Documentação Básica TOP.Framework

Autor: Charles Wellington de Oliveira Fortes

Versão: 0.1 – revisão 1– jul/2012

Sumário

[Estrutura Arquitetural 3](#_Toc329029907)

[Core Business 3](#_Toc329029908)

[Infraestrutura de acesso a dados 4](#_Toc329029909)

[Infraestrutura 5](#_Toc329029910)

[Camada de Apresentação – MVC 3 5](#_Toc329029911)

[Dependencias 5](#_Toc329029912)

[ORM 5](#_Toc329029913)

[HelpersMVC 5](#_Toc329029914)

[MVCContrib 5](#_Toc329029915)

[Hicharts.net 6](#_Toc329029916)

[Passos para criar um cadastro novo 6](#_Toc329029917)

**Figuras**

[Figura 1 - Estrutura de armazenamento das entidades do core business 5](#_Toc329029918)

[Figura 2 - Infraestrutura do core business 5](#_Toc329029919)

[Figura 3 - Estrutura de acesso a dados 6](#_Toc329029920)

[Figura 4 - infraestrutura web 6](#_Toc329029921)

[Figura 5 - Estrutura da apresentação Web 6](#_Toc329029922)

**Top Framework**

# Estrutura Arquitetural

Infraestrutura

Core Business

Infraestrutura de acesso a dados

Banco de Dados

Interface – MVC3

A organização dos dados do sistema se dará de forma que consigamos facilmente substituir ou portar os elementos que compõem o sistema em uma série de plataformas distintas.

O Core Business do sistema é o coração das regras de negócio do sistema, que apoiado em toda a infraestrutura de suporte, permitirá que os demais itens possam ser substituídos facilmente conforme a plataforma.

Os detalhes de implementação de cada um destas camadas no framework se dará da forma como descrito abaixo.

## Core Business

O core business do framework é composto é composto pelas entidades do sistema, serviço de dados e interfaces de contrato de acesso a dados.

As entidades são organizadas dentro da pasta DomainObjects, sendo organizadas em subpastas que representarão cada módulo.

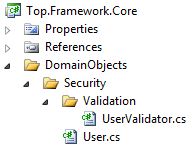


Figura 1 - Estrutura de armazenamento das entidades do core business

No exemplo da Figura 1 a classe que representa o usuário está organizada na pasta DomainObjects dentro do módulo de segurança.

Dentro da pasta de cada módulo deverá conter uma pasta “Validation” onde serão armazenadas os validadores das entidades.

Os itens de infraestrutura necessárias ao funcionamento geral do Core Business serão armazenadas na pasta “Infrastructure”. Na raiz desta pasta encontramos a classe abstrata base que representa as entidades do sistema e as interfaces básicas para as entidades e validadores (Figura 2)

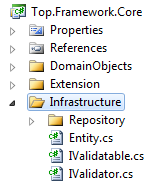


Figura 2 - Infraestrutura do core business

## Infraestrutura de acesso a dados

Os itens de acesso a dados serão armazenados na “Repository” dentro de “Infrastructure”. Na raiz desta pasta estão as interfaces que representam as classes de acesso a dados e as pastas onde se encontram as implementações do ORM.

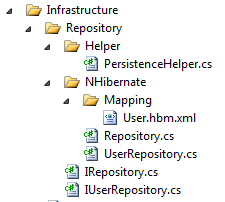


Figura 3 - Estrutura de acesso a dados

A classe “PersistenceHelper” possui métodos de que encapsulam a abertura de sessão com o banco de dados.

A pasta Mapping possui as classes de mapeamento do banco de dados, e na raiz da pasta NHibernate temos as entidades concretas de acesso a dados, sendo que todo repositório deve herdar de “Repository”.

## Infraestrutura

A camada de infraestrutura auxiliar do sistema é composta por classes e métodos auxiliares que não têm relação com o domínio do problema, como a classe “MatchLocalizedAtribute” (Figura 4) da infraestrutura de acesso web.

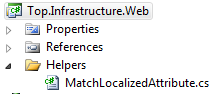


Figura 4 - infraestrutura web

## Camada de Apresentação – MVC 3

A camada de apresentação neste primeiro momento será criada na camada web usando o framework MVC3 da Microsoft.

