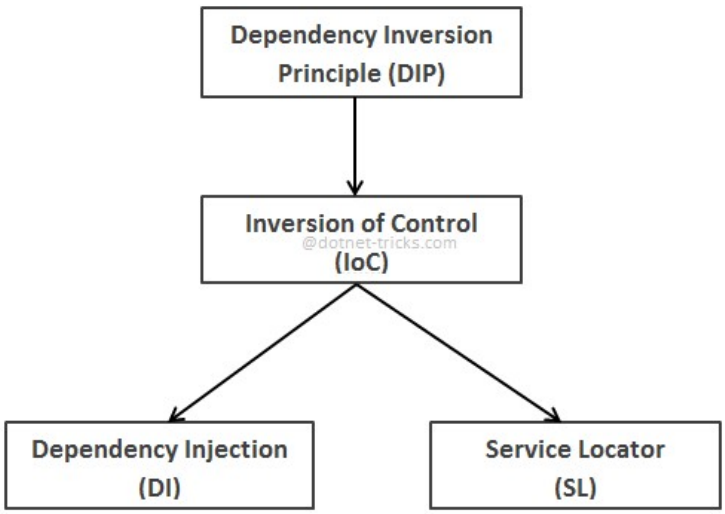
ASP .NET MVC 5 - Implementando a Injeção de dependência com Unity

Neste artigo vou mostrar mais implementação da injeção de dependência desta vez usando o Unity em uma aplicaçãoASP .NET MVC 5.

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.

Assim, podemos ver a DI como uma implementação da *"Inversão de Controle"*, e , a **Inversion of Control (IoC)** diz que os objetos não criam outros objetos nos quais eles confiam para fazer seu trabalho; em vez disso, eles obtêm os objetos de que precisam de uma fonte externa *(por exemplo, um arquivo de configuração XML)*.

Resumindo, o padrão da injeção de dependência é um princípio que nos guia para injetar dependências através da inversão de controle (nossa !!!!)



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);

Hoje vamos usar a ferramenta Unity para implementar a DI em um projeto ASP .NET MVC 5.

Recursos usados:

- VS 2017 Community
- Unity

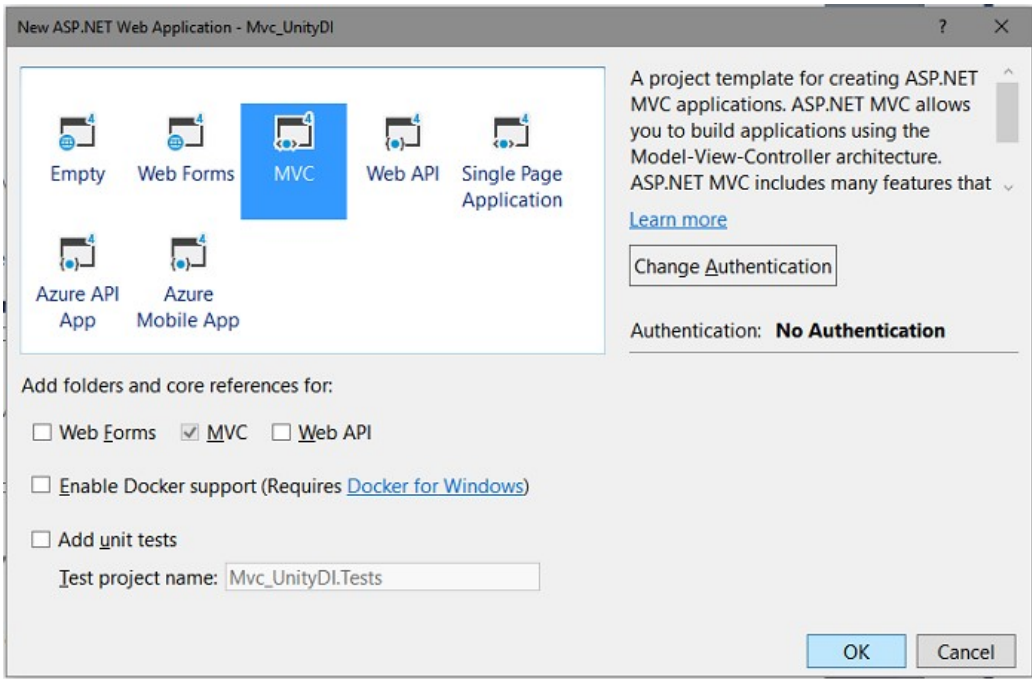
Criando o projeto no VS Community 2017

Abra o VS 2017 Community e clique em **New Project;**

Selecione a linguagem **Visual C# -> Web** e o template **ASP .NET Web Application(.NET Framework);**

