@fpftech.educacional

FPFtech
Escola Tecnológica

FPFtech
Escola Tecnológica

O que sabemos até agora?

1 – Angular baseia-se em **componentes.**

2 – Componentes são subdivididos em **HTML, CSS, TS (TYPESCRIPT).**

3 – Data Binding faz com que dados no TS apareçam no HTML.

4 – Podemos deixar nosso HTML mais dinâmico com **Diretivas**(@If..)

5 – Interfaces transformam Objetos em Tipagem.

6 – Serviços manipulam dados e fazem requisições HTTP.(get,post...)

7 – Serviços são **injetados** no TS, diretamente numa **variável privada**.

8 – Rotas com **RouterLink** e aparecem na tela pelo **<router-outlet>**

Consumo de APIs com HttpClient

O que é o HttpClientModule?

O Angular possui uma biblioteca nativa para comunicação com servidores via HTTP: HttpClientModule.

Com ela, conseguimos buscar **dados de APIs externas**, como notícias, produtos, usuários, etc.

No nosso projeto final, usaremos ela pra pegar valores da nossa API Django.

Exemplo

Imagine um aplicativo de previsão do tempo.

Ele precisa buscar informações atualizadas em tempo real.

Para isso, ele se conecta a um servidor externo (API).

O que ele usa para isso? **HttpClient**

Exemplo url no postman

https://api.open-meteo.com/v1/forecast

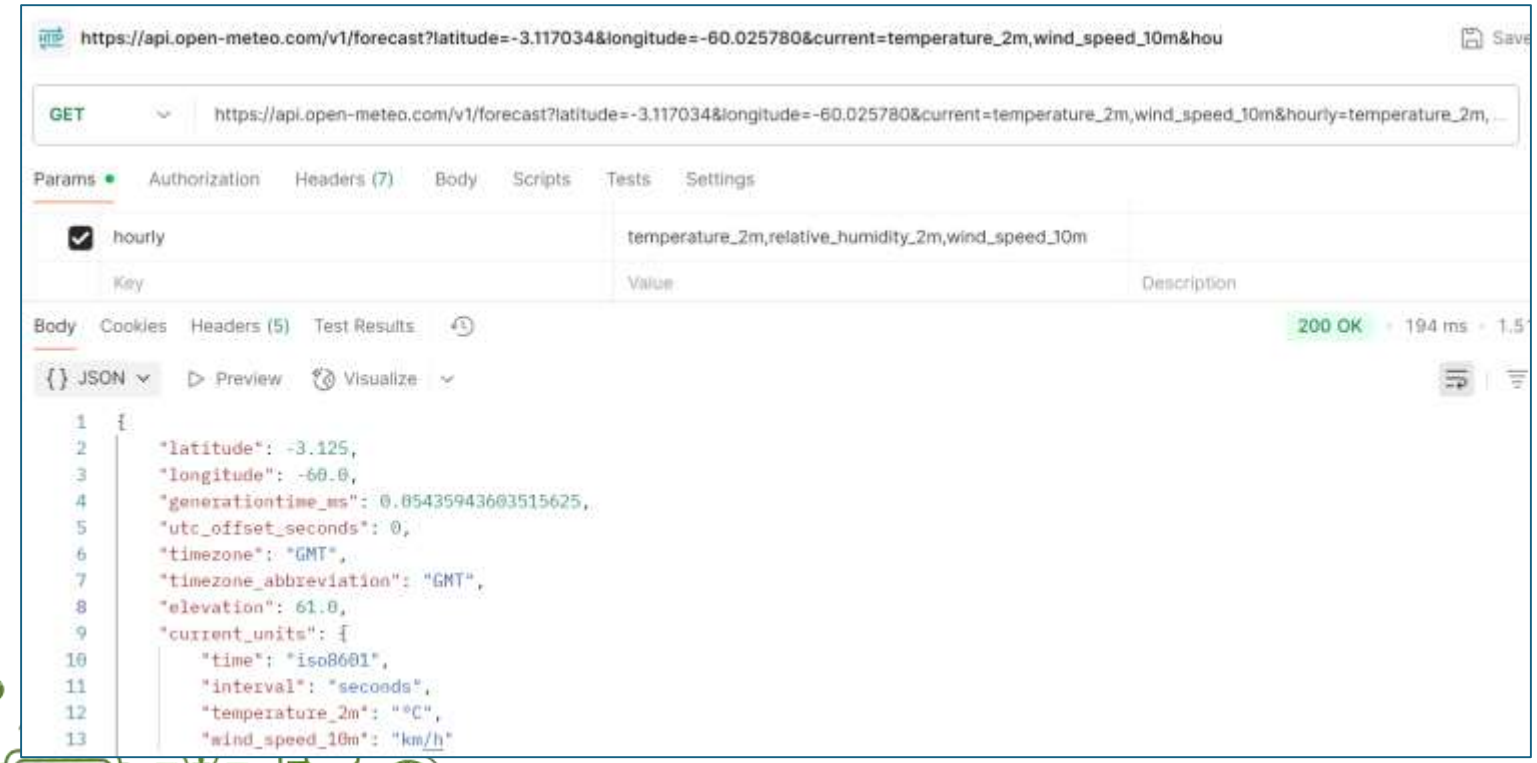
Queryparams:

latitude=-3.117034

longitude=-60.025780

current=temperature_2m,wind_speed_10m

hourly=temperature_2m,relative_humidity_2m,wind_speed_10m



Exemplo postman resposta json


```
{
    "latitude": -3.125,
    "longitude": -60.0,
    "generationtime_ms":
11.741161346435547,
    "utc_offset_seconds": 0,
    "timezone": "GMT",
    "timezone_abbreviation": "GMT",
    "elevation": 61.0,
    "current_units": {
        "time": "iso8601",
        "interval": "seconds",
        "temperature_2m": "°C",
        "wind_speed_10m": "km/h"
    },
    "current": {
        "time": "2025-05-30T16:30",
        "interval": 900,
        "temperature_2m": 29.4,
        "wind_speed_10m": 6.8
    },
    "hourly_units": {
        "time": "iso8601",
        "temperature_2m": "°C",
        "relative_humidity_2m": "%",
        "wind_speed_10m": "km/h"
    },
    "hourly": {
        "time": [
            "2025-05-30T00:00",
            "2025-05-30T01:00",
             ...
            "2025-06-05T22:00",
            "2025-06-05T23:00"
        ],
        "temperature_2m": [
            27.9,
            27.5,
            ...
            29.2,
            28.4
        ],
        "relative_humidity_2m": [
            83,
            86,
            ...
            71,
            77
        ],
        "wind_speed_10m": [
            3.6,
            3.8,
            ...
            6.3,
            4.7
        ]
    }
}
```

Como ativar o HttpClient no seu projeto Angular

1 – **app.config.ts**

Importe provedor HttpClient

Adicione na lista de provedores

```
import { ApplicationConfig, provideZoneChangeDetection } from
import { provideRouter } from '@angular/router';
import {provideHttpClient} from '@angular/common/http';

import { routes } from './app.routes';

1+ usages   jonatas.lopes *
export const appConfig: ApplicationConfig = {
  providers: [
    provideZoneChangeDetection( options: { eventCoalescing: tru
    provideRouter(routes),
    provideHttpClient()
  ]
};
```

Como ativar o HttpClient no seu projeto Angular

FPFtech
Escola Tecnológica

2 – **serviço**

Para usar,

basta injetar no

serviço,

```
import {HttpClient} from '@angular/common/http';
import {Observable} from 'rxjs';

1+ usages  new *
@Injectable({providedIn: 'root'})
export class ClimaService {
  private http = inject(HttpClient);    You, 28/0
```

Depois usa na função. *(Nesse exemplo, usamos o retorno do tipo “any”)*

```
getClimaData(): Observable<any> {
  return this.http.get( url: 'https://api.open-meteo.com/v1/forecast')
}
```

Fazendo “clima”

1 – Crie o componente “clima” = *ng g c components/clima*

2 – Crie a Interface ClimaTable: *{time: string, temperature: number, humidity: number, windSpeed: number}*

3 – Crie o serviço “climaService” = *ng g s services/clima*

3.1 – No climaService, injete Http e crie a função getClima();

```
export class ClimaService {
  private http = inject(HttpClient);
  private apiUrl = 'https://api.open-meteo.com/v1/forecast';
  private latitude = -3.117034;
  private longitude = -60.025780;

  1+ usages  new *
  getClima(): Observable<any> {
    const params = [
      `latitude=${this.latitude}`, `longitude=${this.longitude}`,
      `current=temperature_2m,wind_speed_10m`,
      `hourly=temperature_2m,relative_humidity_2m,wind_speed_10m`
    ].join('&');

    return this.http.get( url: `${this.apiUrl}?${params}`);
  }
```

Fazendo “clima”, 2

2 – **Interface**

Importe provedor HttpClient

Adicione na lista de provedores

```
export interface ClimaTable {
  time: string;
  temperature: number;
  humidity: number;
  windSpeed: number;
}
```

