Injeção de Dependência

.NET Injeção de Dependência

Uma introdução aos principais conceitos da injeção de dependência na plataforma .NET

https://www.youtube.com/playlist?list=PLJ4k1IC8GhW0jE6SxHeb6CFhO2MLbAoKa

Injeção de dependência na prática com Dúvida

Ao fazer a seguinte injeção de dependência:

```
public class EventsController
{
    private readonly IEventService _eventService;
    public EventsController(IEventService eventService)
    {
        _eventService = eventService;
    }
}
```

Da seguinte classe:

```
    public class EventService : IEventService
    {
        private readonly IGeneralRepository _generalRepository;
        private readonly IEventRepository _eventRepository;
        private readonly IMapper _mapper;

        public EventService(IGeneralRepository generalRepository
        {
            _generalRepository = generalRepository;
            _eventRepository = eventRepository;
            _mapper = mapper;
        }

}
```

Em que momento estou dando a minha classe EventService os parâmetros de instância/dependências que foram especificadas no construtor dela?

Percebi que eu não estou instanciando a classe, mas mesmo assim estou utilizando ela e suas dependências, poderia explicar?

Resposta

O processo pelo qual você fornece as dependências necessárias para a classe `EventService` ocorre durante a configuração do mecanismo de injeção de dependência (DI).

Que é normalmente realizado no momento da inicialização da aplicação, geralmente no método `ConfigureServices` da classe `Startup` em uma aplicação

ASP.NET Core.

Quando você registra os serviços no contêiner de injeção de dependência usando o método `AddScoped`, `AddTransient` ou `AddSingleton` no método `ConfigureServices`, você está dizendo ao DI como criar instâncias dessas dependências quando elas forem necessárias.

Por exemplo, suponha que você tenha o seguinte método `ConfigureServices` na sua classe `Startup`:

```
public void ConfigureServices(IServiceCollection services)
{
    // Registro dos serviços no contêiner de DI
    services.AddScoped<IEventService, EventService>();
    services.AddScoped<IGeneralRepository, GeneralRepository>
();
    services.AddScoped<IEventRepository, EventRepository>();
    services.AddSingleton<IMapper, Mapper>();
}
```

Aqui, você está dizendo ao DI que, sempre que uma classe precisar de uma instância de `IEventService`, `IGeneralRepository`, `IEventRepository` ou `IMapper`, ele deve criar uma instância da classe correspondente (`EventService`, `GeneralRepository`, `EventRepository` ou `Mapper`, respectivamente) e fornecê-la.

Portanto, nessa situação:

```
    public class EventsController
    {
        private readonly IEventService _eventService;
        public EventsController(IEventService eventService)
```

```
        {
            _eventService = eventService;
        }
    }
```

O DI resolve a dependência da classe `EventsController`, ele verifica que `EventsController` precisa de um `IEventService`.

Como você registrou `EventService` como implementação de `IEventService` durante a configuração do DI: `services.AddScoped<IEventService, EventService>();`

O DI cria uma instância de `EventService` e a passa como argumento para o construtor de `EventsController`.

```
    public class EventService : IEventService
    {
        private readonly IGeneralRepository _generalRepository;
        private readonly IEventRepository _eventRepository;
        private readonly IMapper _mapper;

        public EventService(IGeneralRepository generalRepository
                                                    IEventRe
                                                    IMapper
        {
            _generalRepository = generalRepository;
            _eventRepository = eventRepository;
            _mapper = mapper;
        }

    }
```

Da mesma forma, quando o DI cria uma instância de `EventService`, ele verifica que `EventService` precisa de instâncias de `IGeneralRepository`, `IEventRepository` e `IMapper`.

Como você registrou implementações correspondentes para essas interfaces durante a configuração do DI:

```
    services.AddScoped<IGeneralRepository, GeneralRepository>();
    services.AddScoped<IEventRepository, EventRepository>();
    services.AddSingleton<IMapper, Mapper>();
```

Ele cria instâncias dessas classes e as passa como argumentos para o construtor de `EventService`.

Portanto, o DI resolve automaticamente as dependências e fornece as instâncias necessárias no momento da criação dos objetos, sem que você precise se preocupar em instanciar manualmente cada uma delas.

Isso é uma das vantagens da injeção de dependência: ela permite que você desacople suas classes e torne seu código mais flexível e testável.