01

O problema da vida assíncrona

Transcrição

Você pode fazer o download (https://github.com/alura-cursos/javascript-avancado-ii/archive/aula4.zip) completo do projeto até aqui e continuar seus estudos.

O nosso código está funcionando, conseguimos recarregar e importar as negociações ao clicar no botão correspondente. No entanto, precisamos encontrar uma forma de importar as negociações da semana atual, da passada e da retrasada para popularmos a lista.

Para isto, copiaremos a lógica do método `obterNegociacoesDaSemana()`, no `NegociacaoService`, e modificaremos o nome para `obterNegociacaoDaSemanaAnterior`:

```
obterNegociacoesDaSemanaAnterior(cb) {

      let xhr = new XMLHttpRequest();
      xhr.open('GET', 'negociacoes/anterior');
      xhr.onreadystatechange = () => {
          if(xhr.readyState == 4) {
              if(xhr.status == 200) {

                cb(null, JSON.parse(xhr.responseText)
                      .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantidad

              } else {
                  console.log(xhr.responseText);
                  cb('Não foi possível obter as negociações da semana', null);
              }
          }
      }

      xhr.send();
  }
}
```

Observe que alteramos o endereço para `negociacoes/anterior`. Faremos algo parecido com o trecho referente à semana **retrasada**:

```
obterNegociacoesDaSemanaRetrasada(cb) {

      let xhr = new XMLHttpRequest();
      xhr.open('GET', 'negociacoes/retrasada');
      xhr.onreadystatechange = () => {
          if(xhr.readyState == 4) {
              if(xhr.status == 200) {

                cb(null, JSON.parse(xhr.responseText)
                      .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantida
```

```
                } else {
                    console.log(xhr.responseText);
                    cb('Não foi possível obter as negociações da semana', null);
                }
            }
        }

        xhr.send();
    }
}
```

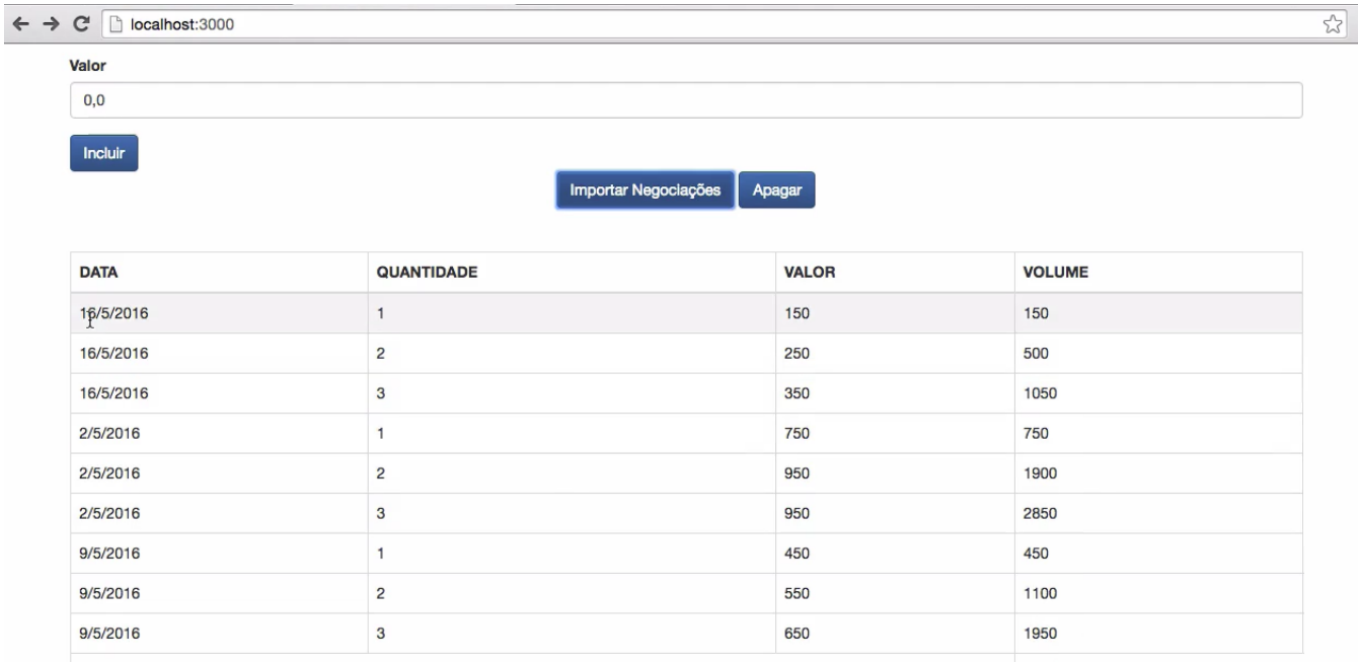
Nós criamos três métodos que irão obter os dados da semana atual, passada e retrasada. Agora, no arquivo `NegociacaoController.js`, já temos o `obterNegociacoesDaSemana` em `importaNegociacoes`:

```
importaNegociacoes() {

    let service = new NegociacaoService();

    service.obterNegociacoesDaSemana((erro, negociacoes) => {
        if(erro) {
            this._mensagem.texto = erro;
            return;
        }

        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações importadas com sucesso';
    });
}
```

