Angular

Sumário

- Pipes
- Services
- Injeção de Dependências
- Referências

Pipes

- São funções simples para usar em expressões de modelo que aceita um valor de entrada e retorna um valor modificado ou seja transforma a saída exibida no template;
- São úteis, pois é possível usá-los em todo o aplicativo angular, declarando cada pipe apenas uma vez;
- Por exemplo, é possível usar uma barra vertical para mostrar uma data como 27 de abril de 1979, em vez do formato de string bruto;

```
data_de_hoje = new Date();
```

```
<h1>{{data_de_hoje | date}}</h1>
```

Dec 21, 2023

```
<h1>{{data_de_hoje | date:'shortDate'}}</h1>
```

12/21/23

```
<h1>{{data_de_hoje | date:'d/M/yy'}}</h1>
```

22/12/23

https://angular.io/api/common/DatePipe

Pipes embutidos no Angular

- **DatePipe** - Formata um valor de data de acordo com as regras de localidade;
- **UpperCasePipe -** Transforma o texto em letras maiúsculas;
- **LowerCasePipe -** Transforma o texto em letras minúsculas;
- **CurrencyPipe -** Transforma um número em uma string de moeda, formatada de acordo com as regras de localidade;
- **DecimalPipe -** Transforma um número em uma string com ponto decimal, formatada de acordo com as regras de localidade;
- **PercentPipe -** Transforma um número em uma string de porcentagem, formatada de acordo com as regras de localidade.

https://angular.io/api/common#pipes

Encadeamento de pipes

- É possível encadear vários pipes quando forem necessários para a transformação dos dados;

```
<h1>{{data_de_hoje | date:'fullDate' | uppercase}}</h1>
```

FRIDAY, DECEMBER 22, 2023

Criando pipes

- Interface PipeTransform – implementada por pipes para realizar uma transformação;
- O Angular invoca o método **transform** com o valor de um binding como o primeiro argumento e quaisquer parâmetros como o segundo argumento em forma de lista;

```
interface PipeTransform {

  transform(value: any, ...args: any[]): any

}
```

https://angular.io/api/core/PipeTransform

ng g pipe <nomeDoPipe>

TruncatePipe

- Retorna o valor abreviado com reticências adicionadas.

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({name: 'truncate'})
export class TruncatePipe implements PipeTransform {
  transform(value: string) {
    return value.split(' ').slice(0, 2).join(' ') + '...';
  }
}
```

Invoking `{{ 'It was the best of times' | truncate }}` in a template will produce `It was...`.

demoPipe

```
import { Pipe, PipeTransform } from "@angular/core";

@Pipe({name: 'demoPipe'})
export class DemoPipe implements PipeTransform {
  transform(valor: string): string {
    return valor.replace(/\b\w/g, char => char.toUpperCase());
  }
}
```

O rato roeu a roupa do rei de roma

O Rato Roeu A Roupa Do Rei De Roma

app.module.ts

```
import { DemoPipe } from './demo.pipe';

@NgModule({
  declarations: [
    AppComponent,
    DemoPipe
  ],
```

ngModelo\src\app\demo.pipe.ts

abreviaPipe

```ts
import { Pipe, PipeTransform } from "@angular/core";

@Pipe({name: 'abreviaPipe'})
export class AbreviaPipe implements PipeTransform {
  transform(valor: string): string {
    return valor
    .split(' ')
    .map(word => word.charAt(0).toUpperCase())
    .join('.');
  }
}
```

```html
<p>{{ frase | abreviaPipe }} </p>
```

O rato roeu a roupa do rei de roma

O Rato Roeu A Roupa Do Rei De Roma

O.R.R.A.R.D.R.D.R

abreviaPipe com parâmetros

```
import { Pipe, PipeTransform } from "@angular/core";

@Pipe({name: 'abreviaPipe'})
export class AbreviaPipe implements PipeTransform {
  transform(valor: string, limite:number = 1): string {
    return valor
    .split(' ')
    .map(word => word.substring(0, limite).toUpperCase())
    .join('.');
  }
}
```

```
<p>{{ frase | abreviaPipe:3 }} </p>
```

O rato roeu a roupa do rei de roma

O Rato Roeu A Roupa Do Rei De Roma

O.RAT.ROE.A.ROU.DO.REI.DE.ROM

Async Pipe

```
teste = new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Promise resolvida");
    }, 2000)});
```

