07

Consumindo dados da API

Transcrição

Agora que temos acesso ao `HttpClient`, vamos realizar a integração. Voltaremos a `app.component.ts` e deletaremos `console.log(http)`. Cada usuário terá um retorno de imagens, então, trabalharemos com os únicos cadastrados, `flavio` e `almeida`, este último não possuindo imagem nenhuma.

Nosso objetivo atual é acessar o endereço `http://localhost:3000/flavio/photos`, o qual nos retornará os dados da imagem no formato JSON (*JavaScript Object Notation*) para serem colocados na propriedade `photos`. Será retornado um `observable`, e precisaremos da notificação do mesmo informando que os dados provenientes do servidor chegaram.

```
export class AppComponent {

    photos = [];

    constructor(http:HttpClient) {

        const observable = http.get('http://localhost:3000/fla
        observable.subscribe();
    }
}
```

COPIAR CÓDIGO

Costumamos dizer que um `observable` é *lazy* (preguiçoso), pois só vai buscar os dados se tiver alguém inscrito nele (`observable.subscribe()`). Mas para não termos que ficar declarando esta variável, podemos encadear uma chamada diretamente, solicitando o `get()` e então um `subscribe()` .

```
constructor(http:HttpClient) {

    http
        .get('http://localhost:3000/flavio/photos')
        .subscribe();
}
```

COPIAR CÓDIGO

O Angular é um *framework* que utiliza TypeScript, linguagem criada pela Microsoft, que usa o RxJS — e o `observable` vem dele —, tecnologia muito poderosa para lidar com operações assíncronas. Por ser mais rico, no Angular 2 preferiram utilizar o *Observable* em vez de *Promise*, por padrão. Perceberemos isso ao longo do curso.

Se isso der certo, serão retornados os dados que poderemos capturar em uma *function*, no entanto optaremos por *arrow function* de nome `photos` , que é o que chega via back end. E então, a propriedade `this.photos` receberá o resultado de `photos` .

```
http
    .get('http://localhost:3000/flavio/photos')
    .subscribe(photos => this.photos = photos);
```

COPIAR CÓDIGO

Isto, porém, resulta em erro de compilação, já que o resultado de tipo `Object` não é compatível com o tipo `any[]` . Quando não explicitamos o tipo de uma

variável em JavaScript, adota-se o tipo `any[]`, ou seja, "qualquer". Portanto, indicaremos que `photos` será do tipo `Object[]`:

```
export class AppComponent {

    photos: Object[] = [];

    constructor(http:HttpClient) {

        http
            .get('http://localhost:3000/flavio/photos')
            .subscribe(photos => this.photos = photos);
    }
}
```

COPIAR CÓDIGO

Para melhor aproveitamento destas aulas, são recomendados os cursos de TypeScript, partes 1 (https://cursos.alura.com.br/course/typescript-parte1) e 2 (https://cursos.alura.com.br/course/typescript-parte2).

Entretanto, o problema persiste, pois `Object` não pertence ao tipo `Object[]`. Nós sabemos que trata-se de uma lista que vem do back end, mas o TypeScript não sabe disso. Para consertar, incluiremos `<Object[]>` no código, e com isso os tipos começam a fazer sentido entre si.

```
http
    .get<Object[]>('http://localhost:3000/flavio/photos')
    .subscribe(photos => this.photos = photos);
```

COPIAR CÓDIGO

Salvaremos, voltaremos ao navegador, e teremos várias imagens diferentes sendo exibidas, provenientes do back end. Para confirmar o bom

funcionamento, podemos abrir um bloco na *arrow function*, usar o `console.log()`, salvar o arquivo, abrir a página no navegador, e depois o console, e teremos uma lista em que cada imagem possui suas respectivas informações.

```
http
    .get<Object[]>('http://localhost:3000/flavio/photos')
    .subscribe(photos => {
        console.log(photos);
        this.photos = photos
    });
```

COPIAR CÓDIGO

E se houver erro, de que forma o trataremos?

O `subscribe()` recebe dois parâmetros, sendo o primeiro deles o *callback* a ser chamado caso haja sucesso, e o segundo a ser chamado passando o erro que vem do back end:

```
http
    .get<Object[]>('http://localhost:3000/flavio/photosx')
    .subscribe(
        photos => this.photos = photos,
        err => console.log(err)
    );
```

COPIAR CÓDIGO

Salvaremos com um endereço que não existe, nada será exibido no navegador, e no console se verifica o tipo de erro, um *HttpErrorResponse*. No trecho acima, ainda poderemos incluir `err.message` em `console.log()` para que o retorno fique mais sucinto:

> Http failure response for http://localhost:3000/flavio/photosx (http://localhost:3000/flavio/photosx): 404 Not Found

Mas por enquanto não lidaremos com tratamento de erros, então voltaremos o código para o que tínhamos feito antes.

```
http
    .get<Object[]>('http://localhost:3000/flavio/photos')
    .subscribe(photos => this.photos = photos);
```

COPIAR CÓDIGO