02

Consumindo API externa

Transcrição

Para consumirmos a API externa, utilizaremos a API `fetch` que usa o padrão de projeto `Promise`. Por usar `Promise`, seu uso é mais simplificado do que trabalharmos com `XMLHttpRequest`.

```
// app/ts/controllers/NegociacaoController.js

    importarDados() {

        fetch('http://localhost:8080/dados')

    }
```

Como `fetch` faz parte do ECMASCRIPT, o TypeScript é capaz de validar a sintaxe que utilizaremos, pois ele já possui um type definition para ele. No curso de JavaScript avançado da Alura, no terceiro curso, a Fetch API é abordada com seus pormenores, mas uma explicação breve para relembrar ou introduzir para quem nunca a viu a API pode acelerar o tempo de construção do nosso projeto.

```
// app/ts/controllers/NegociacaoController.js

    importarDados() {

        fetch('http://localhost:8080/dados')
            .then(res => res.json())
    }
```

Através da chamada da função `then` temos acesso à resposta que precisa ser convertida (parse) adequadamente e a Fetch API já traz na própria resposta o método `.json()` que realiza essa conversão de JSON para objetos em JavaScript. Como usamos arrow function sem bloco, o resultado da instrução `res.json()` é retornado automaticamente sem a necessidade de usarmos um `return` e quando fazemos isso, temos acesso ao retorno na próxima chamada encadeada à função `then`.

Contudo, pode ser que o servidor execute corretamente nossa operação (sem devolver erro 500, por exemplo), mas devolva algum código de status que indique erro, dessa forma, em vez de ficarmos acessando cada código em separadamente podemos acessar `status.ok` para saber se a resposta é válida com um status válido.

Vamos isolar a lógica do teste em uma função que retorna `res` caso esteja tudo correto ou lance uma exceção caso haja algum problema. No caso, podemos usar o tipo `Response` para res, garantindo assim a checagem do TypeScript evitando o acesso a métodos que não existam, seja por um momento lacônico nosso ou por um erro de digitação:

```
// app/ts/controllers/NegociacaoController.js

    importarDados() {

        function isOK(res: Response) {
```

```
            if(res.ok) {
                return res;
            } else {
                throw new Error(res.statusText);
            }
        }

        fetch('http://localhost:8080/dados')
            .then(res => isOK(res))
            .then(res => res.json())
    }
```

Agora, encadeando uma nova chamada ao `then`, temos acesso aos dados parseados:

```
        fetch('http://localhost:8080/dados')
            .then(res => isOK(res))
            .then(res => res.json())
            .then(dados => {

            })
```

Mas o TypeScript reclamará em breve que há um tipo implícito any sendo empregado pelo nosso código. Resolvemos isso explicitando que o tipo retornado é `any[]`, um array de qualquer tipo:

```
        fetch('http://localhost:8080/dados')
            .then(res => isOK(res))
            .then(res => res.json())
            .then((dados: any[]) => {

            })
```

Agora basta convertemos os dados recebidos em uma lista de negociações e adicioná-las na tabela. Não podemos nos esquecer de chamarmos o método `update` da view que representa nossa tabela:

```
        fetch('http://localhost:8080/dados')
            .then(res => isOK(res))
            .then(res => res.json())
            .then((dados: any[]) => {
                dados
                    .map(dado => new Negociacao(new Date(), dado.vezes, dado.montante))
                    .forEach(negociacao => this._negociacoes.adiciona(negociacao));
                this._negociacoesView.update(this._negociacoes);
            })
```

Por fim, vamos adicionar um `catch` para lidar com qualquer erro que possa ter ocorrido em nossa requisição assíncrona:

```
    importarDados() {

        function isOK(res: Response) {
```

```
            if(res.ok) {
                return res;
            } else {
                throw new Error(res.statusText);
            }
        }

        fetch('http://localhost:8080/dados')
            .then(res => isOK(res))
            .then(res => res.json())
            .then((dados: any[]) => {
                dados
                    .map(dado => new Negociacao(new Date(), dado.vezes, dado.montante))
                    .forEach(negociacao => this._negociacoes.adiciona(negociacao));
                this._negociacoesView.update(this._negociacoes);
            })
            .catch(err => console.log(err.message));
    }
```

Excelente, se clicarmos no botão `importar` conseguimos importar todos os dados. Lembre-se que podemos adotar `new Date()` como data dos dados recebidos, pois eles não definem a propriedade data.

Apesar de funcional nosso código deixa a desejar. É isso que veremos no próximo vídeo.