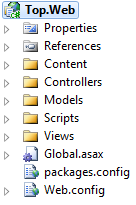


Figura 5 - Estrutura da apresentação Web

O conteúdo de imagens e css são concentrados na pasta “Content”, os controllers, models e views em suas respectivas pastas, e a pasta script armazenam os arquivos javascript usados no projeto.

# Dependências

## ORM

O acesso a dados será feito utilizando um framework de ORM, de forma a não perdermos tempo com a criação de uma camada de acesso a dados consistente e robusta. Para isto usaremos o NHibernate, por ser um framework consolidado e por sua vasta e rica comunidade.

## HelpersMVC

Para acelerar o desenvolvimento, iremos utilizar algumas bibliotecas opensource que estendem o MVC da Microsoft com itens que o framework padrão não implementa.

### MVCContrib

Biblioteca que implementa acesso a extensão de grid para o HtmlHelper (Html.Grid(Model))

### Hicharts.net

Biblioteca que implementa acesso a extensão de gráficos para o HtmlHelper (Html.ChartTypeFor(Model))

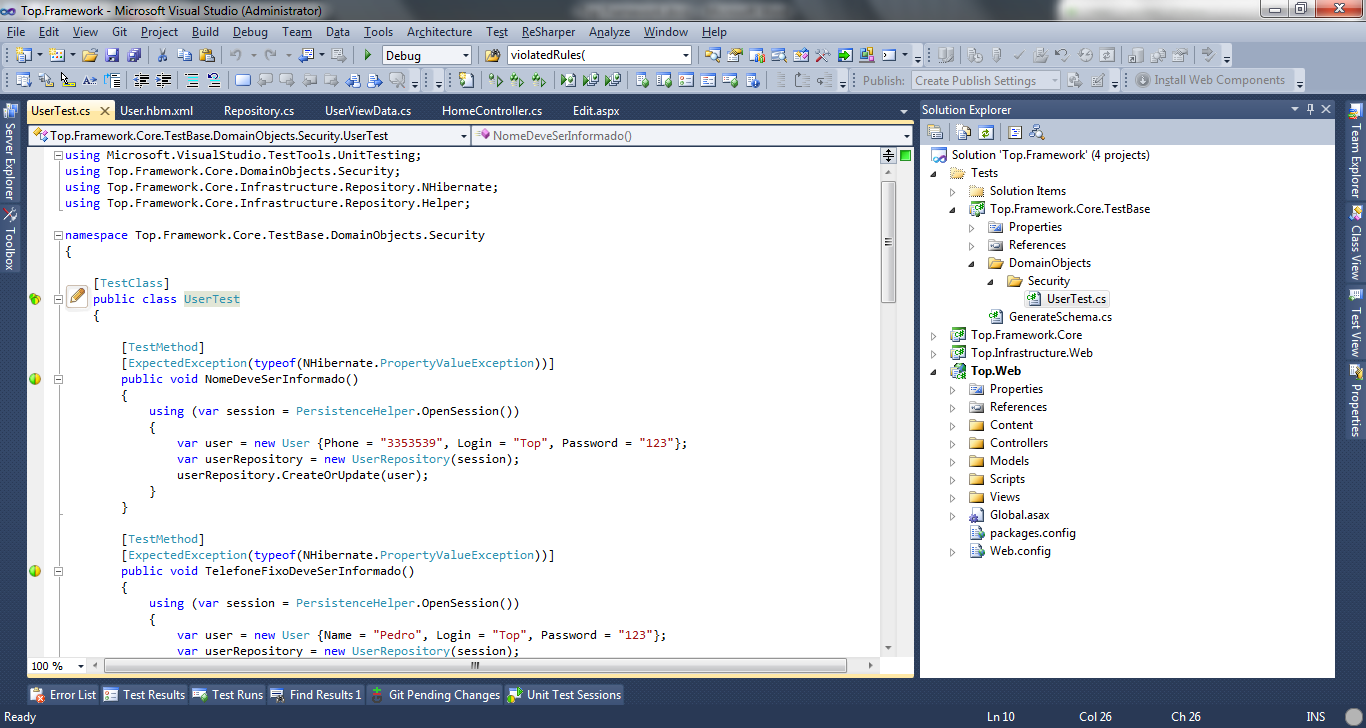
# Passos para criar um cadastro novo

## Criar os testes unitários

Buscando focar no modelo funcional da aplicação, vamos usar o padrão de desenvolvimento de TDD, assim sendo criando primeiramente o teste para a nova entidade, de forma a validar as regras básicas.

