AutoMapper

GUIA PRÁTICO DE UTILIZAÇÃO DO AUTOMAPPER EM C#

Renata Werneck de Macedo

01

INTRODUÇÃO

Mapeamento de Objetos com AutoMapper no .NET

No desenvolvimento de aplicações modernas, a organização do código e a eficiência no tratamento de dados são fatores cruciais. É nesse contexto que o **AutoMapper** surge como uma ferramenta indispensável para facilitar o mapeamento de objetos, promovendo produtividade, simplicidade e a adoção de boas práticas de desenvolvimento.

O AutoMapper é uma biblioteca popular em C# usada para mapear automaticamente objetos de um tipo para outro. Ele é bastante útil em cenários onde você precisa converter entidades de domínio em DTOs (Data Transfer Objects) e vice-versa, reduzindo o código repetitivo de mapeamento

Exemplo simples: Imagine que você tem uma entidade Usuario e precisa transformá-la em um UsuarioDto. Sem o AutoMapper, seria necessário copiar manualmente cada propriedade. Com ele, tudo é automatizado

Por Que o AutoMapper é Importante?

O AutoMapper vai além de economizar tempo: ele promove um código mais mantenível e escalável. Quando utilizado em conjunto com arquiteturas bem definidas, como Clean Architecture, ele contribui diretamente para a separação de responsabilidades.

Os principais benefícios incluem:

- Redução de código repetitivo: Menos linhas de código manual para mapeamento.
- Aumento da produtividade: Mais foco no desenvolvimento de funcionalidades.
- Redução de erros: Mapeamento automático diminui erros de lógica.
- Facilidade de manutenção: Configurações centralizadas facilitam ajustes futuros.

A Importância da Separação de Camadas

A separação de camadas é um dos pilares de uma aplicação robusta. Dividir responsabilidades entre camadas distintas — como **Domínio, Aplicação, Infraestrutura e Apresentação** — melhora a organização do projeto e facilita a evolução do sistema.

Como o AutoMapper se encaixa nesse conceito?

- Isolamento da lógica de transformação: O AutoMapper permite que o mapeamento de dados seja responsabilidade exclusiva de uma camada específica, geralmente na aplicação ou infraestrutura.
- Respeita os princípios SOLID: Ele garante que a lógica de mapeamento não esteja espalhada pelo código, respeitando o princípio da responsabilidade única.
- Promove desacoplamento: Usar o AutoMapper em perfis específicos facilita a troca ou evolução de camadas sem impacto significativo nas demais.

02

UTILIZAÇÃO

Configurando o AutoMapper no Projeto

Para começar, adicione o AutoMapper ao seu projeto usando o NuGet Package Manager:

```
Install-Package AutoMapper
Install-Package AutoMapper.Extensions.Microsoft.DependencyInjection
```

Depois, configure no Startup.cs ou Program.cs:

```
services.AddAutoMapper(typeof(Startup));
```

Crie uma classe de perfil para definir os mapeamentos:

```
public class UsuarioProfile : Profile {
    public UsuarioProfile() {
        CreateMap<Usuario, UsuarioDto>();
    }
}
```

Mapeamento Simples: Transformando Objetos

Use o AutoMapper para mapear propriedades entre objetos de forma direta.

```
public class Usuario {
    public string Nome { get; set; }
    public int Idade { get; set; }
}

public class UsuarioDto {
    public string Nome { get; set; }
    public int Idade { get; set; }
}

// Mapeamento
var usuarioDto = _mapper.Map<UsuarioDto>(usuario);
```

Mapeamento Customizado: Personalizando Propriedades

Você pode controlar como as propriedades são mapeadas.

```
CreateMap<Pedido, PedidoDto>()
    .ForMember(dest ⇒ dest.Total, opt ⇒
        opt.MapFrom(src ⇒ src.Itens.Sum(i ⇒ i.Preco * i.Quantidade)))
    .ForMember(dest ⇒ dest.NomeCliente, opt ⇒
        opt.MapFrom(src ⇒ src.Cliente.Nome));
```

Mapeando Coleções: Trabalhando com Listas

O AutoMapper facilita o mapeamento de coleções.

```
var pedidosDto = _mapper.Map<List<PedidoDto>>(pedidos);
```

Se você tem uma lista de Pedido, ela será transformada automaticamente em uma lista de PedidoDto.

Mapeamento Reverso: Indo e Voltando

Com o AutoMapper, você pode configurar mapeamentos bidirecionais facilmente.

```
CreateMap<Usuario, UsuarioDto>().ReverseMap();
```

Agora, você pode mapear de Usuario para UsuarioDto e vice-versa.

Ignorando Propriedades: Excluindo Campos do Mapeamento

Se uma propriedade não deve ser mapeada, você pode ignorá-la.

```
CreateMap<Produto, ProdutoDto>()
    .ForMember(dest ⇒ dest.CodigoInterno, opt ⇒ opt.Ignore());
```

Validando Configurações: Teste Seu Mapeamento

Garanta que seus mapeamentos estão corretos utilizando validações automáticas.

```
var config = new MapperConfiguration
    (cfg ⇒ cfg.AddProfile<UsuarioProfile>());
config.AssertConfigurationIsValid();
```

Se houver algo errado no mapeamento, você será avisado.

Injeção de Dependência: Mantendo o Código Limpo

O AutoMapper funciona perfeitamente com IoC. Basta injetar o IMapper onde necessário.

```
public class UsuarioService {
    private readonly IMapper _mapper;

    public UsuarioService(IMapper mapper) {
        _mapper = mapper;
    }

    public UsuarioDto ObterUsuarioDto(Usuario usuario) {
        return _mapper.Map<UsuarioDto>(usuario);
    }
}
```

Boas Práticas no Uso do AutoMapper

1. Centralize Perfis: Crie uma pasta exclusiva para classes de perfil.
2. Evite Configurações Globais: Configure os mapeamentos apenas em perfis dedicados.
3. Teste Sempre: Use AssertConfigurationIsValid para validar seus mapeamentos.

03

CONCLUSÃO

OBRIGADA POR LER ATÉ AQUI

O AutoMapper é uma ferramenta indispensável para tornar o mapeamento de objetos mais rápido, eficiente e confiável. Com os exemplos e boas práticas apresentados, você já está preparado para integrar essa biblioteca nos seus projetos e otimizar seu desenvolvimento.

https://github.com/rwmacedo

rwmacedo