Fazendo “clima”, 3

3 – **Vá para um serviço** (nesse exemplo, criamos **serviço de clima**)

Faça as importações:

crie variável privada

e faça injeção do http

mude o tipo do retorno

da função,

Retorne o GET url

```
import { inject, Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {Observable} from 'rxjs';

no usages
@Injectable({providedIn: 'root'})
export class ClimaService {
  private http :HttpClient  = inject(HttpClient);
  private apiUrl :string  = 'https://api.open-meteo.com/v1/forecast';
  private latitude :number  = -3.117034;
  private longitude :number  = -60.025780;

  no usages
  getClima(): Observable<any> {
    const params :string  = [
      `latitude=${this.latitude}`, `longitude=${this.longitude}`,
      `current=temperature_2m,wind_speed_10m`,
      `hourly=temperature_2m,relative_humidity_2m,wind_speed_10m`
    ].join('&');

    return this.http.get( url: `${this.apiUrl}?${params}`);
  }
```

Fazendo “clima”, 4

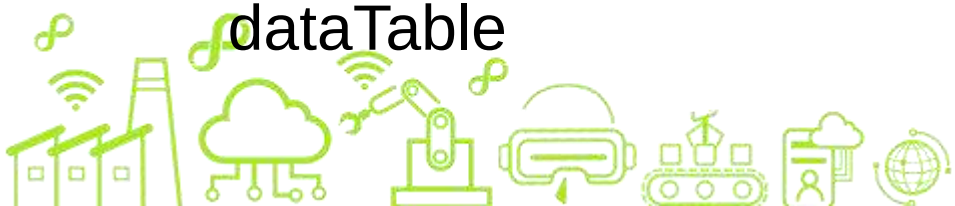
4 –**Component CLIMA.ts**

- Faça as importações:
- crie variável privada e faça injeção do serviço
- Crie variáveis: climaData, date, dataTable

```
import {Component, inject} from '@angular/core';
import {ClimaService} from '../../services/clima.service';
import {ClimaTable} from '../../interfaces/climaTable';

1+ usages  new *
@Component({
  selector: 'app-clima',
  imports: [],
  standalone: true,
  templateUrl: './clima.component.html',
  styleUrl: './clima.component.css'
})

export class ClimaComponent {
  private climaService = inject(ClimaService);
  climaData:any;
  date: Date = new Date();
  dataTable:ClimaTable[] = [];
```



4.1. Component clima.TS

Criamos a função de subtrair4horas que vai ser chamada no NgOnInit

```
subtrair4horas(dataEmString:string) {
  let novaData : Date  = new Date(dataEmString);
  novaData.setHours( hours: novaData.getHours() - 4);
  return novaData
}
```

4.2. Component clima.TS

No componente CLIMA.ts

- Antes função *subtrair 4horas()*
- **ngOnInit** rodar o serviço
- Salvar resultados

```
ngOnInit(){
  this.climaService.getClima()
    .subscribe( observerOrNext: (response:any) => {
      this.climaData = response
      this.date = this.subtrair4horas(response.current.time);
      this.dataTable = response.hourly.time.map((t:string, i:number) => ({
        time: this.subtrair4horas(t),
        temperature: response.hourly.temperature_2m[i],
        humidity: response.hourly.relative_humidity_2m[i],
        windSpeed: response.hourly.wind_speed_10m[i]
      }));
    });
}
```

5. clima.HTML

5 – **No CLIMA.html**

- Faça as interpolações

- Crie a tabela
- Primeira fila é o table_header
- Depois vem o @for, criando os outros table_row

```
<div class="data-info">
  <p>Latitude: {{ climaData.latitude }}</p>
  <p>Longitude: {{ climaData.longitude }}</p>
  <p>Elevação: {{ climaData.elevation }}</p>
  <p>Data: {{ date }}</p>
  <p>Temperatura: {{ climaData.current.temperature_2m }} °C</p>
  <p>Vento: {{ climaData.current.wind_speed_10m }} km/h</p>
</div>
<table>
  <tr>
    <th>Hora</th>
    <th>Temperatura</th>
    <th>Humidade</th>
    <th>Vento</th>
  </tr>
  @for (data of dataTable; track data) {
    <tr>
      <td>{{ data.time }}</td>
      <td>{{ data.temperature }}°C</td>
      <td>{{ data.humidity }}%</td>
      <td>{{ data.windSpeed }} m/s</td>
    </tr>
  }
</table>
```