* Na “*solution folder*” “*Tests*”, acesso o projeto “*Top.Framework.Core.TestBase*”
* Na pasta “*DomainObjects”*, caso não exista a pasta referente ao módulo que você irá implantar, crie-a.
* Dentro da pasta do módulo, adicione uma nova classe com o nome de sua entidade, seguida da palavra “*Test*” (ex.: *UserTest*)
* A classe deve ser decorada com o atributo: *TestClass*
* Todo método criado deve ser decorado com o atributo: *TestMethod*

Lembre-se de que os dados de um teste não podem depender de outro teste, assim como um teste não pode interferir nos testes de outro método.

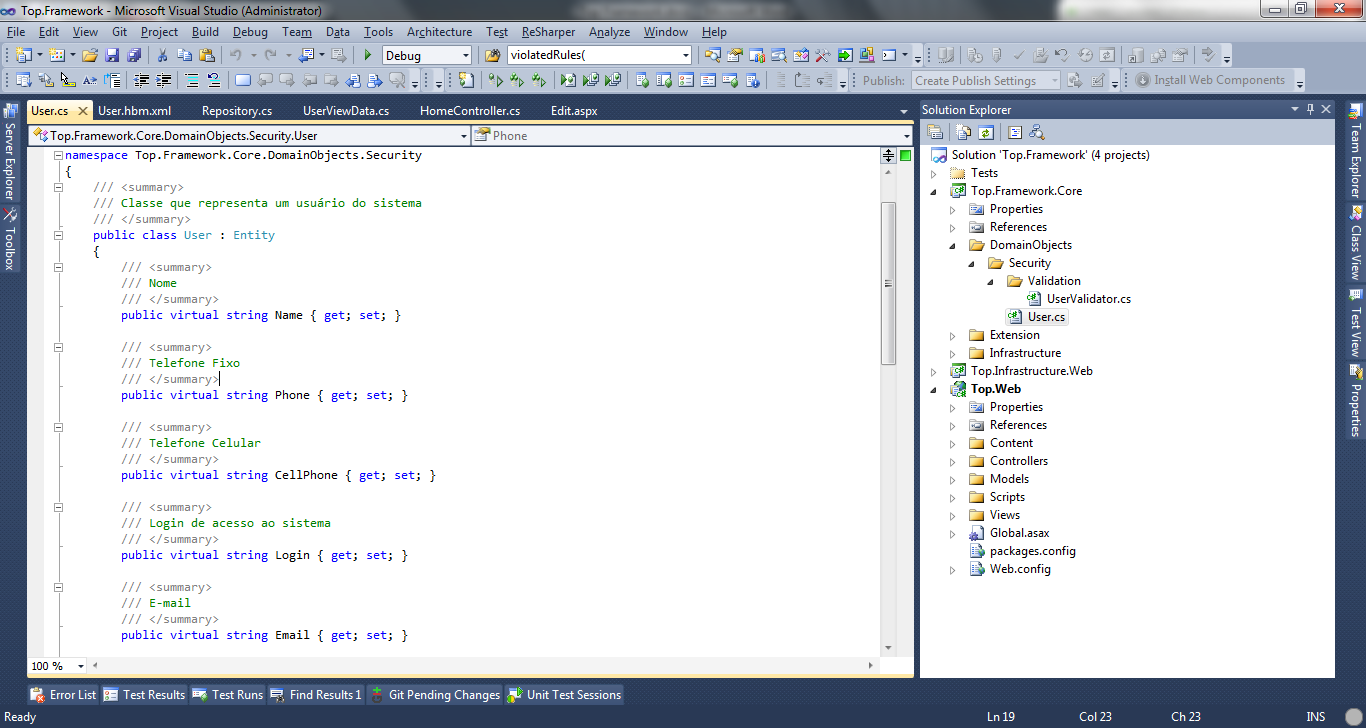


* Para testar se uma exceção irá ocorrer, use o atributo “*ExpectedException*”, seguido do tipo do erro esperado.
* Para testar se o teste está passando, use o objeto “*Assert”* que fornece vários métodos de comparação entre valores para saber se o teste funcionou ou não (ex.: para saber se uma entidade foi salva com sucesso no banco de dados, podemos testar se Id dele ficou diferente de “0” (zero) usando Assert.AreNotEquals(0, EntidadeSalva.Id))

## Criar a entidade

Tendo os testes criados, vamos agora popular os métodos da entidade de forma a fazermos com que os testes passem.

