Macoratti.net ASP .NET MVC - Injeção de dependência com Ninject (revisitado)

No artigo de hoje eu vou mostrar *(mais uma vez)* como podemos usar o container **Ninject** para realizar a injeção de dependência em uma aplicação ASP .NET MVC 5.

Conceitos

*A injeção de dependência(DI) é um padrão de projeto cujo objetivo é manter um baixo acoplamento entre diferentes módulos de um sistema.Nesta solução as dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infraestrutura de software (container) que é responsável por "**injetar**" em cada componente suas dependências declaradas.*

*A Injeção de dependência se relaciona com o padrão Inversão de Controle mas não pode ser considerada um sinônimo deste.(**fonte** http://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_depend%C3%AAncia)*

Mas o que vem a ser acoplamento forte ?

Acoplamento

- Acoplamento é o nível de dependência/conhecimento que pode existir entre as classes;
- Uma classe com acoplamento fraco não é dependente de muitas classes para fazer o que ele tem que fazer;
- Uma classe com acoplamento forte depende de muitas outras classes para fazer o seu serviço;
- Uma classe com acoplamento forte é mais difícil de manter, de entender e de ser reusada;

Coesão

- Coesão é o nível de integralidade interna de uma classe; (Veja o principio da responsabilidade única - SRP)
- A coesão Mede o grau que um classe ou seus métodos fazem sentido, ou seja, quão claro é o entendimento do que a classe ou método possui
- Uma classe com alta coesão possui responsabilidades bem definidas e são difíceis de serem desmembradas em outras classes;
- Uma classe com baixa coesão possui muitas responsabilidades, geralmente que pertencem a outras classes, e podem ser facilmente desmembradas em outras classes;
- Uma classe com baixa coesão é difícil de entender, manter e reusar;

Portanto quanto mais forte o acoplamento e mais baixa a coesão de uma classe mais difícil ela será de entender, manter e ser reusada.

Em suma, a **DI** isola a implementação de um objeto da construção do objeto do qual ele depende.

Podemos implementar a injeção de dependência das seguintes maneiras:

- Injeção via Construtor;
- Injeção via Propriedades (get/set);
- Injeção via Interface;
- **Injeção usando um framework(Spring/Unity/Ninject);**

Neste artigo eu vou mostrar como usar o container **Ninject** para injetar a dependência em uma aplicação ASP .NET MVC de forma bem simples.

Recursos usados :