Informe o nome da solução como **Mvc_UnityDI** e clique no botão OK;

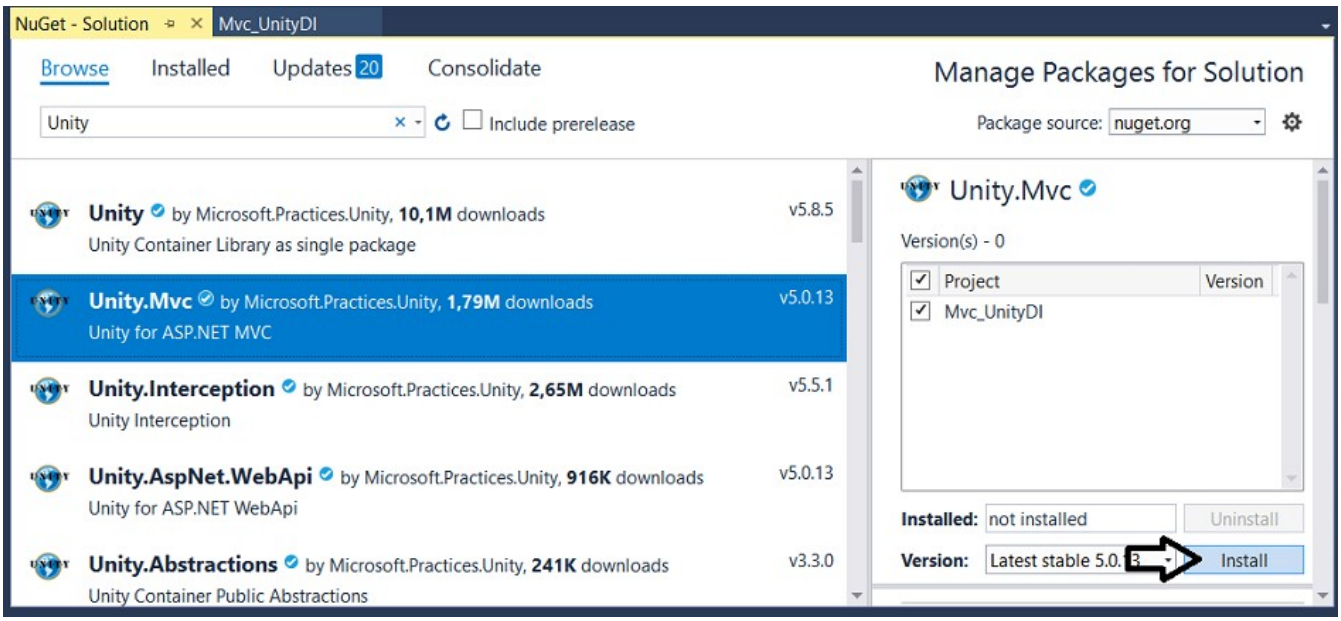
Selecione o template **MVC**, sem autenticação e clique no botão OK;



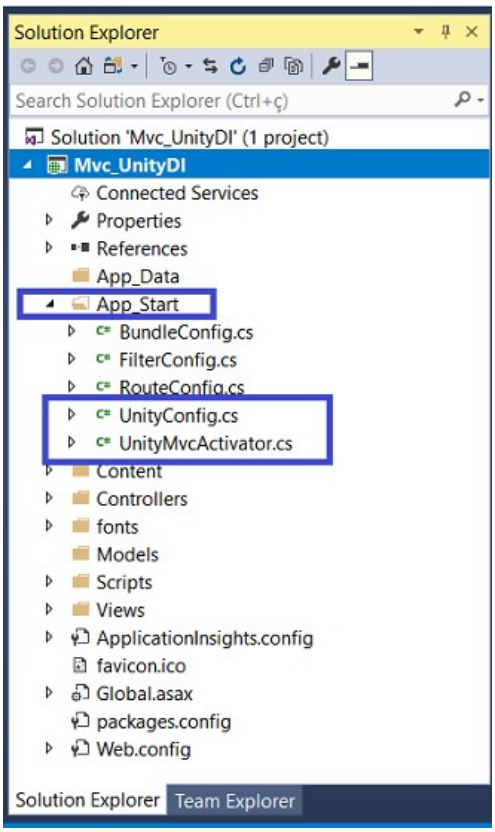
Com o projeto criado vamos incluir o pacote Unity via Nuget.