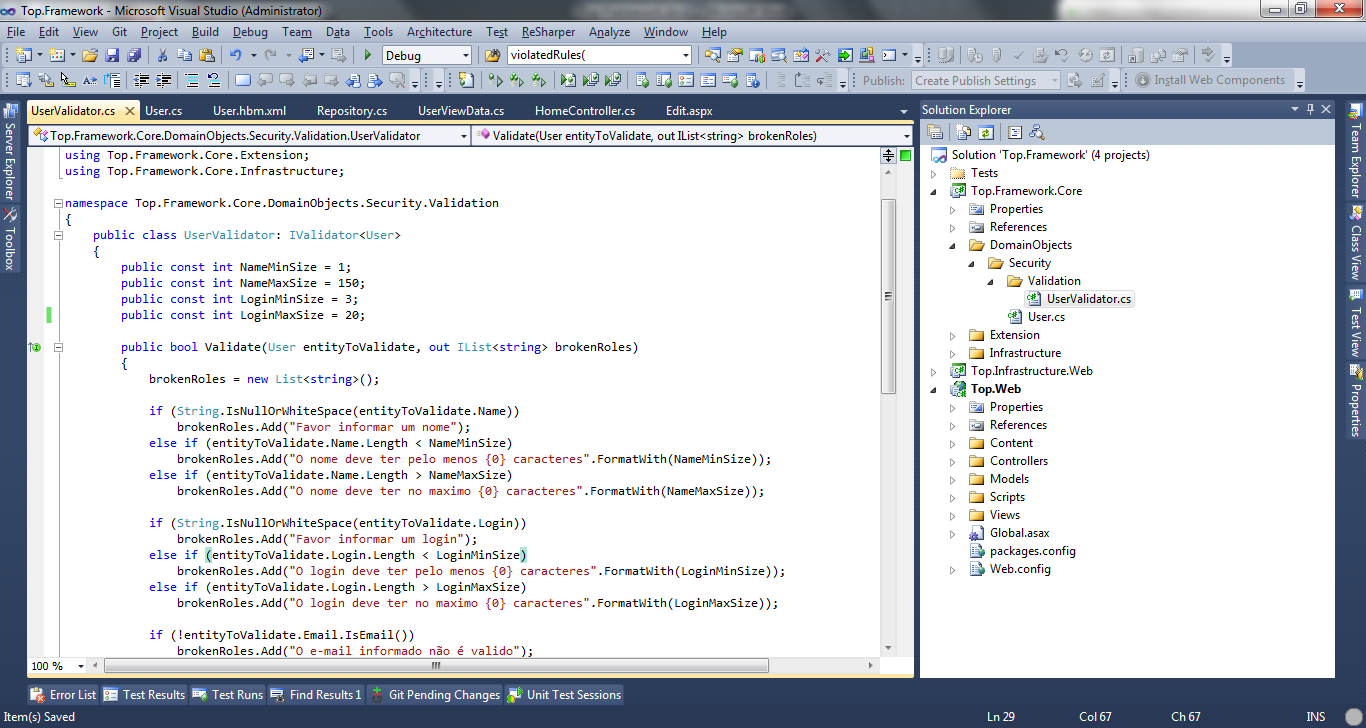
* Toda entidade deve herdar da classe abstrata “Entity” (que já implementa os métodos de Id, Deleted, dentre outros)
* Após herdar da classe “Entity”, implemente as propriedades específicas de sua classe.



## Criar o validador da entidade

De forma complementar ao item 2, vamos criar o validados da entidade de forma a possibilitar os demais testes a passarem e garantir a integridade da entidade.

