01

Somando todos os itens com código específico

Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/772-js-padrao-funcional/stages/02-project.zip) completo do projeto anterior e continuar seus estudos a partir daqui.

Somos capazes de buscar uma lista de notas fiscais, excelente. Todavia, precisamos totalizar o valor dos itens com o código `2143` de todas as notas.

Vamos acessar através do navegador o recurso `http://localhost:3000/notas`. Receberemos um JSON com esta estrutura:

```
[
    {
        data: '2017-10-31',
        itens: [
            { codigo: '2143', valor: 200 },
            { codigo: '2111', valor: 500 }
        ]
    },
    {
        data: '2017-07-12',
        itens: [
            { codigo: '2222', valor: 120 },
            { codigo: '2143', valor: 280 }
        ]
    },
    {
        data: '2017-02-02',
        itens: [
            { codigo: '2143', valor: 110 },
            { codigo: '7777', valor: 390 }
        ]
    },
];
```

Olhando a estrutura de dados retornada e realizando o cálculo mentalmente, o valor total de todos os itens com código `2143` será R$ 590,00. Realizar o cálculo mentalmente não vale, precisamos realizá-lo programaticamente. Mas antes de partirmos para a implementação do código, vamos fazer uma checklist do que precisamos realizar:

1. obter uma lista com todos os itens de todas as notas fiscais
2. filtrar todos os itens pelo código 2143
3. somar o valor de todos os itens

Em JavaScript, podemos realizar cada etapa acima da seguinte maneira:

1. Array.map
2. Array.filter

3. Array.reduce

Agora podemos partir para a implementação:

```
// app/app.js
import { handleStatus } from './utils/promise-helpers.js';

document
.querySelector('#myButton')
.onclick = () =>
    fetch('http://localhost:3000/notas')
    .then(handleStatus)
    // retornará para o próximo then uma lista de itens
    .then(notas => notas.map(nota => nota.itens))
    // retornará para o próximo then uma lista de itens filtrada
    .then(itens => itens.filter(item => item.codigo = '2143'))
    // retornará para o próximo then o total
    .then(itens => itens.reduce((total, item) => total + item.valor, 0))
    // exibe o resultado
    .then(total => console.log(total))
    .catch(console.log);
```

Como só temos interesse em exibir o total no console, podemos simplificar a chamada da mesma maneira que fizemos com o erro capturado em `catch`:

```
// app/app.js
import { handleStatus } from './utils/promise-helpers.js';

document
.querySelector('#myButton')
.onclick = () =>
    fetch('http://localhost:3000/notas')
    .then(handleStatus)
    .then(notas => notas.map(nota => nota.itens))
    .then(itens => itens.filter(item => item.codigo = '2143'))
    .then(itens => itens.reduce((total, item) => total + item.valor, 0))
    // simplificando
    .then(console.log)
    .catch(console.log);
```

Apesar dos nossos esforços, o resultado será `0`. Onde está o erro?

O problema está no mapeamento do array de notas fiscais para um array de itens. Vamos alterar temporariamente nosso código para podermos escrutinar o valor retornado `then` que realiza o map:

```
// app/app.js
import { handleStatus } from './utils/promise-helpers.js';

document
.querySelector('#myButton')
    .onclick = () =>
    fetch('http://localhost:3000/notas')
    .then(handleStatus)
```

```
    .then(notas => notas.map(nota => nota.itens))
    // then extra!
    .then(itens => {
        console.log(itens);
        return itens;
    })
    .then(itens => itens.filter(item => item.codigo == '2143'))
    .then(itens => itens.reduce((total, item) => total + item.valor, 0))
    .then(console.log)
    .catch(console.log);
```

Veremos como saída do console:

```
[Array(2), Array(2), Array(2)]
```

Não temos um array de única dimensão com todos os itens, temos um array multidimensional onde cada array corresponde a lista de itens de cada nota. Nesse caso, o `map` funcionou, mas não da maneira que estávamos esperando. E agora?

Precisamos reduzir todos os itens do array para uma dimensão apenas. Podemos fazer isso trocando map por reduce:

```
// app/app.js
import { handleStatus } from './utils/promise-helpers.js';

document
.querySelector('#myButton')
.onclick = () =>
    fetch('http://localhost:3000/notas')
    .then(handleStatus)
    .then(notas => notas.reduce((array, nota) => array.concat(nota.itens), []))
    .then(itens => {
        // temos agora um array de única dimensão
        console.log(itens);
        return itens;
    })
    .then(itens => itens.filter(item => item.codigo == '2143'))
    .then(itens => itens.reduce((total, item) => total + item.valor, 0))
    .then(console.log)
    .catch(console.log);
```

Conseguimos chegar ao resultado R$ 590,00. Podemos até visualizar a lista filtrada antes dela chegar ao reduce:

```
// app/app.js
import { handleStatus } from './utils/promise-helpers.js';

document
.querySelector('#myButton')
.onclick = () =>
    fetch('http://localhost:3000/notas')
    .then(handleStatus)
    .then(notas => notas.reduce((array, nota) => array.concat(nota.itens), []))
    .then(itens => {
        console.log(itens);
        return itens;
```

```
})
.then(itens => itens.filter(item => item.codigo == '2143'))
.then(itens => {
    // nova lista filtrada
    console.log(itens);
    return itens;
})
.then(itens => itens.reduce((total, item) => total + item.valor, 0))
.then(console.log)
.catch(console.log);
```

Tudo funcionando, porém nosso código deixa a desejar em dois pontos. No próximo vídeo atacaremos o primeiro, para no final atacarmos o segundo.