

we are developers
Eventos
(https://eventos.imasters.com.br/)

Fórum iMasters
(https://forum.imasters.com.br/)

Developer Store
(https://imasters.shop/)

7Masters
(https://setemasters.imasters.com.br/)

Revista Impressa
(https://issuu.com/imasters/docs/imasters_26_v6_isuu)

Certificações
(http://certificacao.imasters.com.br)

B
(http://ww

POWERED BY:
(HTTPS://DEV.ELO.COM.BR)

(HTTP://IMPULSO.NETWORK/?UTM_SOURCE=IMASTERS&UTM_MEDIUM=SITE&UTM_CAMPAIGN=HEADER)

f
(https://www.facebook.com/PortaliMasters)

t
(https://twitter.com/iMasters)

G+
(https://plus.google.com/+imasters)

ps://imasters.com.br/mobile)

Front(https://imasters.com.br/front-end)

DevSecOps(https://imasters.com.br/devsecops)

Design(https://imasters.com.br/design-ux)

Data(https://imasters.com.br/data)

APIs e Microserviços (https://imas

PATROCINADORES:

BACK-END

DialHost

FIAP

impulso

KINGHOST

locaweb

School of Net

uol host

24 AGO, 2018

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

f
(https://www.facebook.com/sharer?u=https://imasters.com.br/back-end/asp-net-mvc-5-implementando-injecao-de-dependencia-com-unity)

t
(https://twitter.com/share?url=https://imasters.com.br/back-end/asp-net-mvc-5-implementando-injecao-de-dependencia-com-unity)


410 visualizações

G+
(https://plus.google.com/share?url=https://imasters.com.br/back-end/asp-net-mvc-5-implementando-injecao-de-dependencia-com-unity)

in
(https://www.linkedin.com/shareArticle?url=https://imasters.com.br/back-end/asp-net-mvc-5-implementando-injecao-de-dependencia-com-unity)

COMPARTILHEI

JOSÉ CARLOS MACORATTI
(HTTPS://IMASTERS.COM.BR/PERFIL/JOSE_CARLOS_MACORATTI)
Tem 545 artigos publicados com 858362 visualizações desde 2011



PUBLICIDADE



JOSÉ CARLOS MACORATTI (HTTPS://IMASTERS.COM.BR/PERFIL/JOSE_CARLOS_MACORATTI)

545

é referência em Visual Basic no Brasil e autor dos livros "Aprenda Rápido: ASP" e "ASP, ADO e Banco de Dados na Internet". Mantenedor do site macoratti.net.

LEIA MAIS (HTTPS://IMASTERS.COM.BR/PERFIL/JOSE_CARLOS_MACORATTI)

6 MAR, 2019
C# – Usando a biblioteca Math.NET (.NET Core) (https://imasters.com.br/back-end/c-usando-biblioteca-math-net-net-core)

4 MAR, 2019
C# – Calculando palavras Fibonacci (https://imasters.com.br/back-end/c-calculando-palavras-fibonacci)

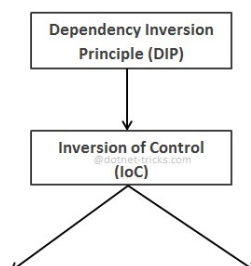
26 FEV, 2019
ASP .NET Core: apresentando e usando o Logging – Parte 01 (https://imasters.com.br/back-end/asp-net-core-apresentando-e-usando-o-logging-parte-01)

Neste artigo vou mostrar mais da implementação da injeção de dependência; desta vez, usando o Unity em uma aplicação ASP .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!).



Dependency Injection
(DI)

Service Locator
(SL)

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)

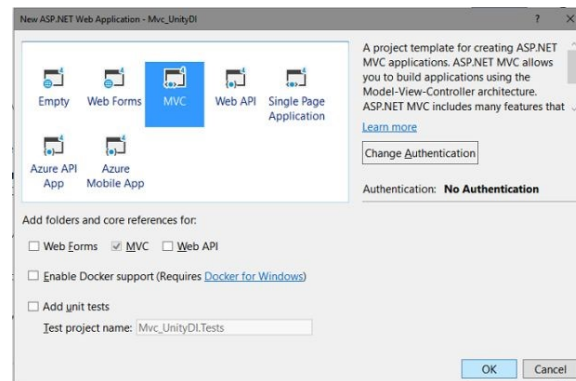
Hoje vamos usar a ferramenta Unity para implementar a DI em um projeto ASP .NET MVC 5.