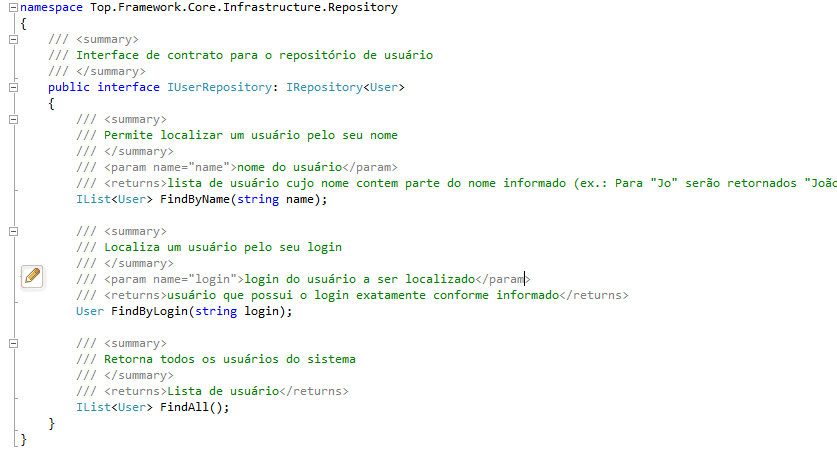
* Ao herdar da classe “Entity”, automaticamente você terá de implementar o método “IsValid()”
* Nele retorne: new NOME\_DA\_ENTIDADE\_Validator().Validate(this, out brokenRoles);
* Na pasta “Validation”, crie a classe que representa seu validador;
* O validador deve implementar a interface “IValidator”, para garantir que todo o sistema irá conseguir chamar o validador;
* O validador é responsável por conhecer as regras que tornam uma entidade válida para o sistema, conhecendo inclusive os tamanhos mínimos e máximos para os campos.



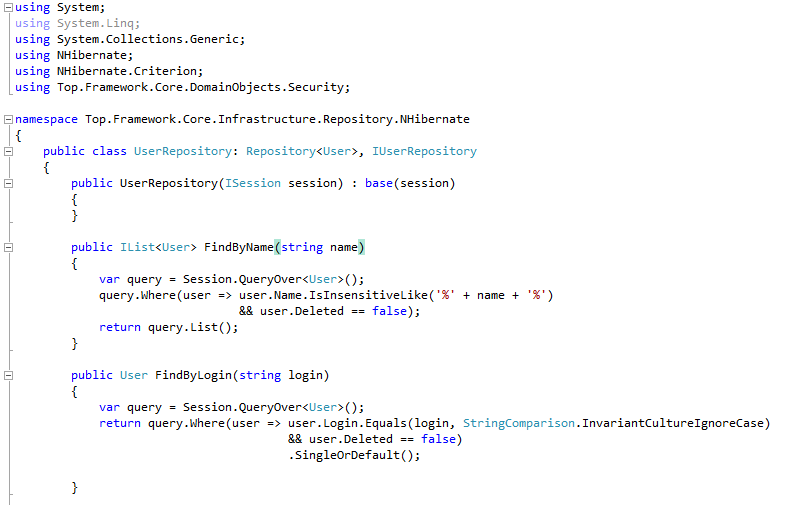
## Criar o repositório da entidade

Tendo a entidade pronta, vamos criar a interface que representa o contrato do repositório e sua implementação concreta.

* A primeira coisa a ser feita é a criação de uma interface que implemente a interface IRepository<T>, onde T é o tipo da entidade a ser persistida no banco de dados.
* Em seguida, implemente na interface os métodos que você irá precisar.