6.CSS

```css
body {
  font-family: 'Segoe UI', sans-serif;
  background: #f5f7fa;
  color: #333;
  padding: 2rem;
  line-height: 1.6;
}

h1 {
  text-align: center;
  color: #2a4365;
}

p {
  margin: 0.5rem 0;
  font-size: 1rem;
}

.data-info {
  background: #e2e8f0;
  padding: 1rem;
  border-radius: 8px;
  margin-bottom: 2rem;
  box-shadow: 0 2px 5px rgba(0,0,0,0.05);
  max-width: 600px;
}
```

```css
table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 1rem;
  background: #ffffff;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);
  border-radius: 8px;
  overflow: hidden;
}

th, td {
  padding: 0.75rem 1rem;
  text-align: center;
  border-bottom: 1px solid #e2e8f0;
}
th {
  background-color: #2b6cb0;
  color: white;
  font-weight: 600;
}
tr:nth-child(even) {
  background-color: #f7fafc;
}
tr:hover {
  background-color: #ebf8ff;
}
```

Fazendo o Desafio: Vizualizando API “clima”

7 – Atualize rotas, adicione o tarefaComponent lá.

7.1. – Atualize o App.HTML, com o link <a> para o novo componente. Rode a aplicação.

App.routes.ts

```
import { Routes } from '@angular/router';
import {NomeComponenteComponent} from './nome-componente/nome-componente.component';
import {ClimaComponent} from './clima/clima.component';

1+ usages  new *
export const routes: Routes = [
  {path: '', component: NomeComponenteComponent},
  {path: 'clima', component: ClimaComponent},
  {path: '**', redirectTo: ''}
];
```

App.component.html

```
<nav>
  <a [routerLink]="['']">Nome Componente</a>
  |
  <a [routerLink]="['/clima']">Clima</a>
</nav>
<hr /> <!--linha horizontal-->
<router-outlet></router-outlet>
```

Resumo

1. – Crie o componente “clima” = *ng g c components/clima*

2. – Crie a Interface ClimaTable:

{time: string, temperature: number, humidity: number, windSpeed: number}

3. – Crie o serviço “climaService” = *ng g s services/clima*

3.1 – No climaService, injete Http e crie a função getClima();

4. – Clima.TS, faça importações(*inject,serviço,interface*), injeção serviço, cria variáveis.

4.1. – Função subtrair4horas;

4.2. – ngOnInit roda função *“getClima”* e atualiza os valores das variáveis.

5. – Clima.HTML mostra os dados em tela.

6. – Clima.CSS deixa os dados mais bonitos.

7. – App.rotas.ts importar o componente criado e adicionar um path pra ele.

7.1. – Atualizar App.HTML em <nav> novo link <a>

Rode e veja

Nome Componente | Clima

Latitude: -3.125

Longitude: -60

Elevação: 61

Data: Wed May 28 2025 10:45:00 GMT-0400 (Horário Padrão do Amazonas)

Temperatura: 31 °C

Vento: 8.3 km/h

Hora	Temperatura	Humidade	Vento
Wed May 28 2025 06:00:00 GMT-0400 (Horário Padrão do Amazonas)	26.2°C	89%	2.5 m/s
Wed May 28 2025 06:00:00 GMT-0400 (Horário Padrão do Amazonas)	26.2°C	90%	2.9 m/s
Wed May 28 2025 06:00:00 GMT-0400 (Horário Padrão do Amazonas)	26°C	92%	4 m/s
Wed May 28 2025 06:00:00 GMT-0400 (Horário Padrão do Amazonas)	26°C	93%	3.6 m/s
Wed May 28 2025 06:00:00 GMT-0400 (Horário Padrão do Amazonas)	25.8°C	94%	3.2 m/s
Wed May 28 2025 06:00:00 GMT-0400 (Horário Padrão do Amazonas)	25.8°C	94%	2.5 m/s

O que sabemos até agora?

1 – Angular baseia-se em **componentes.**

2 – Componentes são subdivididos em **HTML, CSS, TS (TYPESCRIPT).**

3 – Data Binding faz com que dados no TS apareçam no HTML.

4 – Podemos deixar nosso HTML mais dinâmico com **Diretivas**(@If..)

5 – Interfaces transformam Objetos em Tipagem.

6 – Serviços manipulam dados e fazem requisições HTTP.(get,post...)

7 – Serviços são **injetados** no TS, diretamente numa **variável privada**.

8 – Rotas com **RouterLink** e aparecem na tela pelo **<router-outlet>**

9 – Usamos **HttpClient** dentro do **Serviço** para fazer requisição HTTP.

Obrigado!

FPFtech
Escola Tecnológica

@fpftech.educacional