </configSections>

  <!-- Configuração do NHibernate-->
  <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2" >
    <session-factory>
      <property name="dialect">
        NHibernate.Dialect.MsSql2005Dialect
      </property>
      <property name="connection.provider">
        NHibernate.Connection.DriverConnectionProvider
      </property>
      <property name="connection.driver_class">
        NHibernate.Driver.SqlClientDriver
      </property>
      <property name="connection.connection_string">
        Server=.\SQLEXPRESS;
        Database=Acesso;
        Integrated Security=True;
      </property>
       <property name="proxyfactory.factory_class">
            NHibernate.ByteCode.LinFu.ProxyFactoryFactory, NHibernate.ByteCode.LinFu
        </property>
    </session-factory>
 </hibernate-configuration>

</configuration>
```

Arquivo Web.Config

Observe que tivemos primeiro definir a sessão de configuração para depois criar sessão com as configurações do **NHibernate**.

`<!-- Configuração do NHibernate-->`
`<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2" >`
`<session-factory>`
`<property name="dialect">`
NHibernate.Dialect.MsSql2005Dialect
`</property>`
`<property name="connection.provider">`
NHibernate.Connection.DriverConnectionProvider
	Na sessão de configuração do arquivo **web.config** definimos os seguintes itens:
	• No arquivo acima definimos o provedor de acesso a base de dados SQL Server 2005: **NHibernate.Dialect.MsSql2005Dialect ;**
	• O provedor usado: **NHibernate.Connection.DriverConnectionProvider;**

`</property>`
`<property name="connection.driver_class">`
`NHibernate.Driver.SqlClientDriver`
`</property>`
`<property name="connection.connection_string">`
`Server=.\SQLEXPRESS;`
`Database=Acesso;`
`Integrated Security=True;`
`</property>`
`<property name="proxyfactory.factory_class">`
 	
`NHibernate.ByteCode.LinFu.ProxyFactoryFactory,`
`NHibernate.ByteCode.LinFu`
`</property>`
`</session-factory>`
`</hibernate-configuration>`
	• O driver apropriado: `NHibernate.Driver.SqlClientDriver;`
	• Definimos também a string de conexão usada: `Server=.\SQLEXPRESS; Database=Acesso; Integrated Security=True;`
	• A proxyFactory : `NHibernate.ByteCode.LinFu.ProxyFactoryFactory, NHibernate.ByteCode.LinFu`

- Definindo o arquivo de mapeamento

Como já temos a classe de negócio criada vamos definir o arquivo de mapeamento baseado nesta classe.

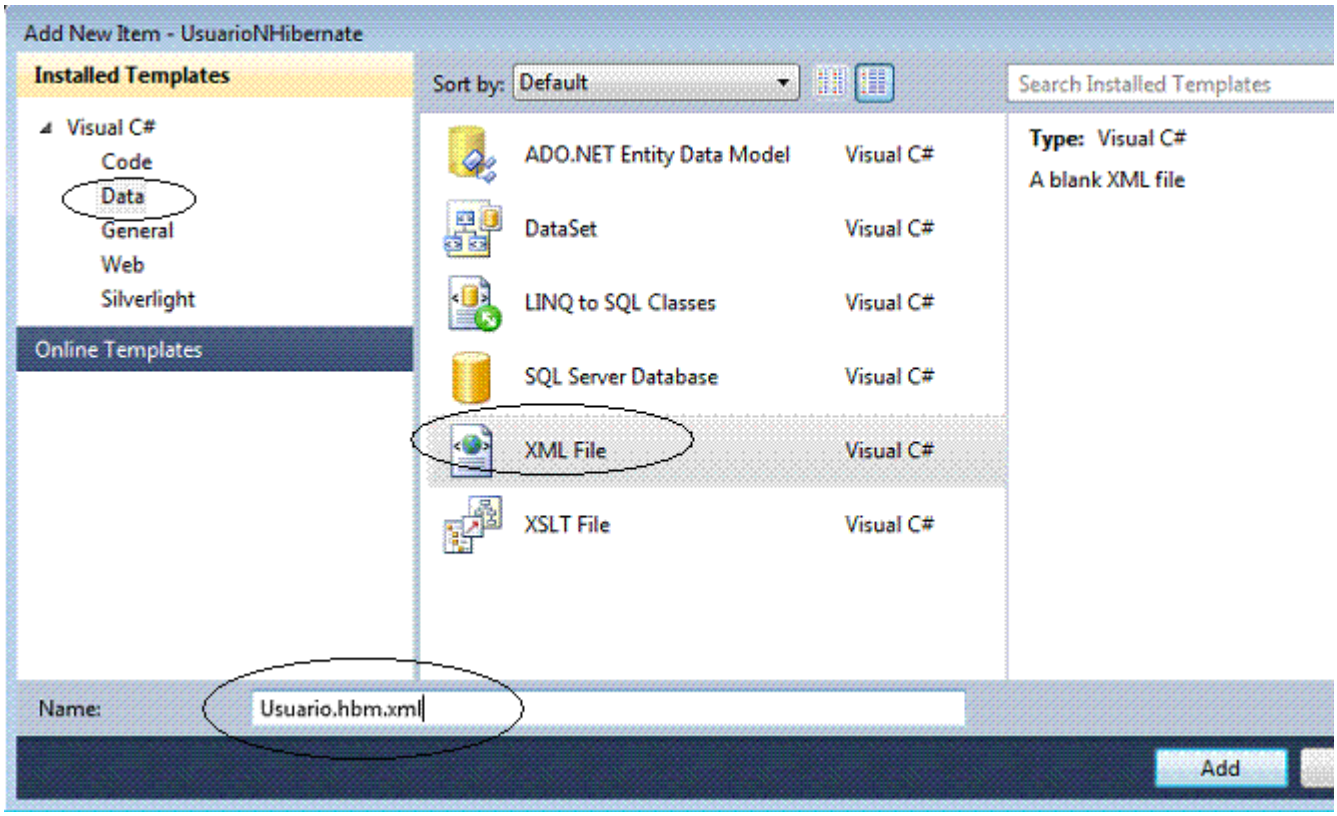
O mapeamento é o coração do **NHibernate** e se não for feito corretamente vai lhe dar muita dor de cabeça.

No arquivo de mapeamento simplesmente especificamos qual tabela no banco de dados estamos relacionando com a classe de negócio.

Podemos definir este mapeamento em um arquivo XML separado ou usando atributos nas classes, propriedades e membros.

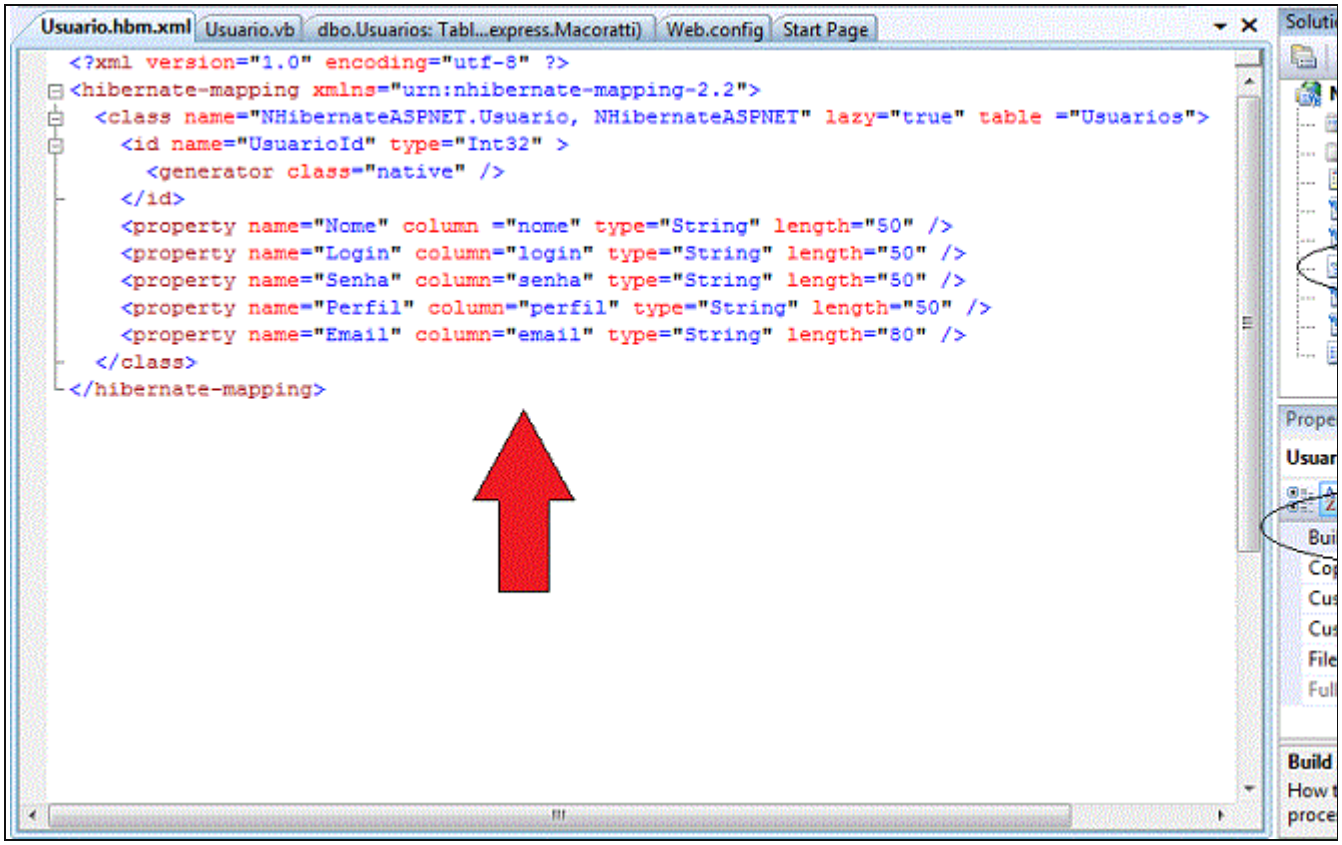
Vamos criar um arquivo XML no projeto. No menu **Project** selecione **Add New Item**;

A seguir selecione o item **Data** e o *template* **XML File** e informe o nome **Usuario.hbm.xml** e clique em **Add**;



A seguir devemos realizar duas tarefas com o arquivo **Usuario.hbm.xml:**

1- Alterar a sua propriedade **Build Action** para : **Embedded Resource;**



2- Definir a tabela e o mapeamento entre a classe de negócio e as tabelas usadas:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-
2.2">
<class name="NHibernateASPNET.Usuario,
NHibernateASPNET" lazy="true" table ="Usuarios">
<id name="UsuarioId" type="Int32" >
    <generator class="native" />
</id>
<property name="Nome" column ="nome" type="String"
length="50" />
<property name="Login" column="login" type="String"
length="50" />
<property name="Senha" column="senha" type="String"
length="50" />
<property name="Perfil" column="perfil" type="String"
length="50" />
<property name="Email" column="email" type="String"
length="80" />
</class>
</hibernate-mapping>
```