Recursos usados

- VS 2017 Community
- Unity

Criando o projeto no VS Community 2017

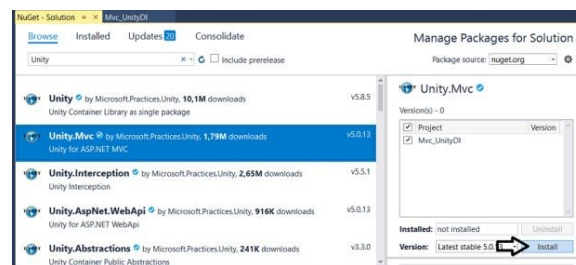
- Abra o VS 2017 Community e clique em "New Project";
- Selecione a linguagem Visual C# > Web e o template ASP .NET Web Application (.NET Framework);
- Informe o nome da solução como Mvc_UnityDI e clique no botão "OK";
- Selecione o template MVC, sem autenticação e clique no botão "OK";



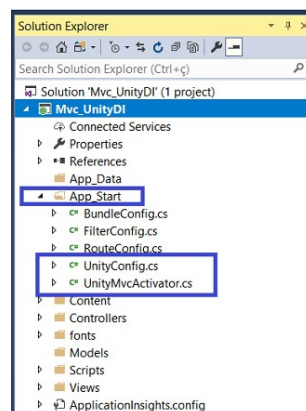
Com o projeto criado, vamos incluir o pacote Unity via Nuget.

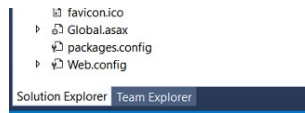
No menu Tools, clique em "Nuget Package Manager" e a seguir em Manger Nuget Packages for Solution;

Clique no link "Browse", 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 representado pela classe **Usuario** e um repositório que será acessado pelo controlador da nossa aplicação.

Nota: Criei o modelo de domínio e o repositório na mesma pasta porque esta aplicação é apenas uma demonstração e também para simplificar o projeto.

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

```
1 public class Usuario
2 {
3     public string Nome { get; set; }
4     public string Senha { get; set; }
5     public string Email { get; set; }
6 }
```

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

```
1 using System.Collections.Generic;
2 namespace Mvc_UnityDI.Repository
3 {
4     public interface IUsuarioRepository
5     {
6         IEnumerable<Usuario> GetAll();
7         Usuario Get(int id);
8         Usuario Add(Usuario item);
9         bool Update(Usuario item);
10        bool Delete(int id);
11    }
12 }
```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 namespace Mvc_UnityDI.Repository
5 {
6     public class UsuarioRepository : IUsuarioRepository
7     {
8         private List<Usuario> usuarios = new List<Usuario>();
9         private int Id = 1;
10        public UsuarioRepository()
11        {
12            // incluindo alguns usuários para demo
13            Add(new Usuario { ID=1, Nome = "Macoratti", Email = "macoratti@teste.com", Senha = "numsey@13" });
14            Add(new Usuario { ID=2, Nome = "Jefferson", Email = "jeffg@teste.com", Senha = "ytedg6543" });
15            Add(new Usuario { ID=3, Nome = "Miriam", Email = "miriam3@teste.com", Senha = "#5496dskj" });
16        }
17        public Usuario Add(Usuario item)
18        {
19            if (item == null)
20            {
21                throw new ArgumentNullException(nameof(item));
22            }
23            item.ID = Id++;
24            usuarios.Add(item);
25            return item;
26        }
27        public bool Delete(int id)
28        {
29            usuarios.RemoveAll(p => p.ID == id);
30            return true;
31        }
32        public Usuario Get(int id)
33        {
34            return usuarios.FirstOrDefault(x => x.ID == id);
35        }
36        public IEnumerable<Usuario> GetAll()
37        {
38            return usuarios;
39        }
40        public bool Update(Usuario item)
41        {
42            if (item == null)
43            {
44                throw new ArgumentNullException(nameof(item));
45            }
46            int index = usuarios.FindIndex(p => p.ID == item.ID);
47            if (index == -1)
48            {
49                return false;
50            }
51            usuarios.RemoveAt(index);
52            usuarios.Add(item);
53            return true;
54        }
55    }
56 }

```

Registrando o repositório no container DI

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

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

```

1 public static void RegistraComponentes()
2 {
3     var container = new UnityContainer();
4     container.RegisterType<IUsuarioRepository, UsuarioRepository>();
5     DependencyResolver.SetResolver(new UnityDependencyResolver(container));
6 }

```

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

```

1 public class MvcApplication : System.Web.HttpApplication
2 {
3     protected void Application_Start()
4     {
5         AreaRegistration.RegisterAllAreas();
6         FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
7         RouteConfig.RegisterRoutes(RouteTable.Routes);
8         BundleConfig.RegisterBundles(BundleTable.Bundles);
9         UnityConfig.RegistraComponentes();
10    }
11 }

```

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:

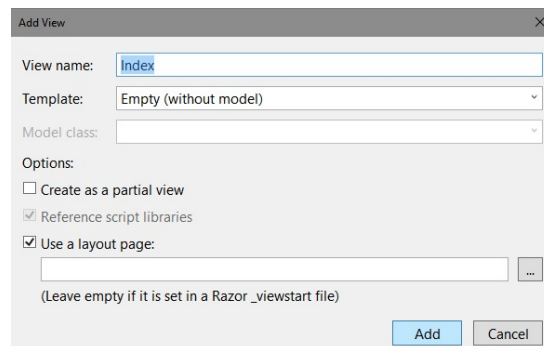
```
1 using Mvc_UnityDI.Repository;
2 using System.Web.Mvc;
3 namespace Mvc_UnityDI.Controllers
4 {
5     public class UsuariosController : Controller
6     {
7         readonly IUseruarioRepository usuarioRepositorio;
8         public UsuariosController(IUsuarioRepository repository)
9         {
10             this.usuarioRepositorio = repository;
11         }
12         // GET: Usuarios
13         public ActionResult Index()
14         {
15             var data = usuarioRepositorio.GetAll();
16             return View(data);
17         }
18     }
19 }
```

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

A seguir retornaremos 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**:

```

1 | @model IEnumerable<Mvc_UnityDI.Repository.Usuario>
2 | @{
3 |     ViewBag.Title = "Usuários";
4 |     Layout = "~/Views/Shared/_Layout.cshtml";
5 | }
6 | <h2>Relação de Usuários</h2>
7 | <hr />
8 | <table class="table">
9 |     <tr>
10 |         <th>
11 |             @Html.DisplayNameFor(model => model.Nome)
12 |         </th>
13 |         <th>
14 |             @Html.DisplayNameFor(model => model.Email)
15 |         </th>
16 |         <th>
17 |             @Html.DisplayNameFor(model => model.Senha)
18 |         </th>
19 |     </tr>
20 |     @foreach (var item in Model)
21 |     {
22 |         <tr>
23 |             <td>
24 |                 @Html.DisplayFor(modelItem => item.Nome)
25 |             </td>
26 |             <td>
27 |                 @Html.DisplayFor(modelItem => item.Email)
28 |             </td>
29 |             <td>
30 |                 @Html.DisplayFor(modelItem => item.Senha)
31 |             </td>
32 |         </tr>
33 |     }
34 | </table>
35 |

```

Agora é só alegria!

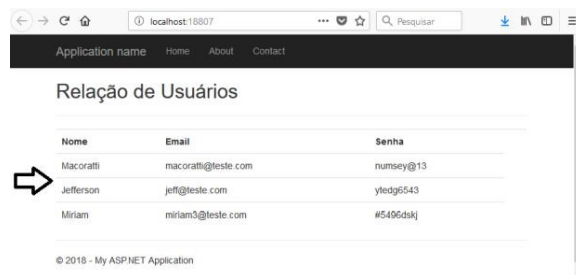
Altere a rota no arquivo **RouteConfig** para exibir o controller **Usuarios** e a sua view **Index**:

```

1 | public static void RegisterRoutes(RouteCollection routes)
2 | {
3 |     routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
4 |     routes.MapRoute(
5 |         name: "Default",
6 |         url: "{controller}/{action}/{id}",
7 |         defaults: new { controller = "Usuarios", action = "Index", id = UrlParameter.Optional }
8 |     );
9 | }

```

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](http://www.macoratti.net/18/06/Mvc_UnityDI.zip) (http://www.macoratti.net/18/06/Mvc_UnityDI.zip) (sem as referências).



De 0 a 10, o quanto você recomendaria este artigo para um amigo?

0 1 2 3 4 5 6 7 8 9 10

ARTIGOS PUBLICADOS POR ESTE AUTOR

JOSÉ CARLOS MACORATTI ([HTTPS://IMASTERS.COM.BR/PERFIL/JOSE_CARLOS_MACORATTI](https://imasters.com.br/perfil/jose_carlos_macoratti))
6 MAR, 2019



C# – Usando a biblioteca Math.NET (.NET Core) (<https://imasters.com.br/back-end/c-usando-biblioteca-math-net-net-core>)



ASSINE NOSSA Newsletter Fique em dia com as novidades do iMasters! Assine nossa newsletter e receba conteúdos especiais curados por nossa equipe

 Qual é o seu e-mail?

ASSINAR



- [SOBRE O IMASTERS](#)
([HTTPS://IMASTERS.COM.BR /P/SOBRE-O-IMASTERS](https://imasters.com.br/p/sobre-o-imasters))
- [POLÍTICA DE PRIVACIDADE](#)
([HTTPS://IMASTERS.COM.BR /P/POLITICA-DE-PRIVACIDADE](https://imasters.com.br/p/politica-de-privacidade))
- [FALE CONOSCO](#)
([HTTPS://IMASTERS.COM.BR /FALE-CONOSCO/](https://imasters.com.br/fale-conosco))
- [QUERO SER AUTOR](#)
([HTTPS://IMASTERS.COM.BR /P/QUERO-SER-AUTOR](https://imasters.com.br/p/quero-ser-autor))
- [FÓRUM](#)
([HTTPS://FORUM.IMASTERS.COM.BR/](https://forum.imasters.com.br/))
- [7MASTERS](#)
([HTTPS://SETEMASTERS.IMASTERS.COM.BR/](https://setemasters.imasters.com.br/))
- [AGENDA](#)
([HTTPS://IMASTERS.COM.BR /AGENDA/](https://imasters.com.br/agenda/))