Resumo em português do vídeo "Curso Angular #39: Injeção de Dependência (DI) + como usar um serviço em um componente" por Loiane Groner

O que é injeção de dependência?

Injeção de dependência é um conceito de software que permite que uma classe obtenha as dependências de que precisa de outra classe, sem precisar instanciá-las manualmente. Isso torna o código mais modular, flexível e fácil de testar.

Como usar injeção de dependência no Angular 2?

No Angular 2, a injeção de dependência é feita através de construtores. Para injetar um serviço em um componente, siga estas etapas:

1. **Decore a classe do serviço com o decorator** `@Injectable()`**.** Isso indica ao Angular que a classe pode ser injetada em outras classes.
2. **Adicione o serviço ao módulo do componente.** Isso faz com que o Angular torne a instância do serviço disponível para o componente.
3. **Injete o serviço no construtor do componente.** Isso permite que o componente acesse a instância do serviço.

Exemplo de injeção de dependência no Angular 2

No vídeo, Loiane demonstra como injetar um serviço chamado `CursosService` em um componente chamado `CursosComponent`. O `CursosService` é responsável por obter dados de cursos de uma API. O `CursosComponent` usa esses dados para exibir uma lista de cursos na tela.

Benefícios da injeção de dependência

A injeção de dependência oferece vários benefícios, incluindo:

- **Código mais modular:** As classes não precisam de ser tightly coupled umas às outras, o que torna o código mais fácil de manter e reutilizar.
- **Código mais flexível:** É mais fácil alterar o código sem afetar outras partes do aplicativo.
- **Código mais testável:** É mais fácil testar as classes em isolamento, pois elas não dependem de outras classes para serem instanciadas.

Recursos adicionais

- Documentação do Angular sobre injeção de dependência
- Vídeo da Loiane Groner sobre injeção de dependência no Angular 2

Exemplos Práticos de Injeção de Dependência no Angular

A Injeção de Dependência (DI) é um mecanismo fundamental no Angular que permite que os componentes obtenham as dependências de que precisam de forma organizada e flexível. Isso facilita a criação de código modular, reutilizável e testável.

1. Injetando um Serviço para Obter Dados:

Imagine um componente `ProdutosComponent` que precisa exibir uma lista de produtos de um serviço API. A DI torna isso possível:

Criando o Serviço:

TypeScript

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class ProdutosService {
  constructor(private http: HttpClient) {}
```

```
  obterProdutos() {
    return this.http.get('/api/produtos');
  }
}
```

Usando o Serviço no Componente:

TypeScript

```
import { Component, OnInit, Inject } from '@angular/core';
import { ProdutosService } from '../produtos.service';

@Component({
  selector: 'app-produtos',
  templateUrl: './produtos.component.html',
  styleUrls: ['./produtos.component.css']
})
export class ProdutosComponent implements OnInit {
  produtos: any[];

  constructor(
    @Inject(ProdutosService) private produtosService: ProdutosService
  ) {}

  ngOnInit() {
    this.produtosService.obterProdutos().subscribe(produtos => {
      this.produtos = produtos;
    });
  }
}
```

Explicação:

- O `ProdutosService` é injetado no construtor do `ProdutosComponent` usando `@Inject(ProdutosService)`.
- O `ngOnInit` obtém os produtos do serviço e os armazena na propriedade `produtos`.

2. Injetando um Serviço para Compartilhar Funcionalidade:

Imagine um serviço `LoggerService` que registra mensagens no console. Diversos componentes podem se beneficiar desse serviço:

Criando o Serviço:

TypeScript

```typescript
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class LoggerService {
  log(mensagem: string) {
    console.log(mensagem);
  }
}
```

Usando o Serviço em Múltiplos Componentes:

TypeScript

```typescript
import { Component, OnInit, Inject } from '@angular/core';
import { LoggerService } from '../logger.service';

@Component({
  selector: 'app-componente1',
  template: `
    <button (click)="logarMensagem()">Logar Mensagem</button>
  `,
})
export class Componente1 implements OnInit {
  constructor(
    @Inject(LoggerService) private loggerService: LoggerService
  ) {}

  ngOnInit() {}

  logarMensagem() {
    this.loggerService.log('Mensagem do Componente 1');
  }
}

@Component({
  selector: 'app-componente2',
  template: `
    <button (click)="logarErro()">Logar Erro</button>
  `,
})
export class Componente2 implements OnInit {
  constructor(
```

```
    @Inject(LoggerService) private loggerService: LoggerService
  ) {}

  ngOnInit() {}

  logarErro() {
    this.loggerService.log('Erro no Componente 2');
  }
}
```

Explicação:

- O `LoggerService` é injetado em cada componente usando `@Inject(LoggerService)`.
- Cada componente chama métodos do `LoggerService` para registrar mensagens no console.

3. Injetando um Serviço para Configuração Global:

Imagine um serviço `ConfiguracaoService` que fornece configurações globais para a aplicação:

Criando o Serviço:

TypeScript

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class ConfiguracaoService {
  apiBaseUrl = 'https://minha-api.com';
  linguagemPadrao = 'pt-BR';
}
```

Usando o Serviço para Acessar Configurações:

TypeScript

```
import { Component, OnInit, Inject } from '@angular/core';
import { ConfiguracaoService } from '../configuracao.service';

@Component({
  selector: 'app-meu-componente',
  template: `
    <p>API Base: {{ apiBaseUrl }}</p>
    <p>Linguagem Padrão: {{ linguagemPadrao }}</p>
  `,
})
export class MeuComponente implements OnInit {
```