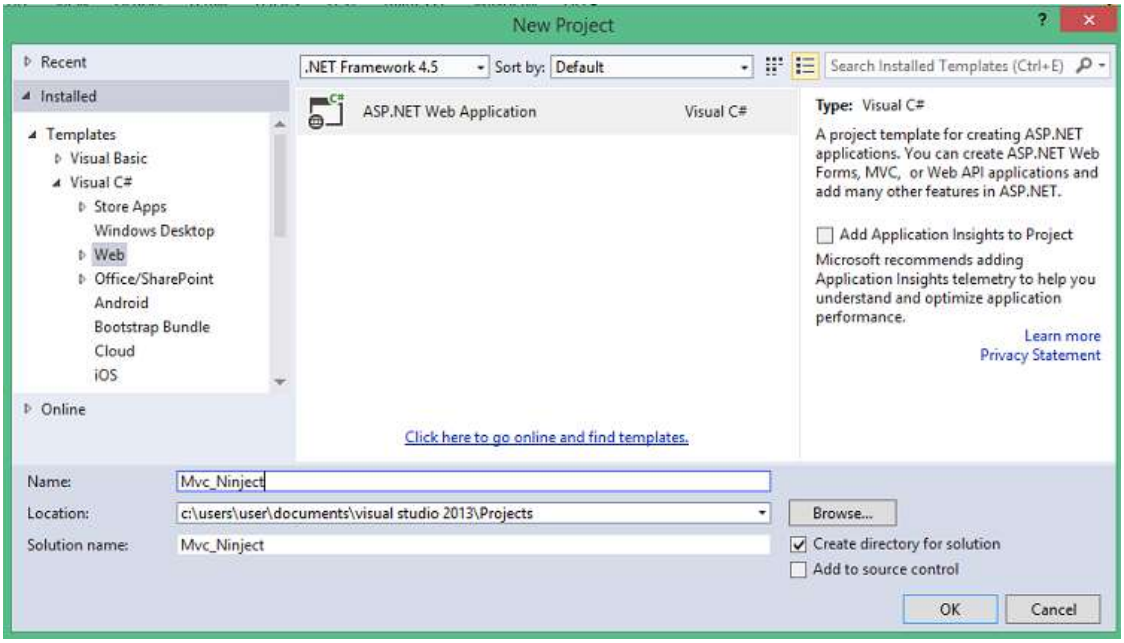
- **Visual Studio Communit 2013**

Criando o projeto ASP .NET MVC no VS Community 2013

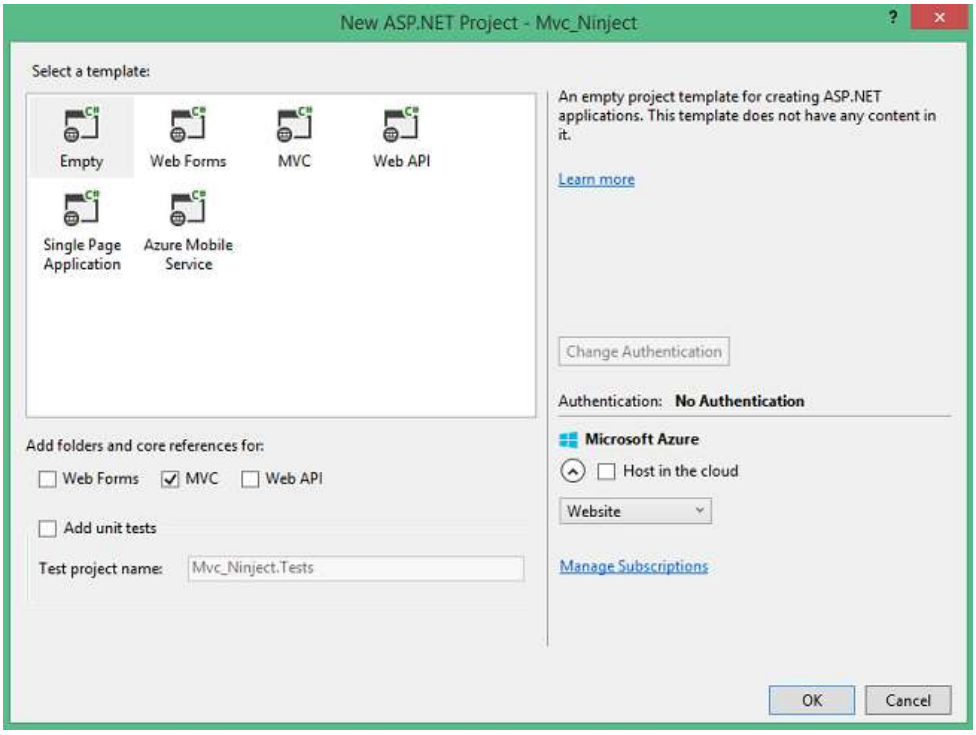
Abra o VS 2013 Community e clique em **New Project;**

A seguir selecione **Visual C# -> Web -> ASP .NET Web Application**

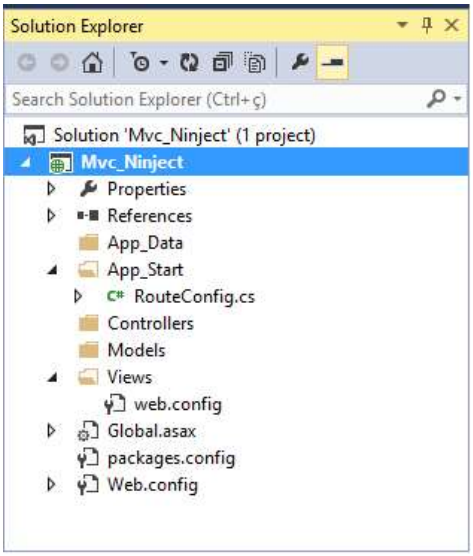
Informe o nome **Mvc_Ninject** e clique no botão **OK**;



A seguir selecione o template Empty, marque MVC e clique no botão OK;



Será criado um projeto ASP .NET MVC vazio com a estrutura básica para criarmos o nosso exemplo.