Como nosso exemplo usa somente uma tabela (de propósito) fica fácil definir o mapeamento conforme o código acima, onde temos:

Agora vamos entender o significado deste código deste arquivo:

- - Este é um arquivo de mapeamento usado pelo **NHibernate** que mapeia as classes para a tabelado banco de dados e possui a terminação **nomedaclasse.hbm.xml.** *(no exemplo o nome da classe é **Usuario**)*
- - Para cada classe deve haver um arquivo de mapeamento.
- - Os elementos para o mapeamento objeto relacional encontram-se entre as tags **<hibernate-mapping>**

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">

Esta linha indica o inicio do mapeamento(**hibernate-mapping)** feito NHibernate e deve ser usada na versão 2.1.2(**rn:nhibernate-mapping-2.2)**.

Atenção: (Se você estiver usando uma versão anterior do NHibernate não use esta sintaxe)

<class name="NHibernateASPNET.Usuario, NHibernateASPNET" table="Usuarios">

-Aqui definimos o nome da classe na tag **class :<class name="NHibernateASPNET.Usuario, NHibernateASPNET"** ***(Note que incluímos o nome do Assembly)***

-O atributo **name** deve conter o *namespace* e o nome do **Assembly**.

-O atributo **table** deve indicar o nome da tabela : **table="Usuarios"** **(se o nome da tabela for igual ao da classe não precisa ser declarado)**

Vejamos agora o mapeamento da chave primária:

```
<id name="Usuarioid" column="usuarioid" type="Int32">
    <generator class="native" />
</id>
```

Nesta definição temos o mapeamento da chave primária onde a propriedade **Usuarioid** é mapeada para a coluna da tabela **usuarioid**.

A tag **id** identifica a chave primária e o atributo **name="Usuarioid"** informa o nome do atributo da classe .NET que se refere à chave primária da tabela.

O atributo **column** informa ao **NHibernate** que coluna na tabela é a chave primária, no caso **usuarioid**.

O atributo **generator** informa qual a estratégia para geração da chave primária. Para o nosso exemplo usamos a estratégia **native** que significa que o **NHibernate** irá usar a estratégia que melhor se adequar ao banco de dados podendo ser: **identity, sequence ou hilo** dependendo das capacidades do banco de dados.