```
<h1>{{teste}}</h1>
```

[object Promise]

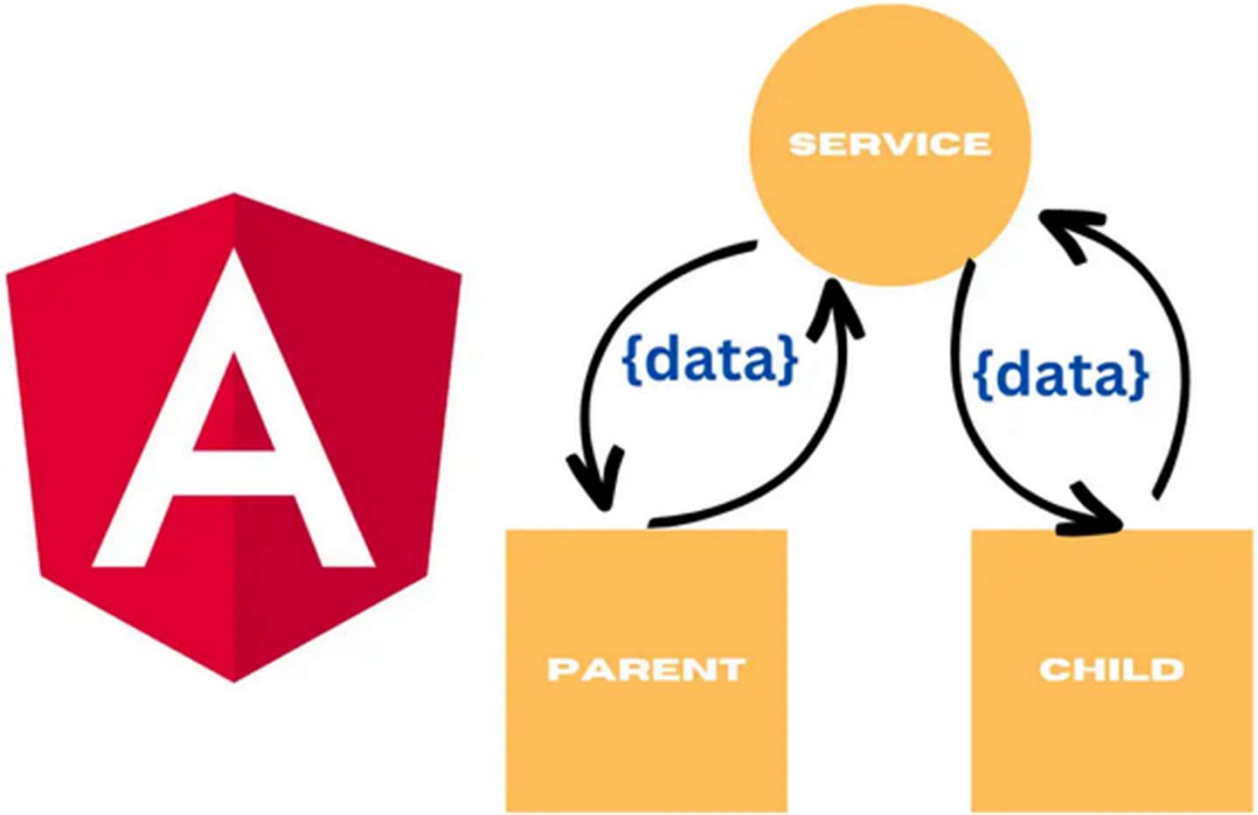
```
<h1>{{teste | async}}</h1>
```

Promise resolvida

Services

- Categoria que abrange qualquer valor, função ou recurso que um aplicativo precisa;
- Classe com uma finalidade restrita e bem definida. Deve fazer algo específico e bem feito;
- O Angular faz uma separação entre componentes e services para aumentar a modularidade e a reutilização;
- A função de um componente é permitir apenas a experiência do usuário e **DEVE** usar os services para tarefas que não envolvam a visualização ou a lógica do aplicativo;
- No Angular, a injeção de dependência disponibiliza os services aos componentes.

Services



Dependency injection (DI)

- A injeção de dependência (DI) é a parte da estrutura Angular que fornece aos componentes acesso aos **services** e outros recursos;
- O Angular fornece a capacidade de injetar um **service** em um componente para dar a esse componente acesso ao **service;**
- DI injeta uma dependência (ou uma instância de uma classe, função ou valor) no componente
- Utilizar o decorador **@Injectable()** a uma classe de **service** para que o Angular possa injetá-lo em um componente como uma dependência;
- É preciso informar ao Angular que determinado componente precisa de determinado serviço.

Dependency injection (DI)

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

@Injectable({
  providedIn: 'root'
})

export class LogandoService {

  constructor() { }
}
```

Ng generate service <nomeDoServico>

provideIn: root -> uma única instância do service fica disponível em todo o aplicativo.

