

## ***1 Introdução***

**1.1** É comum que uma classe possua dependências. Ela pode usar, por exemplo, objetos de outras classes para desempenhar aquilo que promete. Assim, ela **depende** da existência dessas classes para também poder existir.

Há um padrão de projeto conhecido como **Injeção de Dependências** que permite que uma classe solicite suas dependências para uma fonte externa ao invés de ela mesma criá-las.

O Angular possui um mecanismo de Injeção de Dependências embutido que utilizaremos neste material.

**1.2** Aplicações Front end, muito comumente, interagem com um Back end para disponibilizar todas as funcionalidades que prometem. Muitas vezes, a comunicação entre elas é feita utilizando-se o protocolo **HTTP**. Um dos módulos que o Angular oferece implementa o protocolo HTTP e pode ser injetado com o mecanismo de injeção de dependências.

A aplicação desenvolvida neste material fará a exibição de uma **lista de previsões do tempo**. A princípio, ela irá exibir uma coleção de previsões que estão fixas no código. Aplicaremos a injeção de dependências e, a seguir,

Veja a Figura 1.1.

Figura 1.1



## 2 Desenvolvimento

**2.1 (Novo projeto)** Comece abrindo um novo terminal e navegando até o seu workspace, ou seja, a pasta que você está usando para abrigar os seus projetos Angular. Tome o cuidado de não acessar a pasta de nenhum projeto já existente. Use os comandos a seguir para criar um novo projeto Angular e navegar até o seu diretório.

```
ng new nome-do-projeto
cd nome-do-projeto
```

- Quando perguntado, escolha não adicionar um módulo para roteamento e use o modelo CSS simples.

- Instale o Bootstrap com

```
npm install bootstrap@latest
```

- Abra uma instância do VS Code vinculada ao diretório atual com o seguinte comandos

```
code .
```

- A seguir, configure o Bootstrap no arquivo **angular.json** adicionando o arquivo **node\_modules/bootstrap/dist/css/bootstrap.css** ao vetor cuja chave é **projects/nome-do-projeto/architect/build/options/styles**.

- Coloque o servidor de testes em execução com

```
ng serve --open
```

**2.2 (O modelo de dados)** Começamos descrevendo o que é uma previsão. O serviço que utilizaremos devolve diversas informações para cada previsão. Estamos interessados em **data, descrição, temperatura máxima, temperatura mínima, humidade relativa do ar e uma imagem**. Assim, use o Angular CLI para criar uma classe que descreve o que é uma previsão com o seguinte comando.

```
ng g class model/previsao --skipTests=true
```

A Listagem 2.2.1 mostra as propriedades de uma Previsao.

### Listagem 2.2.1

```
export class Previsao {  
  data: string;  
  descricao: string;  
  tempMax: number;  
  tempMin: number;  
  humidity: number;  
  imgURL: string;  
}
```

**2.3 (Vetor de previsões)** O componente principal terá, a princípio, um vetor de previsões fictícias. Veja a Listagem 2.3.1.

### Listagem 2.3.1

```
export class AppComponent {  
  
  previsoes: Previsao[] = [  
    {  
      data: '20/02/2020 15:00',  
      descricao: 'Sol',  
      tempMax: 32,  
      tempMin: 27,  
      humidity: 0.87,  
      imgURL: '../assets/01d.png'  
    },  
    {  
      data: '20/02/2020 18:00',  
      descricao: 'Sol com nuvens',  
      tempMax: 28,  
      tempMin: 22,  
      humidity: 0.85,  
      imgURL: '../assets/02d.png'  
    },  
    {  
      data: '20/02/2020 21:00',  
      descricao: 'Noite limpa',  
      tempMax: 24,  
      tempMin: 18,  
      humidity: 0.82,  
      imgURL: '../assets/01n.png'  
    }  
  ]  
}
```

- Note que cada previsão tem uma figura associada. Faça o download delas do site da disciplina e armazena-as no diretório **assets**.

**2.4 (Componente para a exibição de uma previsão)** Use o comando a seguir para criar um componente Angular cuja finalidade é receber uma Previsao recebida por parâmetro.

#### ng g c previsao

- O arquivo **previsao.component.ts** deve definir um objeto do tipo Previsao que será recebido pelo componente. Assim, ele deve ser anotado com **@Input()**. Veja a Listagem 2.4.1.

Listagem 2.4.1

```
import { Component, Input } from '@angular/core';
import { Previsao } from '../model/previsao';

@Component({
  selector: 'app-previsao',
  templateUrl: './previsao.component.html',
  styleUrls: ['./previsao.component.css']
})
export class PrevisaoComponent {
  @Input() previsao: Previsao;
}
```

- O arquivo **previsao.component.html** define a forma como um objeto Previsao é exibido. Utilizamos um card do Bootstrap. Veja a Listagem 2.4.2.