No menu **Tools** clique em **Nuget Package Manager** e a seguir em **Manger Nuget Packages for Solution;**

Clique no link **Browse** e selecione o pacote **Unity.Mvc,** marque o projeto e clique em **Install:**



Após a instalação você deverá ver na pasta **App_Start** os arquivos : **UnityConfig.cs e UnityMvcActivator.cs**



Criando um modelo de domínio e um repositório de dados

Vamos criar um modelo de domínio representando pela classe **Usuario** e um repositório que será acessado pelo controlador da nossa aplicação.

Nota: *Eu vou criar o modelo de domínio e o repositório na mesma pasta porque esta aplicação é apenas uma demonstração e para simplificar o projeto.*

Crie uma pasta chamada **Repository** no projeto e inclua a classe **Usuario** nesta pasta com o código abaixo:

```
public class Usuario
   {
      public string Nome { get; set; }
      public string Senha { get; set; }
      public string Email { get; set; }
   }
```

A seguir crie uma interface chamada **IUsuarioRepository** nesta pasta com o código a seguir:

```
using System.Collections.Generic;

namespace Mvc_UnityDI.Repository
{
   public interface IUsuarioRepository
   {
      IEnumerable<Usuario> GetAll();
      Usuario Get(int id);
      Usuario Add(Usuario item);
      bool Update(Usuario item);
      bool Delete(int id);
   }
}
```

Agora vamos criar a classe **UsuarioRepository** que implementa a interface acima na mesma pasta:

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;

namespace Mvc_UnityDI.Repository
{
   public class UsuarioRepository : IUsuarioRepository
   {
      private List<Usuario> usuarios = new List<Usuario>();
      private int Id = 1;

      public UsuarioRepository()
      {
         // incluindo alguns usuários para demo
         Add(new Usuario { ID=1, Nome = "Macoratti", Email = "macoratti@teste.com", Senha = "numsey@13" });
         Add(new Usuario { ID=2, Nome = "Jefferson", Email = "jeff@teste.com", Senha = "ytedg6543" });
         Add(new Usuario { ID=3, Nome = "Miriam", Email = "miriam3@teste.com", Senha = "#5496dskj" });
      }

      public Usuario Add(Usuario item)
      {
         if (item == null)
         {
            throw new ArgumentNullException(nameof(item));
         }

         item.ID = Id++;
         usuarios.Add(item);
         return item;
      }

      public bool Delete(int id)
      {
         usuarios.RemoveAll(p => p.ID == id);
         return true;
      }

      public Usuario Get(int id)
      {
         return usuarios.FirstOrDefault(x => x.ID == id);
      }

      public IEnumerable<Usuario> GetAll()
      {
         return usuarios;
      }

      public bool Update(Usuario item)
      {
         if (item == null)
         {
            throw new ArgumentNullException(nameof(item));
         }

         int index = usuarios.FindIndex(p => p.ID == item.ID);
         if (index == -1)
         {
            return false;
         }
         usuarios.RemoveAt(index);
         usuarios.Add(item);
         return true;
      }
   }
}
```

Registrando o repositorio no container DI

Agora vamos registrar a interface e sua implementação no contâiner DI do Unity.

Abra o arquivo **UnityConfig.cs** e inclua um método chamado **RegistraComponentes()** com o código a seguir:

```
        public static void RegistraComponentes()
        {
            var container = new UnityContainer();
            container.RegisterType<IUsuarioRepository, UsuarioRepository>();
            DependencyResolver.SetResolver(new UnityDependencyResolver(container));
        }
```

Agora abra o arquivo **Global.asax** e inclua a linha de código para chamar este método :

```
public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);