. Outros valores possíveis são:

1. **Increment** - Lê o valor máximo da chave primária e incrementa;
2. **Identity** - Mapeado para colunas identity no DB2, MySQL, MSSQL, SyBase, Informix;
3. **sequence** - Mapeado em sequências no DB2, PostGreSQL, Oracle, SAP DB, Firebird;
4. **hilo** - Usa um algoritmo high/low para gerar chaves únicas;
5. **uudi.hex** - Usa uma combinação do IP com um timestamp para gerar um identificador único.
6. **guid** - uUa o novo system.guid como identificador;

Após isso temos os mapeamentos das propriedades persistentes através da tag **property**.

```
<property name="Nome" column ="nome" type="String" length="50" />
<property name="Login" column="login" type="String" length="50" />
<property name="Senha" column="senha" type="String" length="50" />
<property name="Perfil" column="perfil" type="String" length="50" />
<property name="Email" column="email" type="String" length="80" />
```

As tags **property** indicam propriedades simples dos objetos onde o nome das propriedades das classes são definidos pelo atributo **name**, o tipo pelo atributo **type** e a coluna da tabela a que se refere pelo atributo **column**.

No nosso exemplo temos o mapeamento das propriedades **Nome,Login, Senha, Perfil e Email** para as colunas **nome,login,senha,perfil e email** do mesmo tipo.

Se o atributo **column** não aparecer no mapeamento da propriedade, o **NHibernate** considera que a coluna na tabela do banco de dados a que se referencia possui o mesmo nome que o definido pelo atributo **name**.

Dessa forma definimos os arquivos de configuração e mapeamento e já temos a classe de negócio definida e o banco de dados criado.

Só falta definir a interface no projeto ASP .NET onde iremos usar a página **Default.aspx** para realizar a manutenção dos dados dos usuários na tabela **Usuarios** e definir uma sessão **NHibernate** para realizar as operações desejadas persistindo-as no banco de dados.

Já temos portanto tudo pronto para usar os recursos do NHibernate em nossa aplicação **ASP .NET** e realizar as operações **CRUD** relacionadas com as manutenção das informações da tabela **Usuario**.

Definindo uma sessão NHibernate

Mas afinal o que vem a ser uma sessão NHibernate ?

Pense em uma sessão NHibernate como uma ligação abstrata ou virtual de um conduto ou fio com o banco de dados.

Quando se usa o NHibernate não temos que nos preocupar em *criar uma conexão, abrir a conexão, passar a conexão para o objeto Command e criar um DataReader a partir do objeto Command,etc*.

Com o NHibernate o tratamento é diferente. Temos que solicitar um objeto **Session** para o **SessionFactory** e usar a sessão para realizar as operações **CRUD**.

O NHibernate trabalha com um mecanismo de sessão e para criar uma sessão usamos um objeto do tipo **ISessionFactory** do NHibernate.

Além de criar a sessão devemos tomar o cuidado de ela será inicializada apenas uma única vez durante a sessão do usuário na aplicação WEB para evitar degradação no desempenho da aplicação.

Para isso vamos definir uma classe chamada **NHibernateHelper** no projeto.

No menu **Project** selecione o item **Add Class,** selecione o template **Class** e informe o nome **NHibernateHelper.vb**;

A seguir vamos definir o seguinte código nesta classe:

```
Imports NHibernate
Imports NHibernate.Cfg

Public Class NHibernateHelper
    Private Shared _sessionFactory As ISessionFactory
```

```
    Private Shared ReadOnly Property SessionFactory()
As ISessionFactory
        Get
            'se a sessão não existir cria e retorna uma
sessão
            If _sessionFactory Is Nothing Then
                Dim configuration = New Configuration()
                configuration.Configure()

configuration.AddAssembly(GetType(Usuario).Assembly)
                _sessionFactory =
configuration.BuildSessionFactory()
            End If
            Return _sessionFactory
        End Get
    End Property

    Public Shared Function OpenSession() As ISession
        Return SessionFactory.OpenSession()
    End Function
End Class
```

Note que temos a variável estática (*shared*) e privada **_sessionFactory** do tipo ISessionFactory.

É através desse objeto que iremos criar as sessões NHibernate e como ele é estático garantimos que a sessão será criada uma única vez.

O método **SessionFactory** é o método principal da classe sendo o responsável por criar o ISessionFactory e retorná-la.

No código verificamos se não existe uma sessão criada e em caso positivo criamos a sessão.

Na primeira execução do código teremos que criar o ISessionFactory e para isso temos que recuperar as configurações do arquivo de configuração do NHibernate através do método **Configure()** da classe Configuration.

Em seguida registramos os arquivos de mapeamento através do método **AddAssembly** que usa o nome do Assembly onde estão os arquivos de mapeamento do NHibernate.

O método **OpenSession**() cria o **SessionFactory** e retorna um objeto ISession que representa a sessão NHibernate.

Agora vamos criar uma interface no projeto onde iremos definir os métodos que iremos usar na aplicação;

No menu **Project -> Add Class** , informe o nome **IUsuarioRepositorio.vb** e clique em **Add**;

A seguir defina o seguinte código nesta interface:

```
Imports System.Collections.Generic

Public Interface IUsuarioRepositorio

  Sub Add(ByVal product As Usuario)
  Sub Update(ByVal product As
Usuario)
  Sub Remove(ByVal product As
Usuario)
  Function GetById(ByVal id As Guid)
As Usuario
  Function GetByLogin(ByVal login As
String) As Usuario
  Function getAllUsuarios()
  Function GetByPerfil(ByVal perfil As
String) As ICollection(Of Usuario)

End Interface
```

Apenas para lembrar em uma interface definimos apenas as assinaturas dos métodos que deverão ser implementados por quem for implementar a interface.

Para implementar a interface vamos criar uma classe concreta: No menu **Project -> Add Class** , informe o nome **UsuarioRepositorio.vb** e clique em **Add**;

A seguir vamos definir o código que vai implementar cada um dos métodos definidos na interface:

```
Imports NHibernate
Imports NHibernateASPNET
Imports NHibernate.Criterion
Public Class UsuarioRepositorio
    Implements IUsuarioRepositorio

    Public Sub Add(ByVal usuario As Usuario) Implements
IUsuarioRepositorio.Add
        Using session As ISession = NHibernateHelper.OpenSession()
            Using transaction As ITransaction = session.BeginTransaction()
                session.Save(usuario)
                transaction.Commit()
            End Using
        End Using
    End Sub

    Public Function GetById(ByVal id As System.Guid) As Usuario
Implements IUsuarioRepositorio.GetById
        Using session As ISession = NHibernateHelper.OpenSession()
            Return session.[Get](Of Usuario)(id)
        End Using
```