Neste exemplo iremos criar a interface **ITime** e as classes **Barcelona** e **Juventus** que irão implementar essa interface na pasta **Models**.

Vamos criar também o controlador **HomeController** na pasta **Controllers** e a seguir a view **Index**.

Vamos aproveitar a configuração definida no arquivo **RouteConfig.cs** cujo código é exibido abaixo:

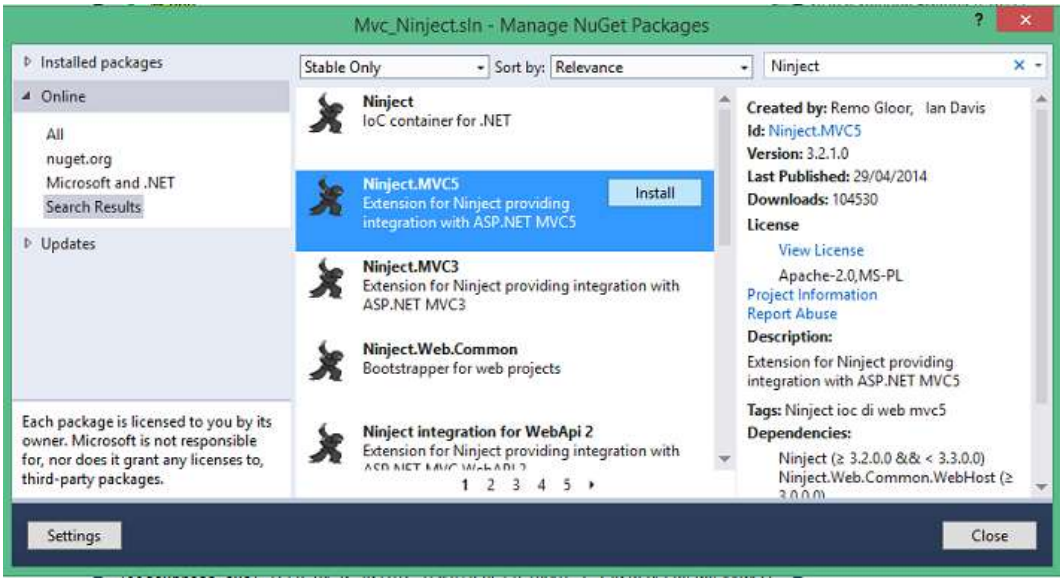
```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

A rota padrão esta definida para acessar o controlador Home a action **Index** que por sua vez irá exibir a view Index.cshtml que iremos criar na pasta **Views**.

Antes de continuar temos que incluir a referência ao Ninject em nosso projeto usando o **Nuget**.

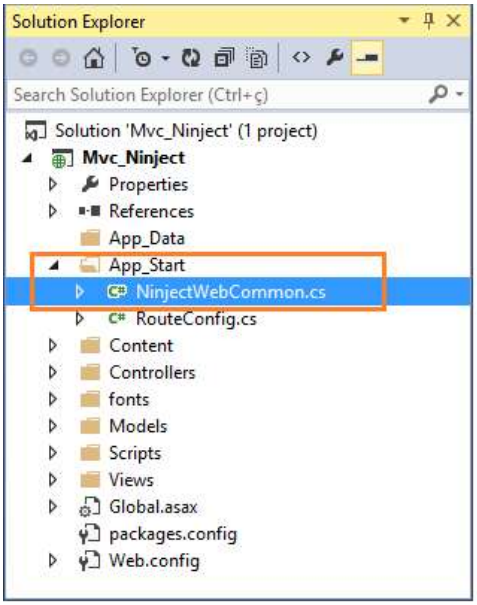
No menu **TOOLS** clique em **Nuget Package Manager -> Manage Nuget Packages for Solution**;

Na janela procure por Ninject e instale o item referente ao **Ninject MVC5** clique no botão **Install**;



Se preferir usar o **Package Manager Console** digite o seguinte comando no console : **Install-Package Ninject.MVC5**

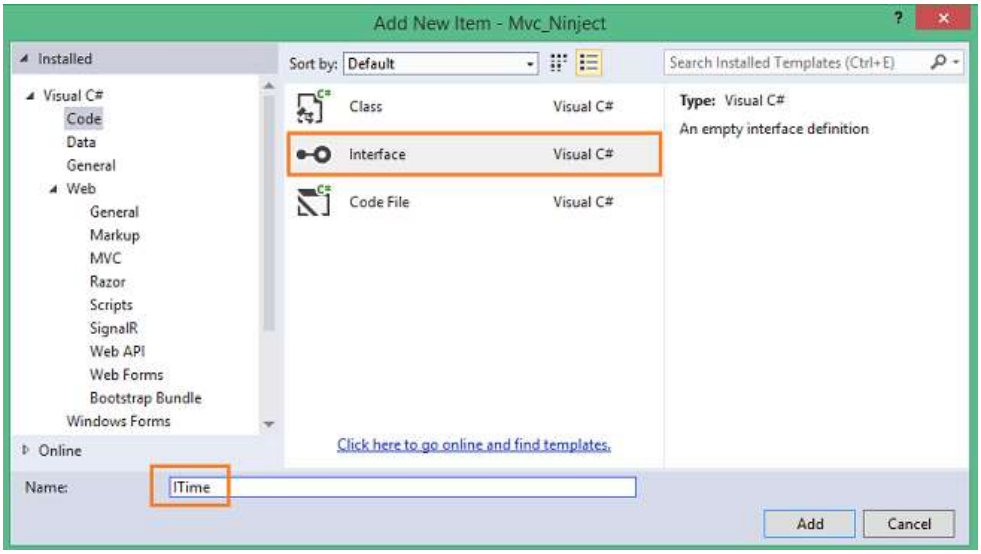
Observe que após a conclusão da instalação do pacote do Ninject foi criado na pasta **App_Start** o arquivo **NinjectWebCommon.cs** que iremos usar para configurar a injeção da dependência em nosso projeto.



Criando a interface e classes concretas na pasta Models

Vamos criar a interface ITime.cs e as classes Barcelona.cs e Juventus.cs na pasta **Models** *(estou fazendo isso para tornar o exemplo mais simples)*

Selecione a pasta **Models** e no menu **PROJECT** clique em **Add New Item,** clique na guia **Code** e no template **Interface** e informe o nome **ITime** e clique no botão **Add**;



A seguir inclua o código abaixo para a interface **ITime**:

```
namespace Mvc_Ninject.Models
{
    public interface ITime
    {
        string GetInformacaoTime();
    }
}
```

Agora, ainda na pasta **Models**, clique no menu **TOOLS** clique em **Add Class** e informe o nome Barcelona e digite o código a seguir para esta classe:

```
namespace Mvc_Ninject.Models
{
    public class Barcelona : ITime
    {
        public string GetInformacaoTime()
        {
            return "Barcelona - Campeão da Liga dos Campeões de 2015";
        }
    }
}
```

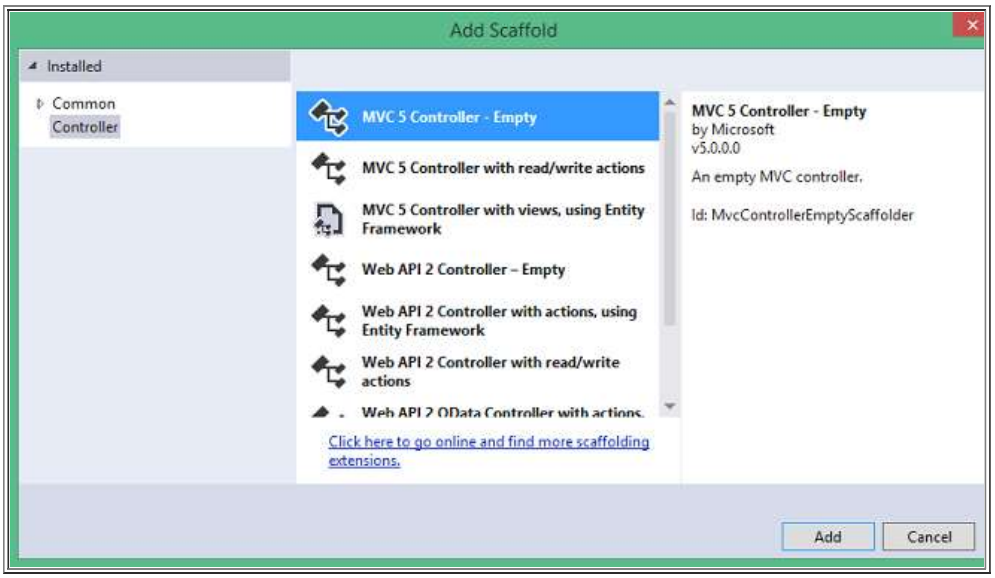
Repita o procedimento anterior e crie a classe **Juventus.cs** na pasta **Models** com o seguinte código:

```
namespace Mvc_Ninject.Models
{
   public class Juventus : ITime
   {
      public string GetInformacaoTime()
      {
         return "Juventus - Vice-Campeão da Liga dos Campeões de 2015";
      }
   }
}
```

Criando o controlador HomeController na pasta ControllersModels

Clique com o botão direito do mouse sobre a pasta **Controllers** e a seguir clique em **Add => Controller**;

A seguir selecione o Scaffold : **MVC 5 Controller Empty** e clique no botão **Add**;



Digite o nome HomeController na janela **Add Controller** e clique no botão **Add**:

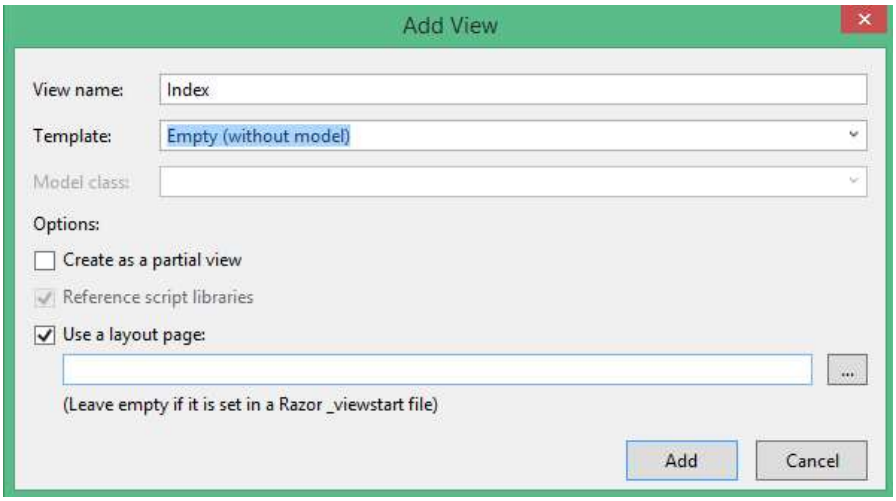
Digite o código abaixo para o **HomeController**:

```
using System.Web.Mvc;
using Mvc_Ninject.Models;

namespace Mvc_Ninject.Controllers
{
   public class HomeController : Controller
   {
      private ITime _time;