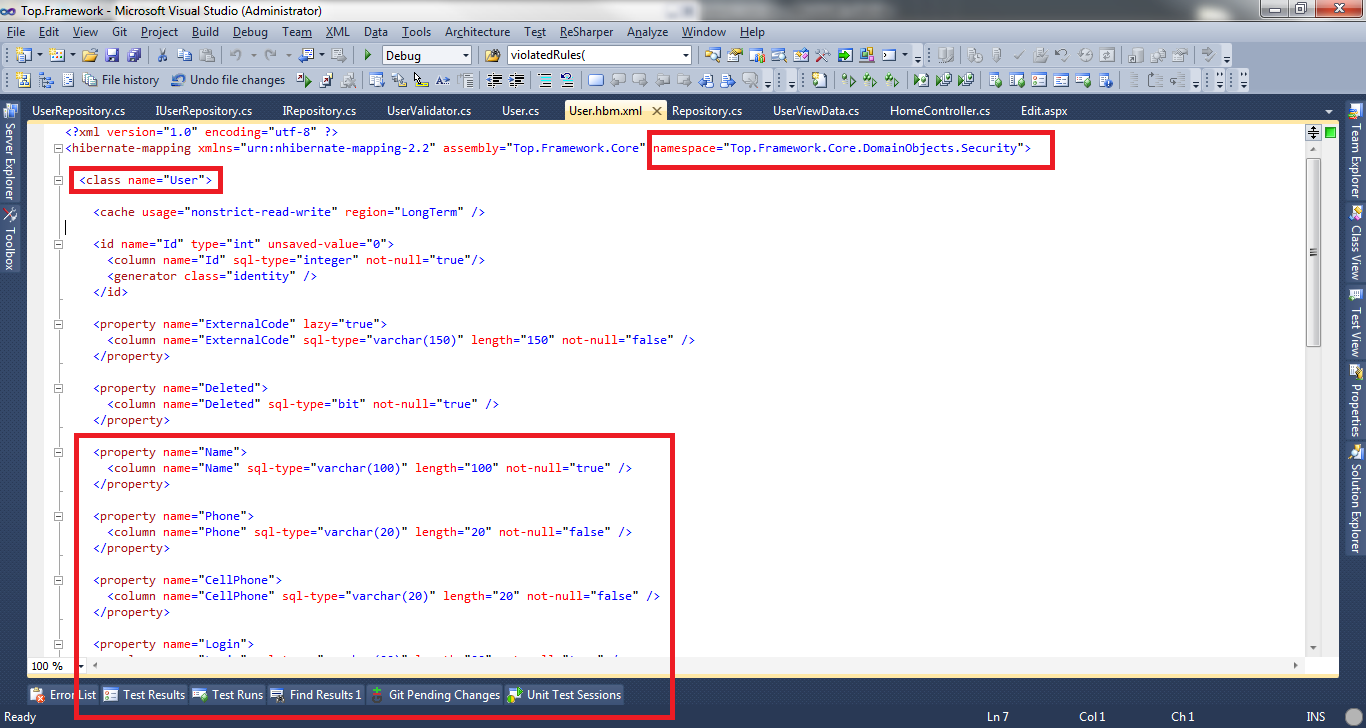
* Após ter feito isto, na pasta NHibernate, crie uma classe concreta que implemente sua interface e que herde de Repository<T>, onde T é o tipo da entidade a ser persistida (exemplo: Para o repositório de usuário, deve-se criar uma Interface IUserRepository e uma Classe UserRepository)
* Ao herdar da classe Repository<T>, os métodos de listagem, criação, atualização, remoção e busca por ID já estarão disponíveis automaticamente para seu repositório.
* Tendo feito isto, popule seus métodos.



## Criar o mapeamento da entidade

Tendo o repositório criado, vamos criar o mapeamento da entidade com o banco de daods.