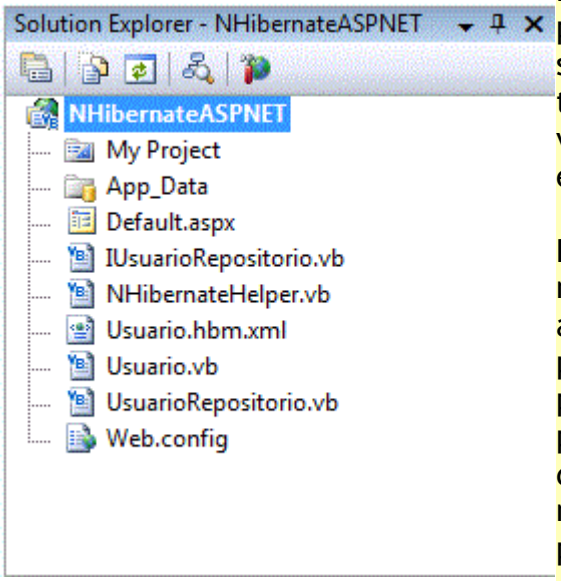
```
    End Function

    Public Function GetByLogin(ByVal login As String) As Usuario
Implements IUsuarioRepositorio.GetByLogin
        Using session As ISession = NHibernateHelper.OpenSession()
            Dim product As Usuario =
session.CreateCriteria(GetType(Usuario)).Add(Restrictions.Eq("Login",
login)).UniqueResult(Of Usuario)()
            Return product
        End Using
    End Function

    Public Sub Remove(ByVal usuario As Usuario) Implements
IUsuarioRepositorio.Remove
        Using session As ISession = NHibernateHelper.OpenSession()
            Using transaction As ITransaction = session.BeginTransaction()
                session.Delete(Usuario)
                transaction.Commit()
            End Using
        End Using
    End Sub

    Public Sub Update(ByVal usuario As Usuario) Implements
IUsuarioRepositorio.Update
        Using session As ISession = NHibernateHelper.OpenSession()
            Using transaction As ITransaction = session.BeginTransaction()
                session.Update(Usuario)
                transaction.Commit()
            End Using
        End Using
    End Sub
    Public Function getAllUsuarios() As Object Implements
IUsuarioRepositorio.getAllUsuarios
        Using session As ISession = NHibernateHelper.OpenSession()
            Dim usuarios As IList =
session.CreateCriteria(GetType(Usuario)).List()
            Return usuarios
        End Using
    End Function

      Public Function GetByPerfil(ByVal perfil As String) As
System.Collections.Generic.ICollection(Of Usuario) Implements
IUsuarioRepositorio.GetByPerfil
        Using session As ISession = NHibernateHelper.OpenSession()
            Dim usuarios =
session.CreateCriteria(GetType(Usuario)).Add(Restrictions.Eq("Perfil",
perfil)).List(Of Usuario)()
            Return usuarios
        End Using
    End Function
```

```
End Class
```

Destaques :

- Note que usamos a palavra chave **implements** e o nome da interface que vamos implementar;
- Em cada requisição de uma sessão estamos usando o recurso **Using** de forma que o recurso usado é liberado ao término da utilização;
- As transações que envolvem atualização no banco de dados usam transações;

Neste momento a nossa solução deverá ter a seguinte estrutura:



Eu optei criar as classes e os arquivos de configuração em um mesmo projeto e no mesmo local.

Fiz isso porque o projeto é bem simples e para tornar mais fácil a visualização da estrutura.

Para projetos maiores é aconselhável criar pastas distintas para as classes e para os arquivos de configuração no mesmo projeto.

Pode-se ainda criar projetos distintos para dividir a solução em camadas o que esta mais aderente as boas práticas.

Agora só falta definirmos a interface da aplicação que será a página **ASP .NET Default.aspx** e usar os recursos que criamos até aqui no projeto.

Agora sim já temos portanto tudo pronto para usar os recursos do NHibernate em nossa aplicação **ASP .NET** e realizar as operações **CRUD** relacionadas com as manutenção das informações da tabela **Usuario**.

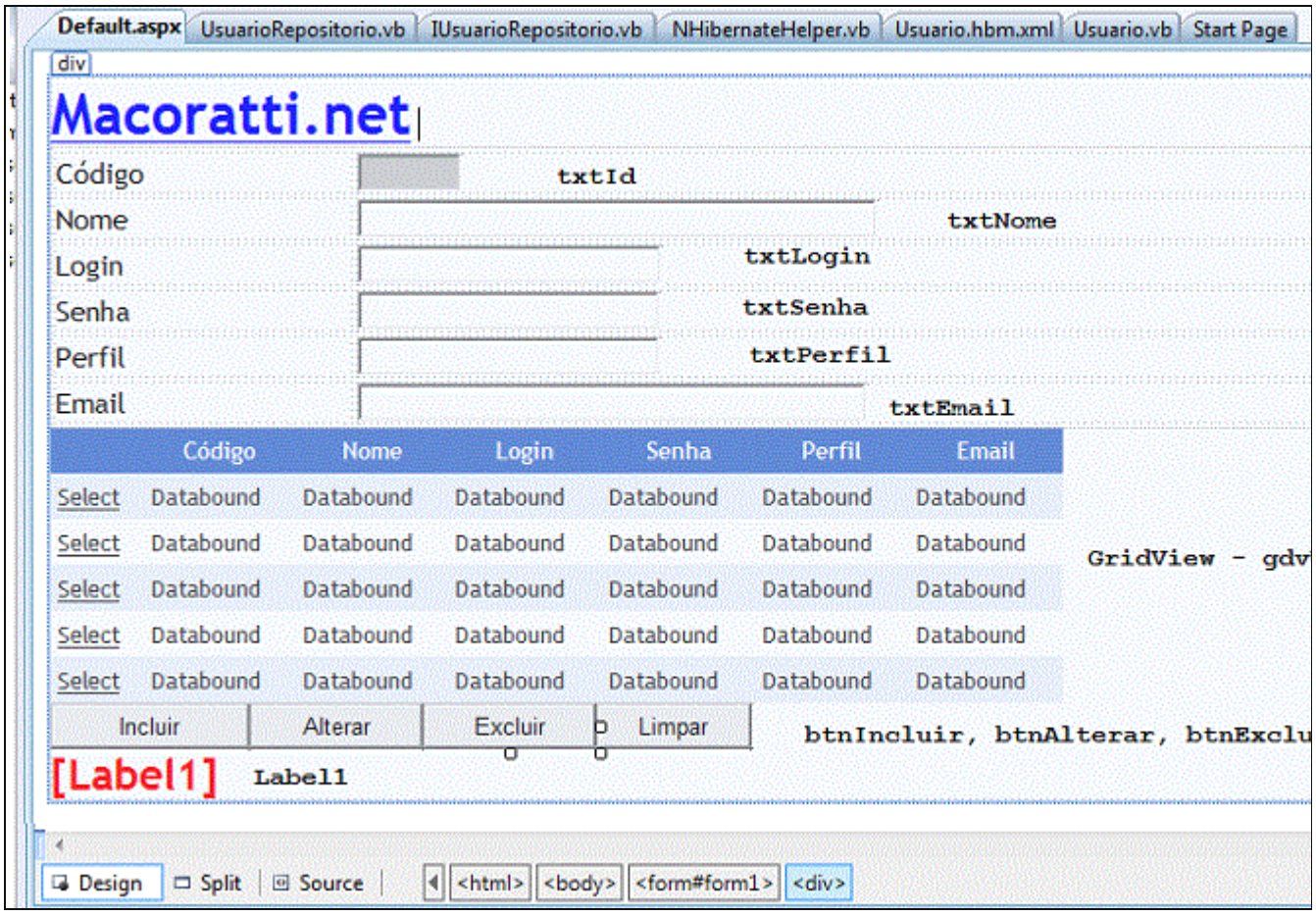
Agora só falta definirmos a interface da aplicação que será a página **ASP .NET Default.aspx** e usar os recursos que criamos até aqui no projeto.