Listagem 2.4.2

```
<div class="card" style="width: 18rem;">
  <div class="card-header text-center">
    <div class="d-flex flex-row">
      
      <div class="">
        <p>{{previsao.data}}</p>
        <p>{{previsao.descricao}}</p>
      </div>
    </div>
  </div>
  <div class="card-body">
    <div class="bg-light p-4 text-center mb-2">Temperatura máxima: {{previsao.tempMax}}&deg;</div>
    <div class="bg-light p-4 text-center mb-2">Temperatura mínima: {{previsao.tempMin}}&deg;</div>
    <div class="bg-light p-4 text-center">Humidade relativa do ar: {{previsao.humidity | percent:'2.1-1'}}</div>
  </div>
</div>
```

**2.5 (Exibindo as previsões no arquivo app.component.html)** Cabe ao componente principal da aplicação utilizar o componente recém criado para exibir cada previsão fictícia contida no vetor. Substitua o conteúdo do arquivo **app.component.html** pelo conteúdo da Listagem 2.5.1.

Listagem 2.5.1

```
<div class="container">
  <div class="row">
    <app-previsao class="col-4 mb-2" *ngFor="let previsao of previsoes"
[previsao]="previsao"></app-previsao>
  </div>
</div>
```

**2.6 (Definindo e injetando um serviço)** Neste passo iremos ilustrar o uso da **injeção de dependências**. No momento, o componente principal da aplicação é quem define o vetor de previsões. Contudo, essa coleção de dados pode ser de interesse para outros componentes, os quais teriam de defini-la novamente ou estabelecer dependências com o componente principal. Além disso, essa é uma tarefa que não cabe a um componente e isso fere a sua **alta coesão**. Assim, vamos definir um **serviço** cuja finalidade é definir a coleção e dar acesso a ela a todos os componentes que precisarem. No futuro, caberá também ao serviço a tarefa de isolar as requisições HTTP que permitem a obtenção da coleção de previsões reais. Para criar um serviço com o Angular CLI, use o seguinte comando.

#### ng g s Previsoes

- A seguir, transporte a definição do vetor de previsões do componente para o serviço recém criado (arquivo **previsoes.service.ts**). Veja a Listagem 2.6.1.

### Listagem 2.6.1

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

```
@Injectable({  
  providedIn: 'root'  
})
```

```
export class PrevisoesService {
```

```
  previsoes: Previsao[] = [  
    {  
      data: '20/02/2020 15:00',  
      descricao: 'Sol',  
      tempMax: 32,  
      tempMin: 27,  
      humidity: 0.87,  
      imgURL: '../assets/01d.png'  
    },  
    {  
      data: '20/02/2020 18:00',  
      descricao: 'Sol com nuvens',  
      tempMax: 28,  
      tempMin: 22,  
      humidity: 0.85,  
      imgURL: '../assets/02d.png'  
    },  
    {  
      data: '20/02/2020 21:00',  
      descricao: 'Noite limpa',  
      tempMax: 24,  
      tempMin: 18,  
      humidity: 0.82,  
      imgURL: '../assets/01n.png'  
    }  
  ]
```

```
  constructor() { }
```

```
}
```

- Ainda no arquivo **previsoes.service.ts**, define um método que devolve o vetor recém-criado. Veja a Listagem 2.6.2.

Listagem 2.6.2

```
public obterPrevisoes(): Previsao[] {  
    return this.previsoes;  
}
```

- Como o serviço é quem define a lista de previsões, o componente principal **depende** de uma instância dele para obtê-la. Utilizaremos, neste momento, o mecanismo de injeção de dependências do Angular para obtê-la. A injeção de dependências é feita no construtor do componente, pois o Angular nos entrega a instância desejada no momento em que constrói o componente. Veja a Listagem 2.6.3.

Listagem 2.6.3

```
import { Component } from '@angular/core';  
import { Previsao } from '../model/previsao';  
import { PrevisoesService } from '../previsoes.service';  
  
@Component({  
    selector: 'app-root',  
    templateUrl: './app.component.html',  
    styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  
    private previsoes: Previsao[];  
  
    constructor(private previsoesService: PrevisoesService) {  
        this.previsoes = previsoesService.obterPrevisoes();  
    }  
}
```

**2.7 (Cadastro no site de previsões)** O site do Link 2.7.1 oferece um serviço de previsões gratuito. É possível solicitar a previsão para os próximos 5 dias de três em três horas. Para utilizá-lo, basta fazer seu cadastro e obter a sua chave de API.

Link 2.7.1

<https://openweathermap.org/>



- Depois de se cadastrar, dê uma olhada no serviço do Link 2.7.2.

Link 2.7.2

<https://openweathermap.org/forecast5>

A Figura 2.7.1 mostra um trecho do seu retorno em JSON.

Figura 2.7.1

```
{
  "cod": "200",
  "message": 0,
  "cnt": 40,
  "list": [
    {
      "dt": 1578409200,
      "main": {
        "temp": 284.92,
        "feels_like": 281.38,
        "temp_min": 283.58,
        "temp_max": 284.92,
        "pressure": 1020,
        "sea_level": 1020,
        "grnd_level": 1016,
        "humidity": 90,
        "temp_kf": 1.34
      },
      "weather": [
        {
          "id": 804,
          "main": "Clouds",
          "description": "overcast clouds",
          "icon": "04d"
        }
      ]
    }
  ]
}
```

O Link 2.7.3 mostra um exemplo de uso da API. Teste-o no seu navegador, usando a sua chave.