Agora, adicionaremos `obterNegociacoesDaSemanaAnterior` e `obterNegociacoesDaSemanaRetrasada`:

```
    service.obterNegociacoesDaSemanaAnterior((erro, negociacoes) => {
        if(erro) {
            this._mensagem.texto = erro;
            return;
        }

        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações importadas com sucesso';
    });

    service.obterNegociacoesDaSemanaRetrasada((erro, negociacoes) => {
        if(erro) {
            this._mensagem.texto = erro;
            return;
        }

        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações importadas com sucesso';
    });
```

Vamos realizar as três chamadas do serviço. Depois, recarregaremos a página e conseguiremos que os dados sejam importados corretamente.

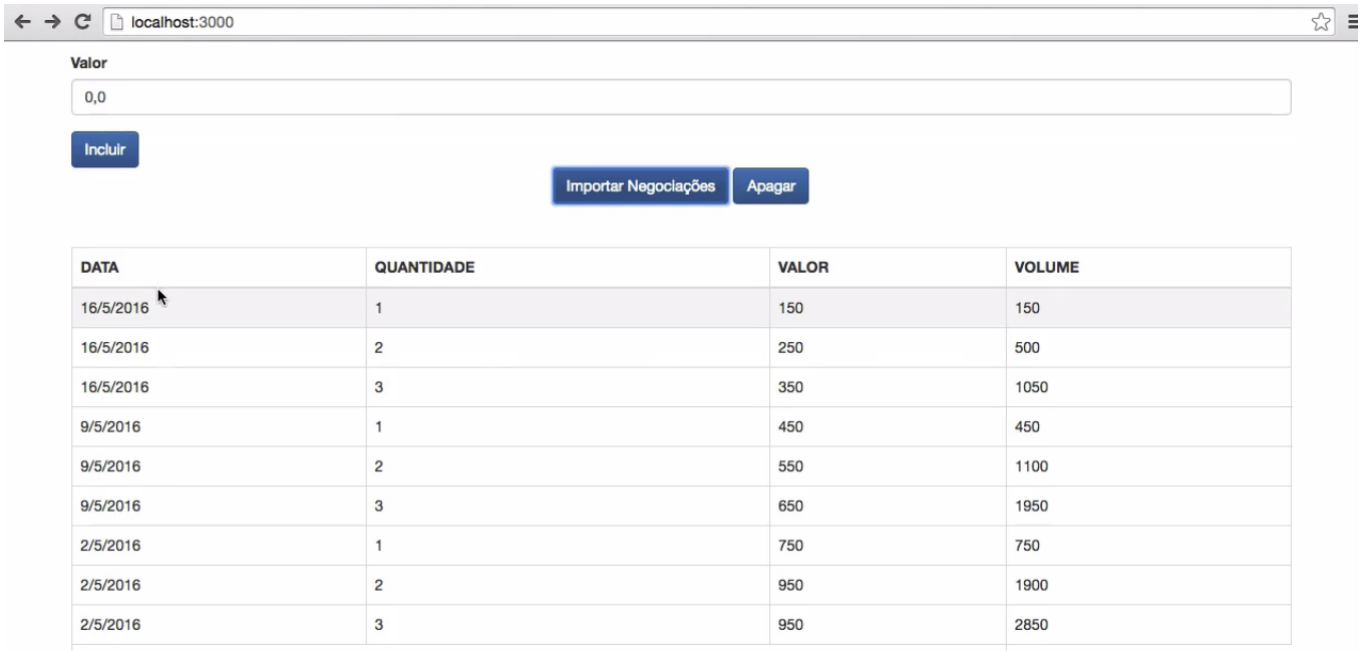


Mas temos um pequeno problema... Como as requisições são assíncronas, elas não esperam a operação terminar para só então começarem a executar a próxima.. Segundo uma regra de negócios, essas importações devem estar em ordem: primeiramente, a negociações desta semana, depois, a da semana anterior. Por último, teremos na semana retrasada. Mas se analisarmos as datas da tabela, veremos que elas não estão em ordem. Nós não temos os controles das datas, e como são assíncronas, cada requisição poderá ser executada em diferentes momentos. O que é preciso fazer para que todas sejam executadas em ordem? Começaremos executando a requisição da semana.

```
importaNegociacoes() {

    let service = new NegociacaoService();

    service.obterNegociacoesDaSemana((erro, negociacoes) => {

        if(erro) {
            this._mensagem.texto = erro;
            return;
        }
        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));

        service.obterNegociacoesDaSemanaAnterior((erro, negociacoes) => {

            if(erro) {
                this._mensagem.texto = erro;
                return;
            }
            negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));

            service.obterNegociacoesDaSemanaRetrasada((erro, negociacoes) => {

                if(erro) {