Então ao trabalho...

Selecione e abra a página Default.aspx e inclua a partir da toolbox nesta páginas os controles:

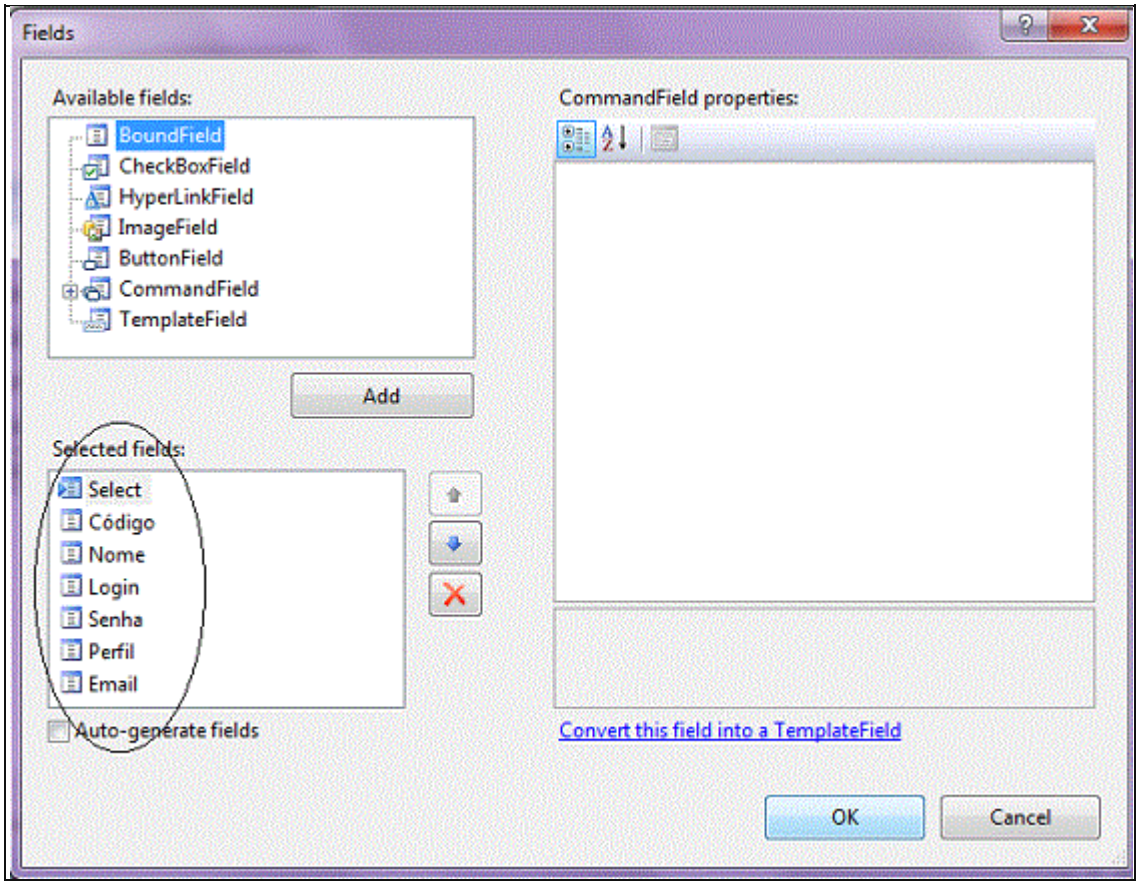
- 7 Labels -
- 6 TextBox
- 1 GridView
- 4 Buttons

Conforme o leiaute da figura abaixo (os IDs dos controles estão definidos abaixo);



É importante definir a propriedade **DatKeyNames** do controle GridView como sendo igual a **usuarioid**;

Editando o controle GridView na opção **Edit Columns** deveremos configurar o controle conforme abaixo:



Agora temos que definir o código usado na página **Default.aspx** no arquivo code-behind **Default.aspx.cs**.

Primeiro definimos o *namespace* usado na página:

```
Imports NHibernate
```

A seguir definimos uma variável que representa a classe **UsuarioRepositorio** que implementa os métodos CRUD e usa a sessão **NHibernate**;

```
Dim repositorio As UsuarioRepositorio
```

Agora vamos definir o código em cada evento da página conforme a seguir:

No evento **Load** da página temos o código abaixo que chama a rotina **carregaUsuarios**:

```
Protected Sub Page_Load(ByVal sender As Object,
ByVal e As EventArgs) Handles Me.Load
        If Not IsPostBack Then
            carregaUsuarios()
        End If
    End Sub
```