Link 2.7.3

[http://api.openweathermap.org/data/2.5/forecast?  
q=itu&appid=SUA\\_CHAVE\\_AQUI&units=metric&lang=pt\\_br&cnt=16](http://api.openweathermap.org/data/2.5/forecast?q=itu&appid=SUA_CHAVE_AQUI&units=metric&lang=pt_br&cnt=16)

**2.8 (Usando o módulo HTTP)** O primeiro passo para utilizar o módulo que implementa o protocolo HTTP é importá-lo no arquivo **app.module.ts**. Veja a Listagem 2.8.1.

Listagem 2.8.1

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http'

import { AppComponent } from './app.component';
import { PrevisaoComponent } from './previsao/previsao.component';

@NgModule({
  declarations: [
    AppComponent,
    PrevisaoComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- A seguir, podemos injetar uma instância de `HttpClient` no serviço. Veja a Listagem 2.8.2.

Listagem 2.8.2

```
constructor(private httpClient: HttpClient) {

}
```

- Uma requisição a um servidor pode levar um tempo para ser concluída. Por isso, o resultado não nos é entregue imediatamente. Utilizaremos o método **get** do módulo HTTP para obter um **Observable**, que é um objeto que representa a possibilidade de obtenção do resultado de interesse no futuro. Veja a nova implementação do método `obterPrevisoes` no arquivo **previsoes.services.ts** na Listagem 2.8.3.

### Listagem 2.8.3

```
import { Observable } from 'rxjs';
public obterPrevisoes(): Observable<Previsao[]> {
    return this.httpClient.get<Previsao[]>('URL_AQUI');
}
```

- Desta forma, a definição do vetor de previsões fictícias pode ser comentada ou apagada do serviço.

**2.9 (Usando o Observable no componente principal)** O componente principal precisa ser ajustado para utilizar o Observable adequadamente. Veja a Listagem 2.9.1.

### Listagem 2.9.1

```
import { Component } from '@angular/core';
import { Previsao } from './model/previsao';
import { PrevisoesService } from './previsoes.service';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {

    private previsoes: Previsao[];

    constructor(private previsoesService: PrevisoesService) {
        previsoesService.obterPrevisoes().subscribe((previsoes) => {
            this.previsoes = previsoes['list'];
            console.log(this.previsoes);
        });
    }
}
```

- Inspeccione a saída no log para entender o objeto recebido.

**2.10 (Classes de modelo condizentes com o JSON do serviço)** Quando fizermos uma requisição HTTP, o resultado pode ser convertido automaticamente para objetos JSON caso tenhamos a sua descrição apropriada. Por isso, vamos definir novas classes de modelo que estejam de acordo com aquilo que nos é entregue pelo serviço. Use o comando **ng g class model/nome** para criar cada uma das classes da Listagem 2.10.1. Além disso, altere a implementação da classe Previsao.

### Listagem 2.10.1

```
export class Main {
  temp_max: number;
  temp_min: number;
  humidity: number;
}

export class Weather {
  description: string;
  icon: string;
}

import { Main } from './main';
import { Weather } from './weather';

export class Previsao {
  dt: number;
  main: Main;
  weather: Weather[];
}
```

**2.11 (Exibição das previsões de acordo com o modelo novo)** O arquivo **previsao.component.html** precisa ser atualizado para acessar as propriedades corretas de cada previsão, de acordo com o modelo novo. Veja a Listagem 2.11.1.

## Listagem 2.11.1

```

<div class="card" style="width: 18rem;">
  <div class="card-header text-center">
    <div class="d-flex flex-row">
      
      <div class="">
        <p>{{previsao.dt * 1000 | date: 'dd/MM/y hh:mm:ss'}}</p>
        <p>{{previsao.weather[0].description}}</p>
      </div>
    </div>
  </div>
  <div class="card-body">
    <div class="bg-light p-4 text-center mb-2">Temperatura máxima:
{{previsao.main.temp_max}}&deg;</div>
    <div class="bg-light p-4 text-center mb-2">Temperatura mínima:
{{previsao.main.temp_min}}&deg;</div>
    <div class="bg-light p-4 text-center">Humidade relativa do ar:
    {{previsao.main.humidity / 100 | percent:'2.1-1'}}
    </div>
  </div>
</div>

```

## ***Exercícios***

1. Adicione um campo textual e um botão à aplicação. O usuário pode digitar a cidade que desejar no campo textual e, quando clicar no botão, ver as previsões para aquela cidade.

## ***Referências***

Angular. 2020. Disponível em <<https://angular.io>>. Acesso em junho de 2020.

Current weather and forecast - OpenWeatherMap. 2020. Disponível em <<https://openweathermap.org/>>. Acesso em junho de 2020.