

## Requisições HTTP

Realizar requisições HTTP é uma das tarefas mais comuns em SPAs, atualmente. É por meio delas que a aplicação se comunica com APIs que lhes provêm dados e funcionalidades. Portanto, saber realizar esse tipo de ação no Angular é fundamental para todo desenvolvedor que utiliza esse framework.

Atualmente, a maioria das aplicações do tipo SPA (Single Page Applications) se comunicam com algum serviço no back-end por meio do protocolo HTTP, a fim de consumir dados e funcionalidades expostos por esse serviço, que, normalmente, é uma API. A aplicação envia requisições HTTP utilizando um dos seus diferentes verbos (GET, POST, PUT, DELETE etc.) e trata os seus possíveis resultados.

Para que possamos fazer esse tipo de requisição no JavaScript, dispomos, hoje, de duas APIs suportadas pelos browsers modernos: **XMLHttpRequest** (mais antiga) e a **fetch** (mais recente e que utiliza promises). E quando se trata de um projeto Angular, contamos com um mecanismo nativo deste framework que oferece uma interface simplificada para realizar essa tarefa, mas que internamente utiliza o XMLHttpRequest. Trata-se da classe **HttpClient**, disponível no módulo **HttpClientModule**.

### Primeiros Passos

Vamos treinar as requisições HTTP dentro do Angular. Para isso, vamos utilizar uma API gratuita disponibilizada na internet, a API do Github para pegar dados de usuários. Para iniciarmos, vamos importar o módulo HttpClientModule no módulo principal da nossa aplicação (AppModule)

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5 import { HttpClientModule } from '@angular/common/http'
6
7 @NgModule({
8   declarations: [
9     AppComponent
10  ],
11   imports: [
12     BrowserModule,
13     HttpClientModule
14  ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
```

Agora, com o `HttpClientModule` adicionado, vamos criar um serviço que vai fazer as requisições HTTP para a api de usuários do Github. Vamos chamar esse serviço de **GithubUsersApi**



```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class GithubUsersApiService {
7
8    constructor() { }
9  }
10
```

Agora, dentro do construtor, vamos injetar uma dependência do tipo **HttpClient**, para que possamos fazer as requisições Http a partir dela



```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class GithubUsersApiService {
8
9   constructor(
10     private http: HttpClient
11   ) { }
12 }
13
```

Crie agora um atributo que vai ter a URL base da nossa API. Isso nos ajudará a não repetir a base todas as vezes que precisarmos usar.



```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class GithubUsersApiService {
8
9   private readonly baseUrl: string = 'https://api.github.com/users/'
10
11   constructor(
12     private http: HttpClient
13   ) { }
14 }
15
```

Agora vamos criar um método no serviço que irá recuperar os dados de um usuário do Github. Para esse método, será necessário passar o nome de usuário.

```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class GithubUsersApiService {
8
9   private readonly baseUrl: string = 'https://api.github.com/users/'
10
11   constructor(
12     private http: HttpClient
13   ) { }
14
15   getUser(username: string) {
16
17   }
18 }
19
```

Métodos de um serviço que fazem requisições HTTP devem retornar um Observable para que, quem for executar esse método, realizar o subscribe nesse método. Nesse caso, vamos retornar um Observable que possui os dados de um usuário do Github. Mas... quais os dados essa API do Github retorna?

Como estamos trabalhando com o Typescript, precisamos informar que tipo de dados deverá ser retornado. Para isso, vamos criar uma interface que possui alguns campos do retorno da API do Github. Vamos chamá-la de **GithubUsersApiData** e vamos criar um arquivo para essa interface do Typescript.

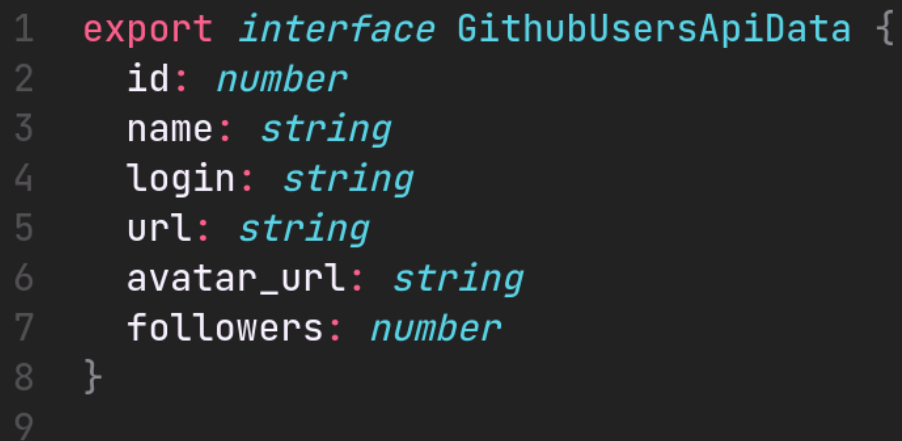
O Angular possui um comando para criar uma interface Typescript de maneira rápida. Utilize o comando:

```
ng generate interface GithubUsersApiData
```

ou

```
ng g i GithubUsersApiData
```

Agora, com a nossa interface criada, vamos colocar o nome de alguns campos que são retornados na API. LEMBRE-SE de que o nome do campo na interface deve ser idêntico ao nome ao nome do campo retornado na API. Vamos colocar os campos **id**, **login**, **url**, **avatar\_url**, **name** e **followers**



```
1  export interface GithubUsersApiData {  
2    id: number  
3    name: string  
4    login: string  
5    url: string  
6    avatar_url: string  
7    followers: number  
8  }  
9
```

Agora, vamos informar que nosso método do nosso serviço retorna um Observable do tipo GithubUsersApiData.

```

1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { GithubUsersApiData } from '../interfaces/github-users-api-data';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class GithubUsersApiService {
10
11   private readonly baseUrl: string = 'https://api.github.com/users/';
12
13   constructor(
14     private http: HttpClient
15   ) { }
16
17   getUser(username: string): Observable<GithubUsersApiData {
18
19   }
20 }
21

```

Agora, vamos fazer a lógica para retornar o Observable desse tipo

```

1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { GithubUsersApiData } from '../interfaces/github-users-api-data';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class GithubUsersApiService {
10
11   private readonly baseUrl: string = 'https://api.github.com/users/';
12
13   constructor(
14     private http: HttpClient
15   ) { }
16
17   getUser(username: string): Observable<GithubUsersApiData {
18     return this.http.get<GithubUsersApiData>(`${this.baseUrl}${username}`)
19   }
20 }
21

```

Como nós vamos fazer uma requisição HTTP do tipo GET, nós iremos chamar nossa propriedade chamada **http** e utilizar o método **get()<T>**. Esse método também precisa de um generic para saber que tipo de dado ele retornará. Nesse caso, será do tipo **GithubUsersApiData**. Após isso, vamos informar uma string com a URL de onde deve ser feita a requisição. Passamos, nesse caso, nossa propriedade **baseURL** e depois o username informado por parâmetro.

Agora, para testar, no componente principal, vamos injetar nosso serviço e, a partir do lifecycle **ngOnInit**, executar nosso método **getUser** e ver o que acontece.

```
1 import { Component } from '@angular/core';
2 import { GithubUsersApiService } from '../services/github-users-api.service';
3
4 @Component({
5   selector: 'app-root',
6   templateUrl: './app.component.html',
7   styleUrls: ['./app.component.css']
8 })
9 export class AppComponent {
10
11   constructor(
12     private gitUsersApi: GithubUsersApiService
13   ) {}
14
15   ngOnInit() {
16     this.gitUsersApi
17       .getUser('renato3x')
18       .subscribe(
19         (user) => {
20           console.log(user)
21         }
22       )
23   }
24 }
25
```