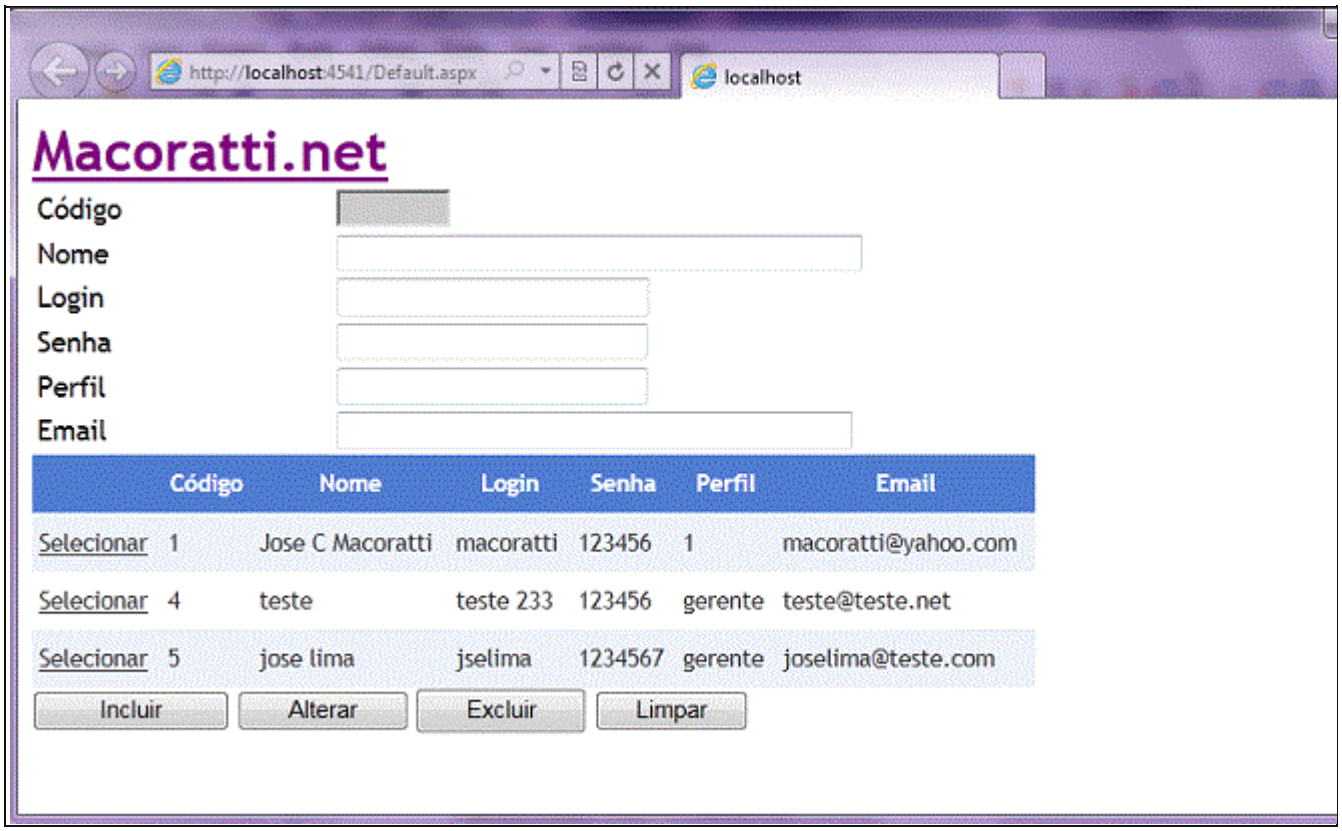
A rotina **carregaUsuarios**() possui o seguinte código:

```
 Private Sub carregaUsuarios()
```

```
        repositorio = New UsuarioRepositorio()
        Dim resultado = repositorio.getAllUsuarios()
        gdvUsuarios.DataSource = resultado
        gdvUsuarios.DataBind()
    End Sub
```

Neste código criamos uma instância da classe **UsuarioRepositorio** e usamos o método **getAllUsuarios()** para obter os usuários e exibir no **GridView**;

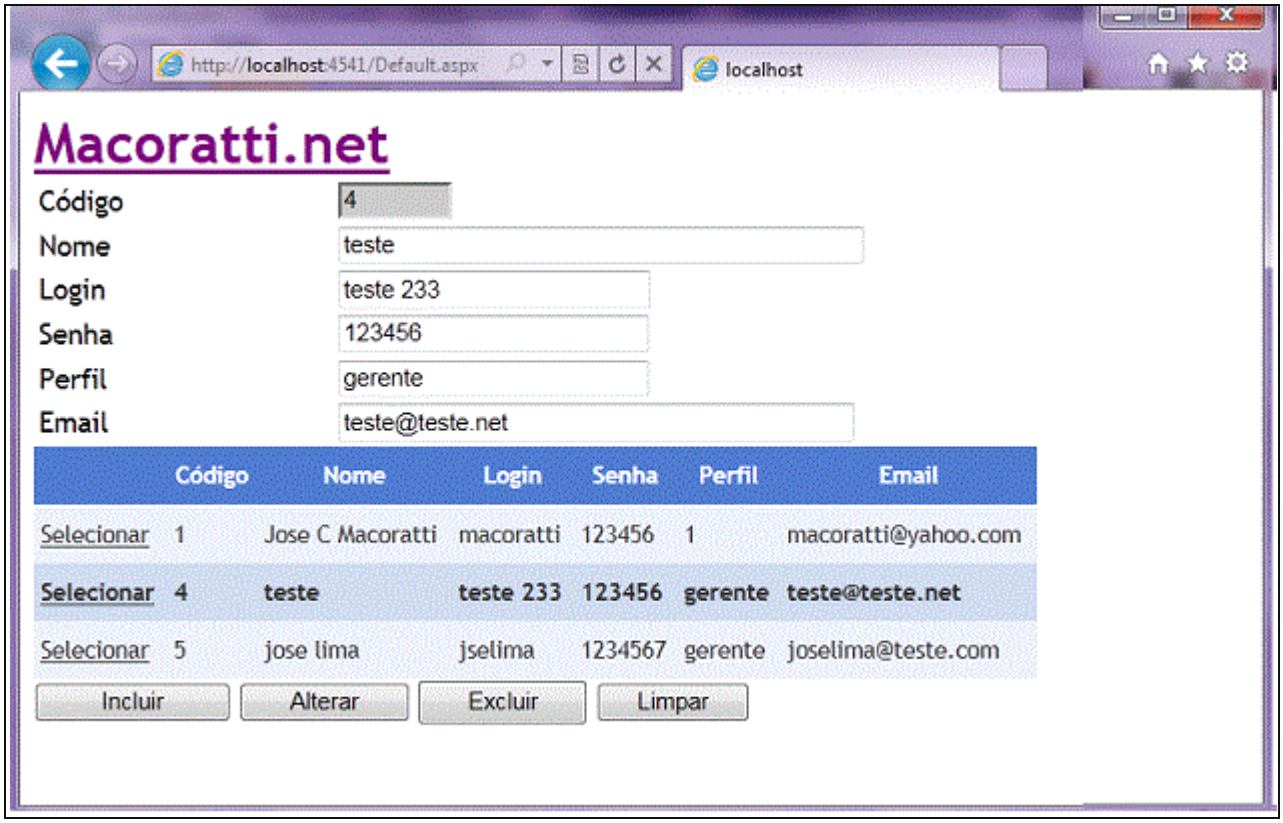
Ao ser executada o projeto teremos a exibição da página abaixo:



No evento **SelectedIndexChanged** temos o código que verifica se uma linha do grid foi selecionada e em caso positivo exibe os dados nos controles **TextBox** da página:

```
Protected Sub gdvUsuarios_SelectedIndexChanged(ByVal sender As Object, ByVal e As
EventArgs) Handles gdvUsuarios.SelectedIndexChanged
        If gdvUsuarios.SelectedDataKey IsNot Nothing Then
            Label1.Text = String.empty
            txtId.Text = gdvUsuarios.SelectedDataKey.Value.ToString()
            txtNome.Text = gdvUsuarios.SelectedRow.Cells(2).Text
            txtLogin.Text = gdvUsuarios.SelectedRow.Cells(3).Text
            txtSenha.Text = gdvUsuarios.SelectedRow.Cells(4).Text
            txtPerfil.Text = gdvUsuarios.SelectedRow.Cells(5).Text
            txtEmail.Text = gdvUsuarios.SelectedRow.Cells(6).Text
        End If
    End Sub
```