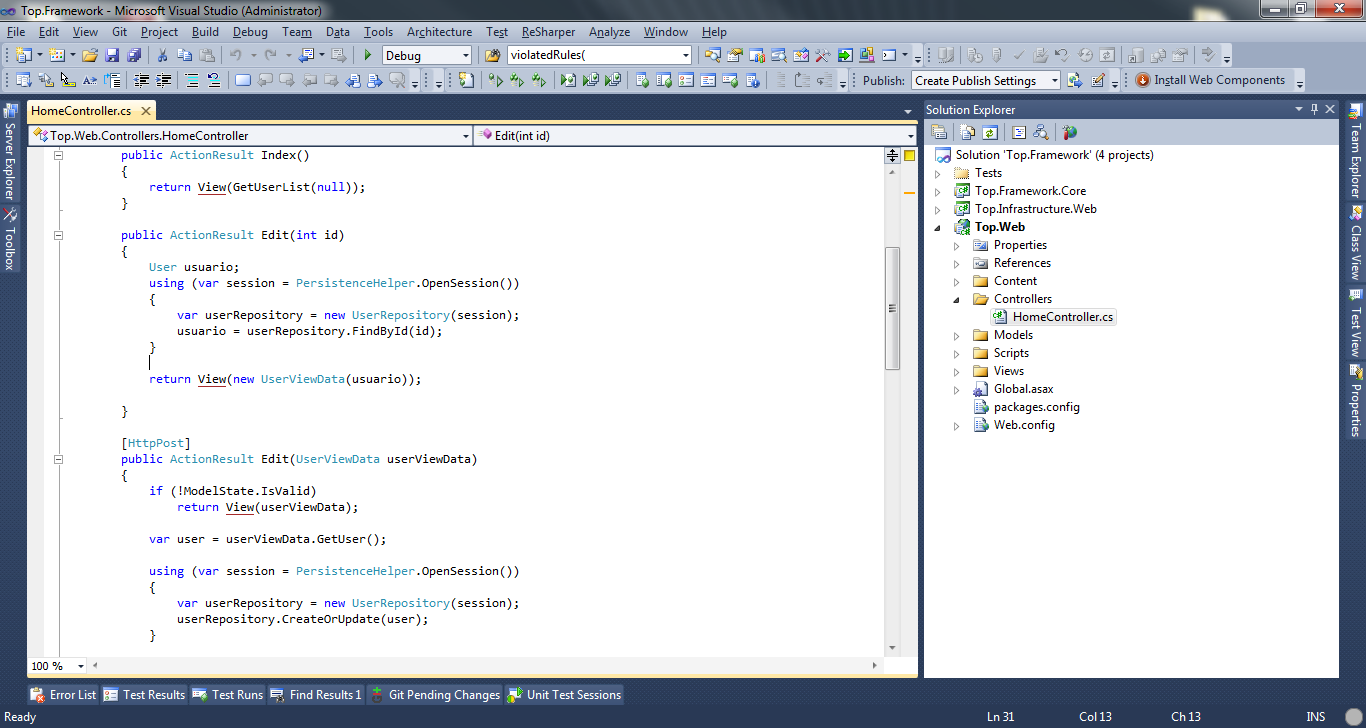
* Na pasta Mapping, dentro de Infrastructure.Repository.NHibernate, copie e cole um mapeamento existente (para ser mais rápido)
* Altere seu nome para NOME\_DA\_SUA\_ENTIDADE.hbm.xml
* É muito importante que o nome do mapeamento tenha o final .hbm.xml
* Verifique se o mapeamento está marcado como EmbededResource na janela de propriedades do visual studio
* Edite o arquivo substituindo o nome do classe e os campos mapeados
* Lembre-se de deixar os campos que são herdados de “Entity” (Id, ExternalCode, Deleted, etc...)



## Criar controller

Agora que somos capazes de criar um nova entidade, gravá-la e recuperá-la do banco de dados, vamos criar os métodos do controller que permitiram que cadastremos, excluamos e listemos os dados salvos.