            UnityConfig.RegistraComponentes();
        }
    }
```

Agora podemos usar a injeção de independência para o nosso repositório no controlador.

Realizando a Injeção de dependência no controlador

Primeiro vamos criar um controlador chamado **UsuariosController** para exibir os usuários em nosso projeto.

Clique com o botão direito sobre a pasta **Controllers** e a seguir clique em **Add ->Controller** e selecione o template **MVC 5 Controller - Empty;**

informe o nome **UsuariosController**;

A seguir inclua o código abaixo neste controlador:

```
using Mvc_UnityDI.Repository;
using System.Web.Mvc;

namespace Mvc_UnityDI.Controllers
{
    public class UsuariosController : Controller
    {
        readonly IUsuarioRepository usuarioRepositorio;
        public UsuariosController(IUsuarioRepository repository)
        {
            this.usuarioRepositorio = repository;
        }

        // GET: Usuarios
        public ActionResult Index()
        {
            var data = usuarioRepositorio.GetAll();
            return View(data);
        }
    }
}
```

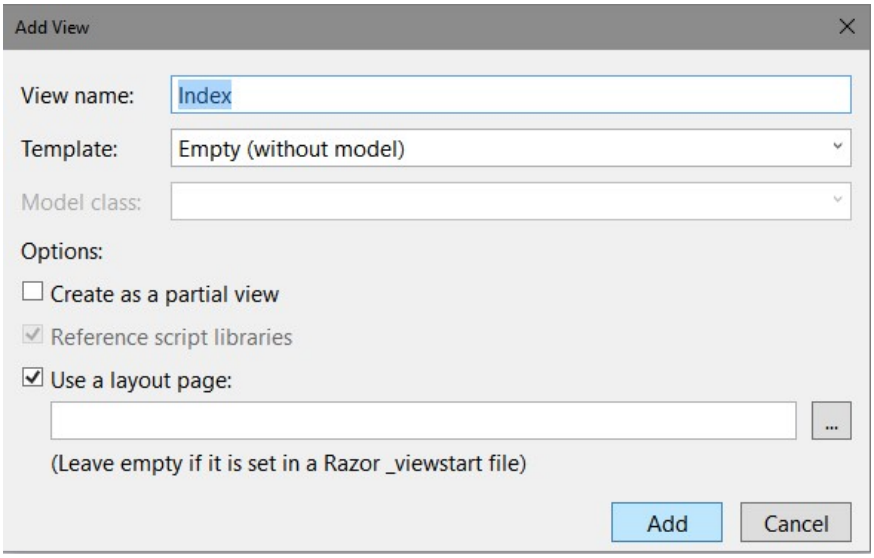
No código destacado em azul estamos fazendo a injeção de dependência do repositório no construtor do controlador e obtendo assim uma instância do nosso repositório.

A seguir estamos retornando todos os usuário usando o método **GetAll**().

Só falta criar a View **Index** para exibir os usuários.

Clique com o botão direito sobre o método Index e a seguir em **Add -> View;**

Na janela Add View defina as opções conforme mostra a figura e clique **Add**;



Defina o código abaixo no arquivo **Index.cshtml** da pasta **Views/Usuarios :**

```
@model IEnumerable<Mvc_UnityDI.Repository.Usuario>
@{
   ViewBag.Title = "Usuários";
   Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>Relação de Usuários</h2>
<hr />
<table class="table">
   <tr>
      <th>
         @Html.DisplayNameFor(model => model.Nome)
      </th>
      <th>
         @Html.DisplayNameFor(model => model.Email)
      </th>
      <th>
         @Html.DisplayNameFor(model => model.Senha)
      </th>
      <th></th>
   </tr>