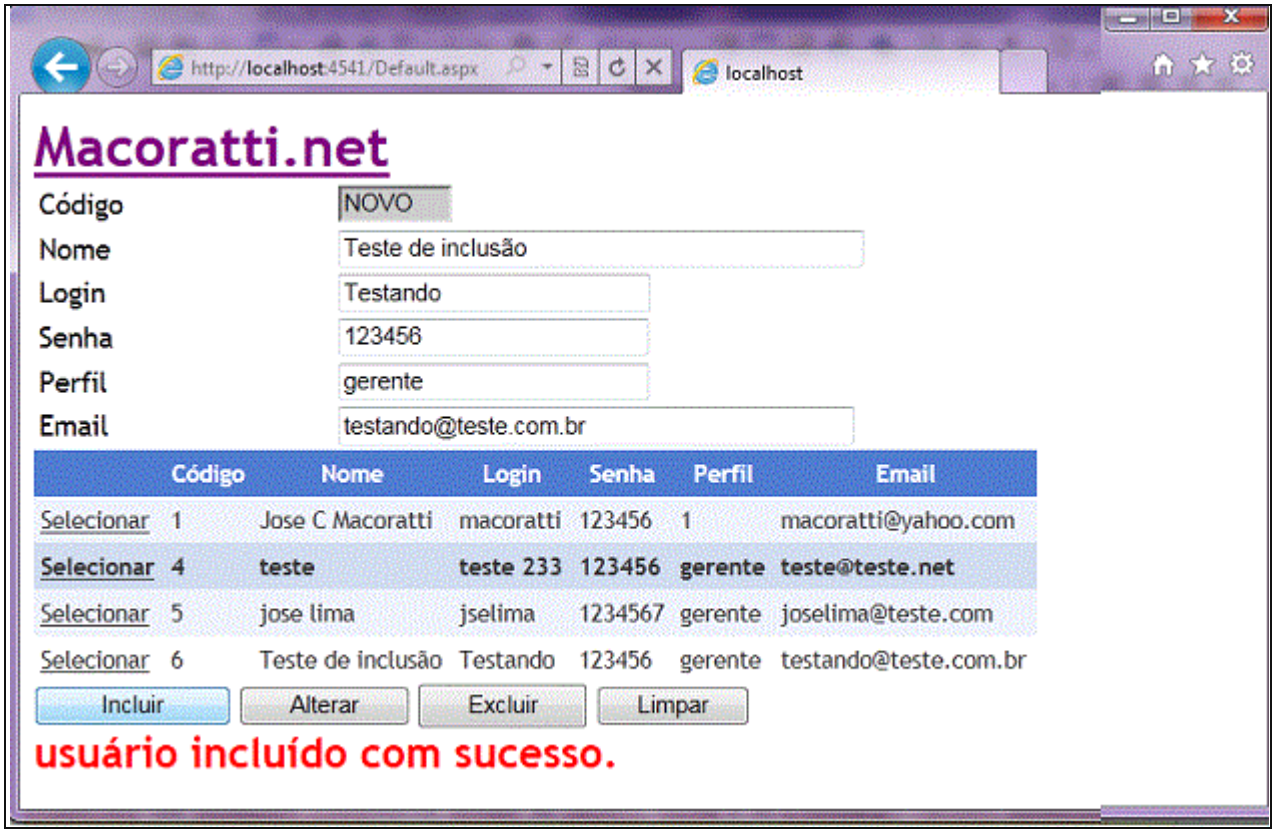
O resultado pode ser visto na figura abaixo:



O código do evento **Click** do botão Incluir é mostrado a seguir. No código é criado uma instância de **usuario** e atribuído os valores informados na página web e em seguida usamos o método **Add** para incluir o usuário;

```
Protected Sub btnIncluir_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles btnIncluir.Click
        Dim usuario As New Usuario()
        usuario.Nome = txtNome.Text
        usuario.Login = txtLogin.Text
        usuario.Senha = txtSenha.Text
        usuario.Perfil = txtPerfil.Text
        usuario.Email = txtEmail.Text
        repositorio = New UsuarioRepositorio()
        repositorio.Add(usuario)
        Label1.Text = "usuário incluído com sucesso."
        carregaUsuarios()
    End Sub
```

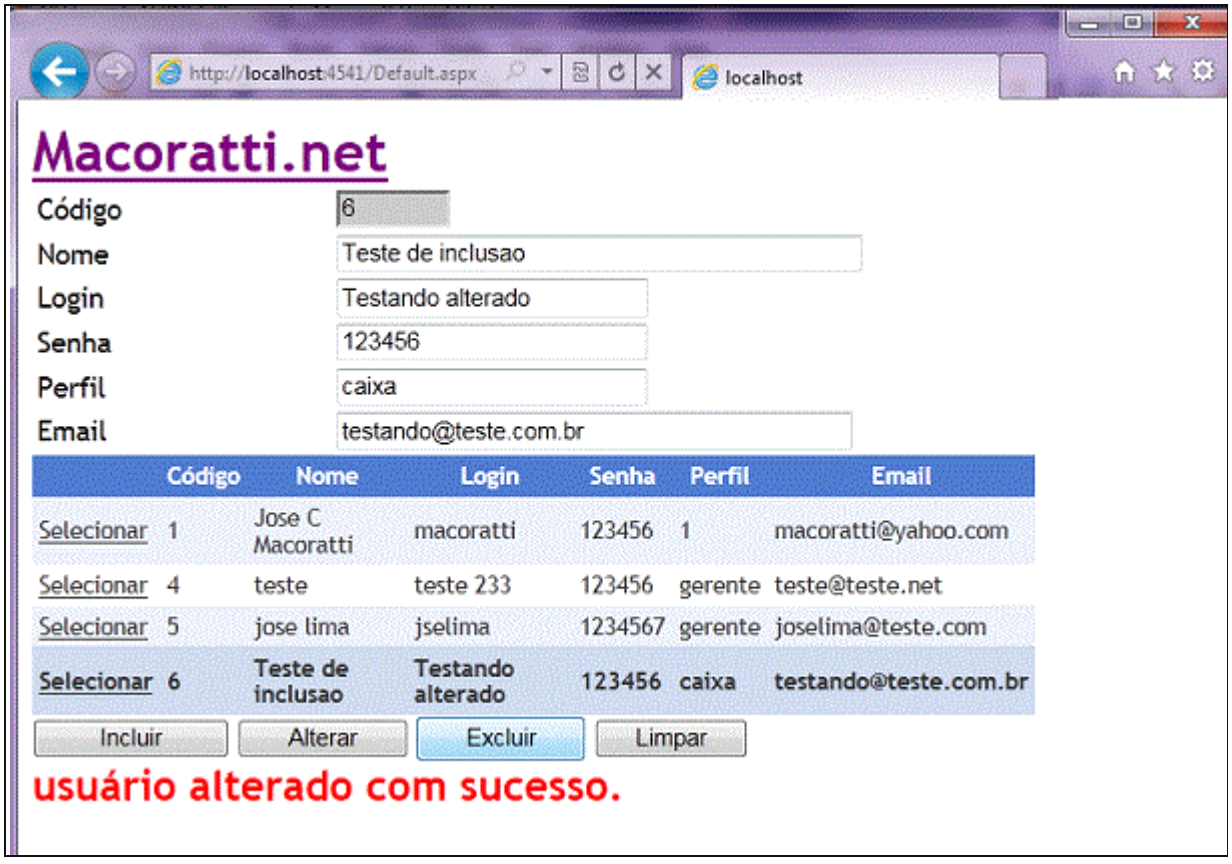
O resultado é exibido a seguir:



No evento **Click** do botão Alterar temos o código que cria uma nova instância de **usuario** e usa o método **Update** para alterar o registro selecionado:

```
Protected Sub btnAlterar_Click(ByVal sender As Object, ByVal e
As EventArgs) Handles btnAlterar.Click
        Label1.Text = String.empty
       Dim usuario As New Usuario
       usuario.UsuarioId = Convert.ToInt32(txtId.Text)
       usuario.Nome = txtNome.Text
       usuario.Login = txtLogin.Text
       usuario.Senha = txtSenha.Text
       usuario.Perfil = txtPerfil.Text
       usuario.Email = txtEmail.Text
       repositorio = New UsuarioRepositorio()
       repositorio.Update(usuario)
       Label1.Text = "usuário alterado com sucesso."
       carregaUsuarios()
   End Sub
```

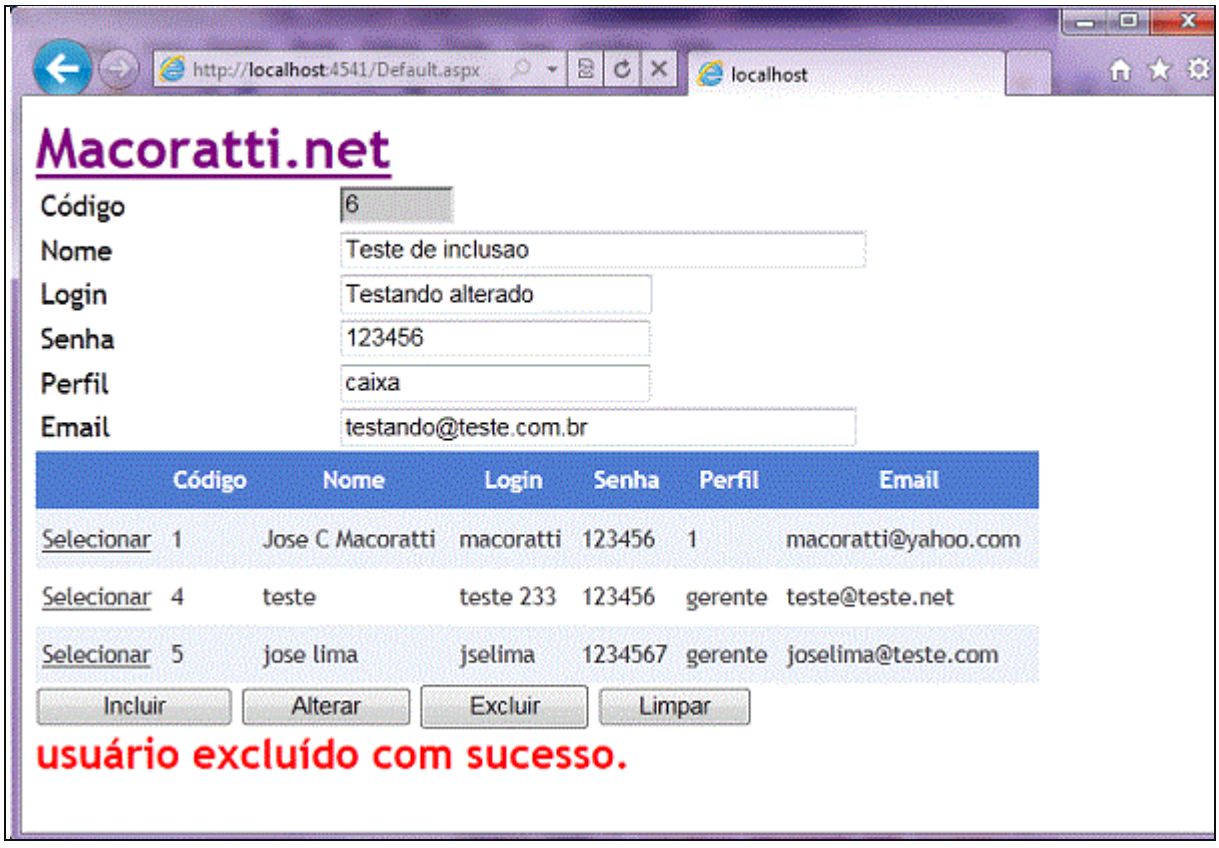
Abaixo vamos o resultado da alteração de um registro:



Para excluir um registro temos no evento **CLick** do botão **Excluir** o código abaixo que usa o método **Remove** passando o objeto **usuario** a ser excluído.

```
Protected Sub btnExcluir_Click(ByVal sender As Object,
ByVal e As EventArgs) Handles btnExcluir.Click
        Label1.Text = String.empty
        Dim usuario As New Usuario
        usuario.UsuarioId = Convert.ToInt32(txtId.Text)
        repositorio = New UsuarioRepositorio()
        repositorio.Remove(usuario)
        Label1.Text = "usuário excluído com sucesso."
        carregaUsuarios()
    End Sub
```

A exclusão do registro é mostrada na figura a seguir: *(Podemos melhorar incluindo uma mensagem solicitando a confirmação da exclusão)*



O código do botão Limpar é dado a seguir e apenas limpa o conteúdo dos controles da página. Ele é usado para incluir um novo usuário e para limpar os controles:

```
Protected Sub btnLimpar_Click(ByVal sender As
Object, ByVal e As EventArgs) Handles
btnLimpar.Click
        txtId.Text="NOVO"
        txtNome.Text=""
        txtLogin.Text=""
        txtSenha.Text=""
        txtPerfil.Text=""
        txtEmail.Text = ""
        txtNome.Focus()
    End Sub
```

Dessa forma encerramos a série de 3 artigos onde mostramos como usar o NHibernate para realizar as operações **CRUD** de uma maneira mais consistente usando a sessão NHibernate de uma forma otimizado.

Podemos melhorar ainda mais criando mais recursos em nosso projeto mas isso é assunto para outro artigo...

Pegue o projeto completo aqui:

Eu sei é apenas NHibernate mas eu gosto...

Referências:

- http://sourceforge.net/projects/nhibernate
- VB.NET - Primeiros passos - Conceitos - V
- NHibernate - Usando o NHibernate 2.1 com o SharpDevelop 3.0 (C#)
- NHibernate - Usando o ActiveRecord - Macoratti.net
- VB .NET - Usando o NHibernate - Macoratti.net

José Carlos Macoratti