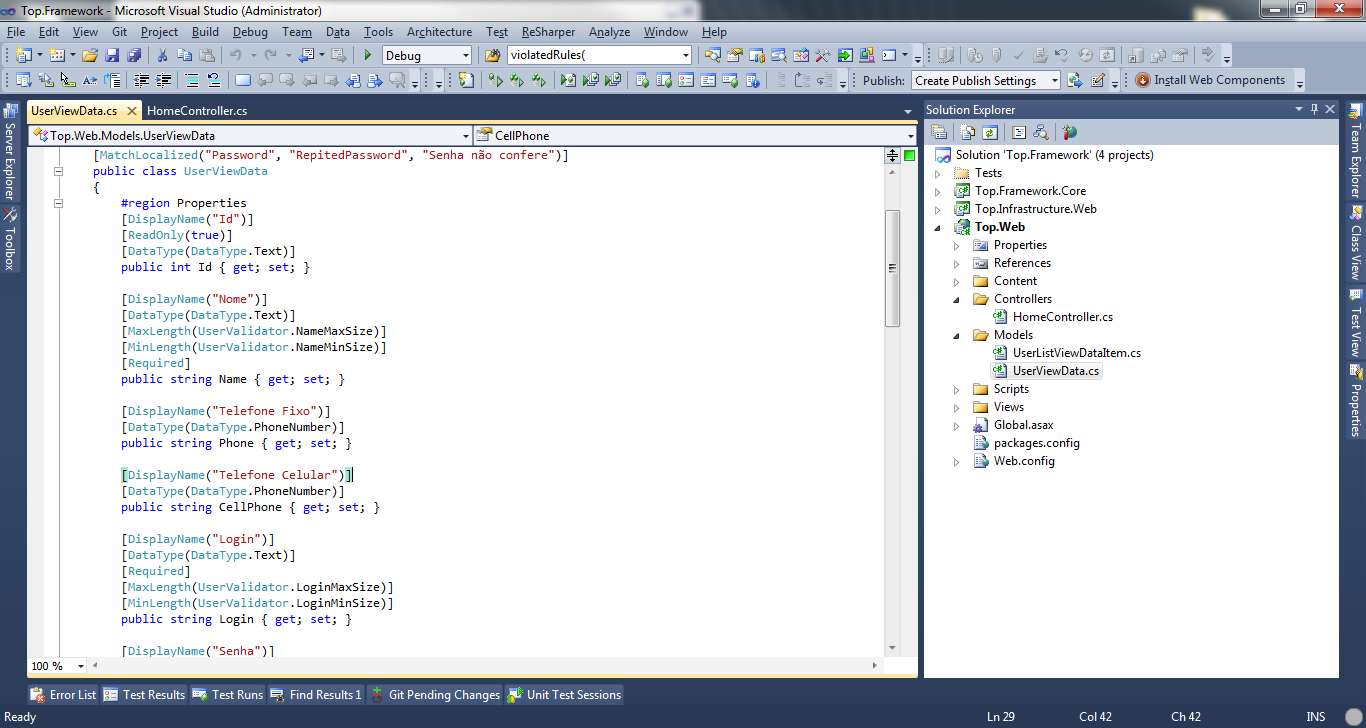
* No projeto web, clique sobre a pasta Controller com o botão direito, em seguida em Add 🡪 Controller
* Na tela que abriu, digite o nome do seu controle, lembrando de manter a palavra chave “Controller” no final (Exemplo: Para cadastro de usuário informe: “UserController”.
* Deixe o Template selecionado como “Empty Controller”
* Edite o arquivo do controller e crie os métodos necessários (Index(), Edit(int id), Detail(int id), Remove(int id))
* Popule os métodos para acessar os dados no repositório e devolver uma lista de “ViewDatas” para a View



## Criar model

Para que o controller consiga trafegar os dados para a interface visual, vamos criar o modelo que representa os dados na interface.

* Crie agora os “ViewDatas” (models) que representam sua tela
* Decore as propriedades com os atributos necessários para que o framework saiba como agir na tela
  + Adicione [Required] aos campos obrigatórios
  + Adicione o [DisplayName] nos campos
  + Adicione as validações de tamanho mínimo e máximo
  + Adicione o tipo de dados



## Criar view

E por fim, vamos criar as visualizações que representam os dados e que vão atuar com os controllers para que o cadastro da nova entidade funcione.

* Na pasta views do projeto web, crie uma pasta com o nome do seu controller (sem a palavra “Controller” ao final) (exemplo: para a view de user, crie uma pasta chamada User)
* Nesta pasta, crie um novo arquivo com o nome de cada Action que você criou no controller (Index.aspx, Edit.aspx, Detail.aspx, etc...)
* Na view de listagem (index.aspx), adicone um grid para a exibição dos dados, usando a linha:

<%: Html.Grid(Model).AutoGenerateColumns()

.Columns(col =>

{

col.For(user => Html.ActionLink("Visualizar", "Detail", new {id = user.Id})).Named("");

col.For(user => Html.ActionLink("Editar", "Edit", new {id = user.Id})).Named("");

col.For(user => Html.ActionLink("Remover", "Remove", new {id = user.Id})).Named("");

}) %>

* Na view de edição e criação, adicione as linhas:

<%: Html.ValidationSummary() %>

<% using (Html.BeginForm()){%>

<%: Html.EditorForModel() %>

<input type="submit" value="Salvar"/>

<% } %>

* Na view de detalhamento, adicione a linha:

<%:Html.DisplayForModel() %>