```

▼ Object ⓘ
  avatar_url: "https://avatars.githubusercontent.com/u/66842838?v=4"
  bio: "18 years old and passionate in Javascript!"
  blog: "https://linktr.ee/renato3x"
  company: null
  created_at: "2020-06-12T17:23:36Z"
  email: null
  events_url: "https://api.github.com/users/renato3x/events{/privacy}"
  followers: 86
  followers_url: "https://api.github.com/users/renato3x/followers"
  following: 28
  following_url: "https://api.github.com/users/renato3x/following{/other_user}"
  gists_url: "https://api.github.com/users/renato3x/gists{/gist_id}"
  gravatar_id: ""
  hireable: null
  html_url: "https://github.com/renato3x"
  id: 66842838
  location: "Croatá-CE"
  login: "renato3x"
  name: "Renato Pereira"
  node_id: "MDQ6VXNlcjY2ODQyODM4"
  organizations_url: "https://api.github.com/users/renato3x/orgs"
  public_gists: 1
  public_repos: 14
  received_events_url: "https://api.github.com/users/renato3x/received_events"
  repos_url: "https://api.github.com/users/renato3x/repos"
  site_admin: false
  starred_url: "https://api.github.com/users/renato3x/starred{/owner}/{/repo}"
  subscriptions_url: "https://api.github.com/users/renato3x/subscriptions"
  twitter_username: null
  type: "User"
  updated_at: "2022-02-27T20:28:13Z"
  url: "https://api.github.com/users/renato3x"
▶ [[Prototype]]: Object

```

Ao executar a sua aplicação, você verá este resultado na sua aplicação. Repare que, mesmo colocando somente alguns campos na nossa interface, todos os dados foram retornados. A interface que informamos possui apenas os campos que realmente precisamos dentro da nossa aplicação. Não é necessário colocar todos os campos, já que você pode não precisar de todos. Caso queira, crie todos os campos na sua interface. A interface dentro do Typescript é essencial para podermos acessar os dados dentro da aplicação. Sem ela, poderemos ser avisados de possíveis erros e possíveis inconsistências no código. Por isso, sempre declare uma interface a utilize para fazer as requisições HTTP.

## POST

A execução de um método POST é bastante parecido com o do método GET, mas, como o método POST é um método HTTP que utilizamos para salvar dados, precisamos passar os dados que também serão guardados. Para executarmos uma requisição do tipo POST, basta utilizarmos o método **post()** do HttpClient, onde passamos a URL de acesso e os dados que devem ser salvos



```

1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { Observable } from 'rxjs';
4
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class GithubUsersApiService {
9
10     private readonly baseUrl: string = 'https://api.github.com/users/'
11
12     constructor(
13       private http: HttpClient
14     ) { }
15
16     createUser(data: string): Observable<any> {
17       return this.http.post<any>(this.baseUrl, data)
18     }
19   }
20

```

Dessa maneira, os dados serão enviados para a API e será salvo.

**OBS:** Este exemplo é meramente ilustrativo. A API de usuários do Github nos permite apenas recuperar dados do usuário. Não é possível criar, deletar ou alterar esses dados, nem que os dados sejam seus. Esse e os próximos exemplos serão meramente utilizados de maneira didática.

## PUT

O método PUT do HTTP é utilizado para realizar atualizações de dados que estão salvos em um banco de dados. Para executarmos esse método pelo angular, a partir do HttpClient, utilizamos o método **put()**, informando a URL de acesso e os dados que devem ser salvos e atualizados.

```

1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class GithubUsersApiService {
9
10   private readonly baseUrl: string = 'https://api.github.com/users/'
11
12   constructor(
13     private http: HttpClient
14   ) { }
15
16   updateUser(username: string, data: string): Observable<any> {
17     return this.http.put<any>(`${this.baseUrl}${username}`, data)
18   }
19 }

```

Assim como no método **post()**, os dados para atualização são obrigatórios.

## DELETE

O método DELETE é responsável por excluir dados de uma API. Para executarmos ele, utilizamos o método **delete()** do HttpClient

```

1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class GithubUsersApiService {
9
10   private readonly baseUrl: string = 'https://api.github.com/users/'
11
12   constructor(
13     private http: HttpClient
14   ) { }
15
16   deleteUser(username: string): Observable<any> {
17     return this.http.delete(`${this.baseUrl}${username}`)
18   }
19 }
20

```