Injetando um Service num Componente

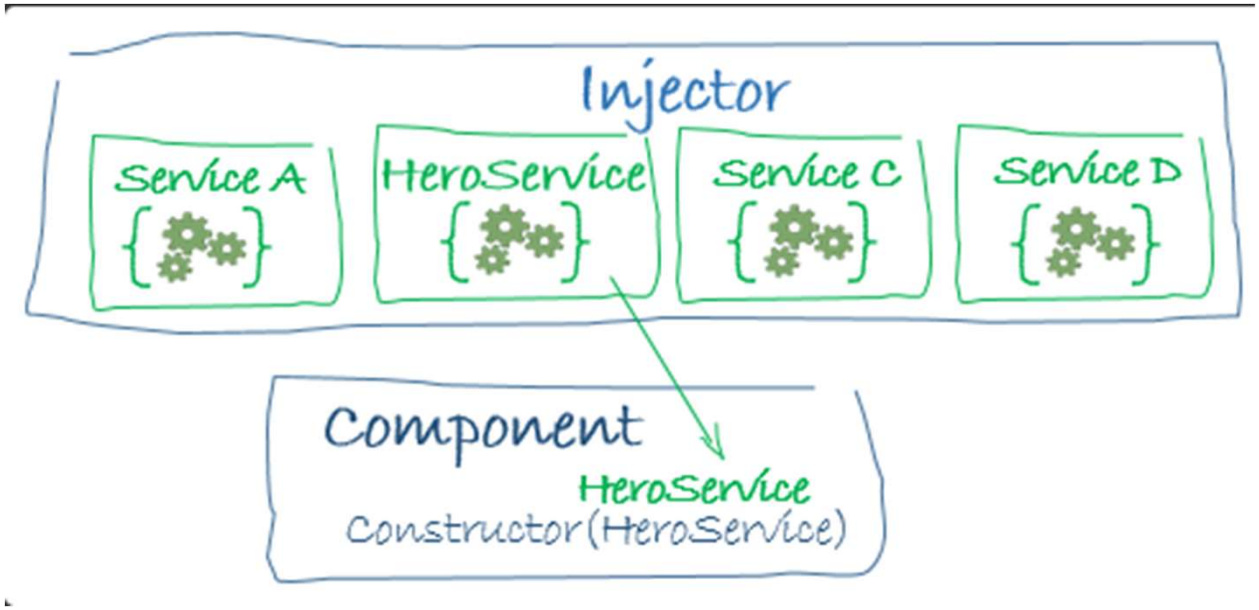
- Quando o Angular cria uma nova instância de uma classe de componente, ele determina quais serviços ou outras dependências esse componente precisa, observando os tipos de parâmetros do construtor;

```
export class AppComponent {
  title = 'Services';
  weatherData: any;

  constructor(private tempoService: PrevisaoTempoService) {

  }
```

Injetando um Service num Componente



- um **Injetor** é um componente central do sistema de dependency Injector (DI), que é responsável por criar e gerenciar instâncias de serviços para aquele componente.

Injetando um Service num Componente

- Quando o Angular descobre que um componente depende de um serviço, ele primeiro verifica se o **Injector** possui alguma instância desse serviço;
- Caso ainda não exista uma instância de serviço solicitada, o **Injector** cria uma usando o provedor cadastrado e a adiciona ao **Injector** antes de retornar o serviço ao Angular;
- Quando todos os serviços solicitados forem resolvidos e retornados, o Angular poderá chamar o construtor do componente com esses serviços como argumentos.

Hierarquia de Injectors

- Cada componente Angular possui seu próprio **injector**;
- Os **injectors** são organizados hierarquicamente, formando uma árvore que espelha a árvore de componentes;
- Quando um componente solicita um serviço, o Angular procura o serviço no injetor associado a esse componente;
- Caso o serviço não seja encontrado, o Angular continua a busca no injetor pai, e assim por diante, até chegar ao injetor raiz;
- O escopo de uma instância de serviço é limitado ao **injector** associado ao componente que solicitou o serviço. Isto significa que diferentes instâncias de um serviço podem existir em diferentes níveis da árvore de componentes.