      public HomeController(ITime time)
      {
         this._time = time;
      }

      public ActionResult Index()
      {
         string timeInfo = this._time.GetInformacaoTime();
         return View(timeInfo as object);
      }
   }
}
```

Neste código definimos uma variável **_time** do tipo da interface **ITime** e criamos um construtor para o controlador que recebe uma dependência da interface chamada **time** que iremos usar para chamar o método **GetInformacaoTime**() e retornar para nossa View que iremos criar a seguir.

Para criar a view clique com o botão direito do mouse sobre a **Action Index()** e a seguir clique em **Add View**;

Na janela **Add View** informe o nome **Index** para a view e defina o **Template** e demais opções conforme figura abaixo:

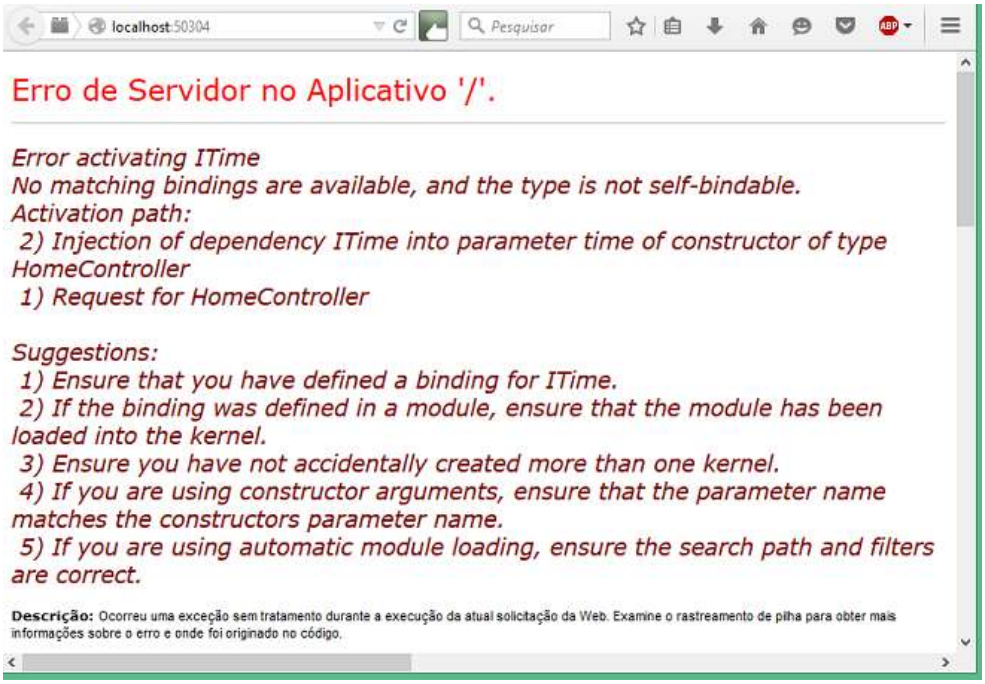


Será criada a view **Index.cshtml** na pasta **Views\Home**.

A seguir digite o código abaixo para a view **Index.cshtml** :

```
@{
    ViewBag.Title = "Index";
}

<h2>Informação do Time</h2>

<h3>@Model</h3>
```

Com isso terminamos o nosso trabalho, e, se executarmos a aplicação iremos obter o seguinte resultado:



Esse erro ocorre porque precisamos injetar a dependência no construtor do controlador **HomeController**:

```
public HomeController(ITime time)
{
    this._time = time;
}
```

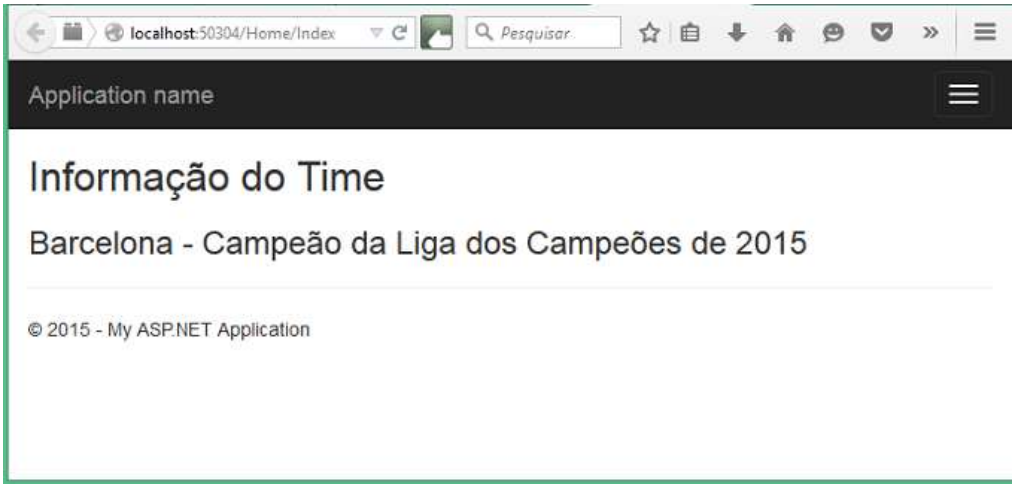
Para fazer isso vamos usar a classe **NinjectWebCommon**.cs na pasta **App_Start.**

Por padrão esse método esta vazio e precisamos configurar o **Ninject** indicando o direcionamento da interface para a classe que a implementa fazendo a vinculação entre elas.

Para isso inclua o código destacado em azul conforme mostrado abaixo:

```
/// <summary>
/// Load your modules or register your services here!
/// </summary>
/// <param name="kernel">The kernel.</param>
private static void RegisterServices(IKernel kernel)
{
   kernel.Bind<ITime>().To<Barcelona>();
}
```

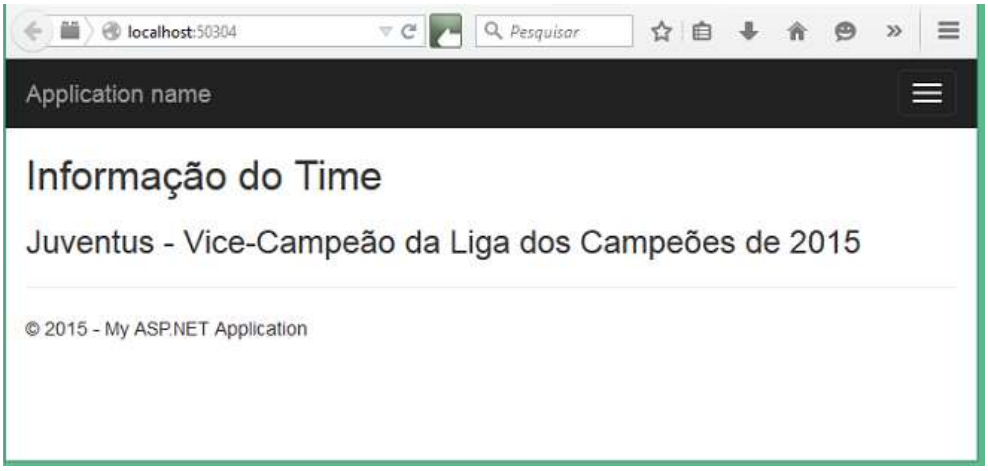
Executando agora o projeto teremos:



Alterando o código para usar a classe concreta **Juventus** teremos :

```
/// <summary>
/// Load your modules or register your services here!
/// </summary>
/// <param name="kernel">The kernel.</param>
private static void RegisterServices(IKernel kernel)
{
   kernel.Bind<ITime>().To<Juventus>();
}
```

E obtemos :



Simples, não é mesmo...

É claro que isso é apenas a '*ponta do iceberg*' do Ninject mas creio que você entendeu o princípio.

Pegue o projeto completo aqui: Mvc_Ninject.zip

Todo aquele que prevarica, e não persevera na doutrina de Cristo, não tem a Deus. Quem persevera na doutrina de Cristo, esse tem tanto ao Pai como ao Filho.
2 João 1:9

Gostou ? Compartilhe no Facebook Compartilhe no Twitter

Referências:

- Seção VB .NET do Site Macoratti.net
- Super DVD .NET - A sua porta de entrada na plataforma .NET
- Super DVD Vídeo Aulas - Vídeo Aula sobre VB .NET, ASP .NET e C#
- Seção C# do site Macoratti.net
- Super DVD C#
- Super DVD Visual Basic
- Curso Básico VB .NET - Vídeo Aulas
- Curso C# Básico - Vídeo Aulas
- macoratti - YouTube
- Macoratti.net | Facebook
- Jose C Macoratti (@macorati) | Twitter
- IoC - Macoratti.net
- .NET - Injeção de dependência com Ninject - Macoratti.net
- O padrão inversão de controle (IoC) - Macoratti.net
- .NET - Inversão de controle e Injeção de dependência para ...

José Carlos Macoratti