   @foreach (var item in Model)
   {
      <tr>
         <td>
            @Html.DisplayFor(modelItem => item.Nome)
         </td>
         <td>
            @Html.DisplayFor(modelItem => item.Email)
         </td>
         <td>
            @Html.DisplayFor(modelItem => item.Senha)
         </td>
      </tr>
   }
</table>
```

Agora é só alegria...

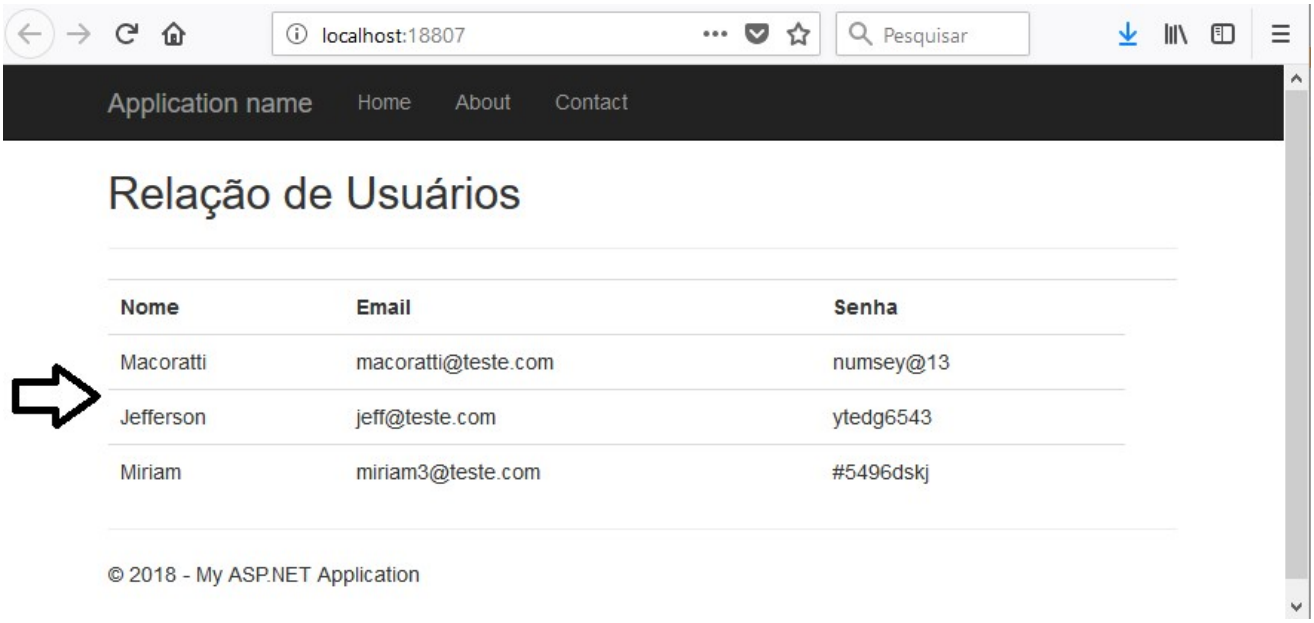
Altere a rota no arquivo **RouteConfig** para exibir o controller **Usuarios** e a sua view **Index** :

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

         routes.MapRoute(
```

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

Executando o projeto teremos o seguinte resultado:



Vimos assim a injeção de dependência em ação usando o Unity;

Pegue o projeto aqui : Mvc_UnityDI.zip *(sem as referências)*

'(Disse Jesus) - Nisto todos conhecerão que sois meus discípulos, se vos amardes uns aos outros.'
João 13:35

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
- ASP .NET MVC - Filtrando registros com Dropdownlist - Macoratti
- ASP .NET MVC - Movendo itens entre dois DropDownList ... - Macoratti
- Usando DropDownList em páginas ASP.NET - Macoratti
- ASP .NET MVC - Exibindo uma lista de itens selecionáveis - Macoratti
- ASP .NET MVC 5 - Dropwdownlist revistado - Macoratti
- ASP .NET MVC 3 - Gerando PDF - Macoratti
- ASP .NET MVC - Baixando arquivos - Macoratti.net
- ASP .NET MVC 3 - Usando o HTML Helper WebGrid - Macoratti
- ASP .NET MVC 5 - Exibindo dados no formato JSON - Macoratti
- ASP .NET MVC 4 - Exibindo dados JSON usando Knockout - Macoratti
- ASP .NET - Filtrando com WebGrid - Macoratti.net
- ASP .NET MVC - WebGrid : Crud com Entity Framework - I - Macoratti
- ASP .NET MVC - Vinculando um Calendar jQuery em um WebGrid
- ASP .NET MVC 5 - Usando o StructureMap para injeção de ... - Macoratti
- ASP .NET MVC 5 - Aplicando boas práticas em seu projeto - Macoratti
- ASP .NET MVC - Injeção de Dependência e repositório Mock - Macoratti

---

José Carlos Macoratti