Hierarquia de Injectors

AppModule

Mesma Instância do Service está disponível para toda a aplicação (todos os componentes, todas as directivas, todos os outros Services)

AppComponent

Mesma Instância do Service está disponível para todos os componentes (mas não para todos os Services)

Qualquer outro **Component**

Mesma Instância do Service está disponível o **component** e todos os seus filhos

Hierarquia de Injectors

App.componente.ts

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers: [UsuariosService]
})
export class AppComponent {
  title = 'Services';
  weatherData: any;
```

```
export class AdministradoresComponent {
  usuarios: any = [];
  constructor(private usuariosService: UsuariosService) { }
```

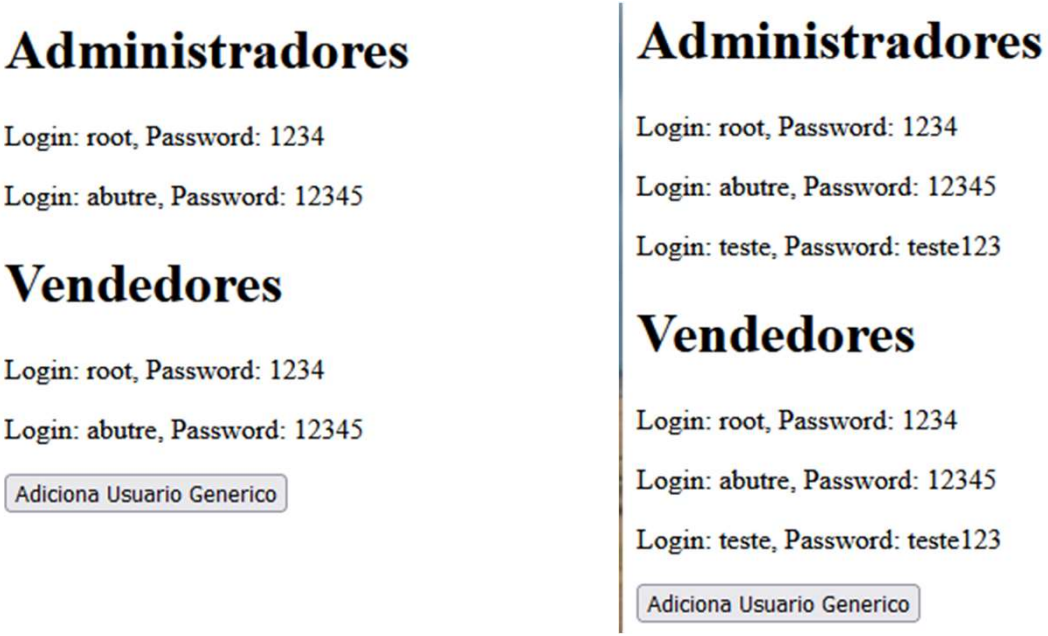
```
export class VendedoresComponent {
  usuarios: any = [];

  constructor(private usuariosService: UsuariosService) { }
```

```
@Injectable()
export class UsuariosService {
  private usuarios = [
    { login: 'root', password: '1234' },
    { login: 'abutre', password: '12345' },
  ];
```

- Ambos os componentes (Administradores e Vendedores) estão compartilhando a mesma instância do Service UsuariosService

Hierarquia de Injectors



Aula8\1.1_Services\src\app\app.component.html

Hierarquia de Injectors

Administradores

Login: root, Password: 1234

Login: abutre, Password: 12345

Vendedores

Login: root, Password: 1234

Login: abutre, Password: 12345

Login: teste, Password: teste123

Login: teste, Password: teste123

Login: teste, Password: teste123

Adiciona Usuario Generico

Exercício 01

- Crie uma aplicação angular para exibir o nome de todos os países do mundo e suas populações numa tabela (Aula 5);
- Crie um serviço que que acessa a API https://restcountries.com/ e alimente uma propriedade que que contenha o nome e a população de cada país no mundo;
- Crie um componente para exibir os dados dos países, injete o serviço criado nesse componente para ter acesso aos países;
- Organize os dados em forma de uma tabela;

Referências

- https://angular.io/api/common/DatePipe
- https://angular.io/guide/dependency-injection
- https://angular.io/guide/architecture-services