09

Tipando nossa API

Transcrição

Organizamos o código de maneira mais elegante, e o isolamos em uma camada de serviço. Conseguimos exibir as fotos, mas avançaremos um pouco e adquiriremos maturidade em termos de acesso à API. Quando a acessamos via navegador, recebemos uma lista de objetos com propriedades `id`, `postDate`, URL, e outras tantas informações que dizem respeito à cada imagem.

Por ora, só estamos utilizando `url` e `description`, certo? Para provar isso, em `app.component.ts`, criaremos um bloco na *arrow function* e, no *array* de `photos`, queremos acessar o `id` da imagem da primeira posição. Mas ao digitarmos `photos[0].`, não são exibidas opções de *autocomplete*. Se tentarmos incluir `url` logo em seguida, teremos um erro de compilação.

Quando o código está sendo executado, internamente ele roda como JavaScript, sem tipagem, e por isso funciona. Porém, do ponto de vista do TypeScript, esta lista é do tipo `Object[]`, o qual não possui `url` ou outras propriedades além de `constructor`, `hasOwnProperty()`, `isPrototypeOf()`, e por aí vai.

Sendo assim, não é possível exibir os dados da primeira imagem, pois utilizamos esta tipagem (`Object[]`). Vamos tentar trocá-la por `any[]`:

```
export class AppComponent {

    photos: any[] = [];
```

```
    constructor(photoService: PhotoService) {

        photoService
            .listFromUser('flavio')
            .subscribe(photos => {
                photos[0].
                this.photos = photos
            });
}
```

COPIAR CÓDIGO

Com a tecla "Ctrl" pressionada, e clicando em `listFromUser()`, acessamos o método, e indicaremos que seu retorno será do tipo `any[]`:

```
listFromUser(userName: string) {
    return this.http
        .get<any[]>(API + '/flavio/photos');
}
```

COPIAR CÓDIGO

Voltaremos a `app.component.ts`, e o *autocomplete* em `photos[0].description` continua não funcionando, mas o seu acesso sim. Ou seja, não há mais erro de compilação. E como usamos `any[]`, o tipo poderá ser qualquer um. Incluiremos um `console.log()` no código, salvaremos, voltaremos ao navegador, abriremos o console, em que encontraremos "Farol iluminado", descrição da primeira imagem.

```
photoService
    .listFromUser('flavio')
    .subscribe(photos => {
        console.log(photos[0].description);
        this.photos = photos
    });
```

COPIAR CÓDIGO

Resolvemos o nosso problema? Mais ou menos, porque se digitarmos `descript` em vez de `description`, o TypeScript não acusará erro, e no console será lido simplesmente "undefined". Além disso, também perdemos o *autocomplete* ao desenvolvermos a aplicação.

Para corrigirmos isto, tiparemos o retorno da API. Em `photo.service.ts` informamos que o retorno é do tipo `any[]`, porém trocaremos para `Photo[]`, com propriedades específicas. Então, na pasta "photo", junto ao serviço, criaremos o arquivo `photo.ts`, que não será um componente, tampouco uma classe, e sim uma interface chamada `Photo`.

No conceito do TypeScript, a ideia da interface tem a ver com encaixes específicos, e incluiremos nela todas as propriedades dos objetos retornados na lista da API, e atribuiremos um tipo para cada uma delas:

```
export interface Photo {
    id:number;
    postDate:Date;
    url:string;
    description:string;
    allowComments:boolean;
    likes:number;
    comments:number;
    userId:number;
}
```

COPIAR CÓDIGO

> A interface não diz em nenhum momento quais dados precisam estar em cada uma destas propriedades, e sim o *shape*, a forma que um objeto deve ter.

Em `photo.service.ts`, importaremos `Photo[]` clicando no ícone de lâmpada, e o dado será tratado como um *array* deste tipo. Nisto, `app.component.ts`

começará a dar erro, pois quando acessamos a posição `0` do *array*, veremos que não existe a propriedade `descript`.

Com isso, ficamos menos suscetíveis a erros corriqueiros ao acessarmos estas propriedades, pois as padronizamos e tipamos, e somos mais produtivos com o *autocomplete*. Caso a API mude, a aplicação deixará de funcionar, pois o TypeScript não irá prevenir isso, apenas receberá os dados e tentará tratá-los, sem conseguir acessá-los.

Se o acesso do retorno da API ocorre em inúmeros locais, poderemos abrir a interface, clicar com o botão direito na propriedade alterada, escolher a opção "Rename Symbol", digitar o novo nome e pressionar "Enter". Assim, modificamos o nome da propriedade em todos os locais em que é acessada.

O código de `photoService` em `app.component.ts` ficará da seguinte forma:

```
photoService
    .listFromUser('flavio')
    .subscribe(photos => this.photos = photos);
```

COPIAR CÓDIGO