                    this._mensagem.texto = erro;
                    return;
```

```
                }
                negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociaca
                this._mensagem.texto = 'Negociações importadas com sucesso';
            });
        });
    });
}
```

Quando a requisição de `obterNegociacoesDaSemana` terminar, as negociações serão adicionadas na lista. Então, teremos certeza de que a requisição está terminada e que podemos chamar a requisição, referente à semana anterior. E por último, virão as da semana retrasada. Vamos testar se as requisições são executadas na ordem correta.



DATA	QUANTIDADE	VALOR	VOLUME
16/5/2016	1	150	150
16/5/2016	2	250	500
16/5/2016	3	350	1050
9/5/2016	1	450	450
9/5/2016	2	550	1100
9/5/2016	3	650	1950
2/5/2016	1	750	750
2/5/2016	2	950	1900
2/5/2016	3	950	2850

Agora, as datas estão em ordem.

> Quando você for fazer o teste na sua máquina, o servidor está gerando essas negociações aleatoriamente e podem aparecer outras datas.

Conseguimos resolver o nosso problema, mas se analisarmos o "desenho" do código, veremos que ele forma uma **pirâmide**. Costumamos dar o nome de ***Pyramid of Doom*** (traduzida para o português, significa **pirâmide do destino**) em situações que isto ocorre e temos uma função aninhada dentro de outra. A pirâmide é um forte indício de que temos problemas de legitimidade do código, na verdade, é o sintoma de um problema maior, o ***Callback Hell***. Ocorre quando temos requisições assíncronas executadas em determinada ordem, que chama vários callbacks seguidos.

Se tivéssemos mais ações que precisassem ser executadas em ordem, teríamos um pirâmide com mais funções. Também vale ressaltar: em uma situação de erro - por exemplo, se a URL estivesse equivocada -, dentro do `importaNegociacoes`, executaríamos diversos trechos de código referentes ao erro. Testaríamos diversas vezes se ocorreu o erro, porque o código está repetido.

Agora, aplicaremos um padrão de projeto que nos ajudará a lidar com a complexidade da programação assíncrona, além de evitar que o erro seja tratado em diversos lugares.