02

O padrão de projeto Promise

Transcrição

Vamos melhorar a legibilidade do código, que está repetindo o tratamento de erro:

```
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(negociacao));
                this._mensagem.texto = 'Negociações importadas com sucesso';
            });
        });
    });
}
```

Em `NegociacaoService.js`, faremos pequenos ajustes nas mensagens de erros dos métodos de `obterNegociacoesDaSemana()`, `obterNegociacoesDaSemanaAnterior()` e `obterNegociacoesDaSemanaRetrasada`, especificando qual semana está sendo trabalhada.

```
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.quantidade, ol

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

    xhr.send();
  }
}
```

Desta forma, se tivermos problemas, saberemos que será na semana anterior. Queremos reduzir a complexidade do código de programação assíncrona. Para isto, aplicaremos um padrão de projeto chamado **Promessa** (*Promise*, em inglês).

Vamos comentar o trecho que criamos até agora, porque faremos um antes e depois. Enquanto trabalhamos com o padrão *Promise*, não implementaremos ainda o `NegociacaoService`. Mas vamos considerar que ele usará o padrão de projeto Promise. Veremos a seguir, como usaremos o `NegociacaoService()`, no `NegociacaoController.js`:

```
importaNegociacoes() {

  let service = new NegociacaoService();

  let retorno = service.obterNegociacoesDaSemana();
}
```

Se observarmos o método `obterNegociacoesDaSemana()`, veremos que este não recebe mais o callback - apenas nos devolverá um valor. Parecerá ser um método **síncrono**. No entanto, ele não é... Porque ele não devolverá a lista de negociações, mas, sim, uma *promise* - que não poderá encontrar o que busca. A promessa é o resultado futuro de uma operação. Quando pensamos no conceito de uma promessa, nos vem a ideia de que "se você cumprir a promessa, **então** algo irá acontecer...". Seguindo está relação com **então**, chamaremos o método `then()` na *promise*.

```
importaNegociacoes() {

  let service = new NegociacaoService();

  let promise = service.obterNegociacoesDaSemana();
  promise
      .then(negociacoes => {
          negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao))
          this._mensagem.texto = 'Negociação da semana obtida com sucesso';
      });
```

Se a promessa for cumprida, receberemos a lista de negociação e, com esta, poderemos fazer o `forEach()`. O método `obterNegociacoesDaSemana()` devolve uma promessa de que tentará obter os dados. Caso a promessa seja cumprida, receberemos uma lista de negociações e exibiremos a mensagem para o usuário.

Para o caso de ocorrer um erro, vamos encadear uma função `catch()`, na `promise`.

```
.catch(erro => this.mensagem.texto = erro);
```

Com ela, iremos capturar o erro da `promise` e exibiremos para o usuário.

No entanto, se tentarmos executar o nosso código como está, receberemos várias mensagens de erro no Console.

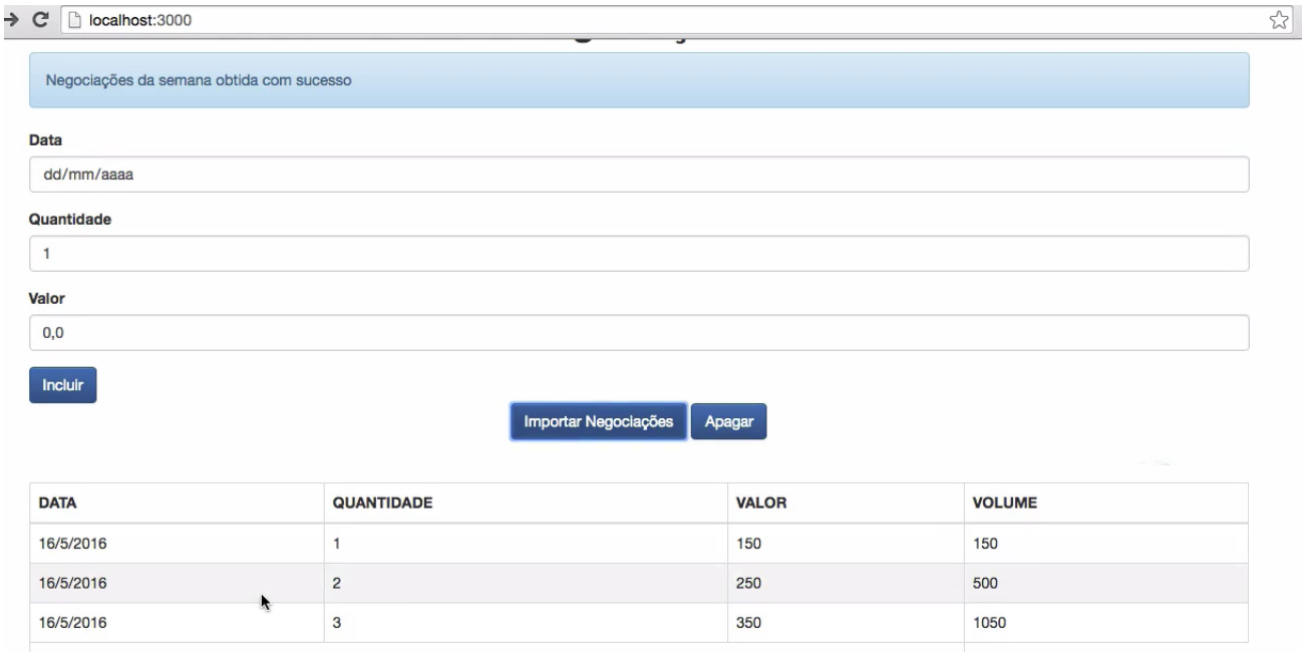
Ele nos informa que não recebe `then()`, porque `obterNegociacoesDaSemana()` não é uma `promise`. Precisamos fazer esta transformação. O ES6 suporta a promise nativamente, então, o método deverá retornar uma `Promise()`, que receberá dois parâmetros (`resolve` e `reject`). Em que momento sabemos que os dados são retornados? É onde temos o `cb`, que não será mais necessário, por isso, vamos substitui-los por `resolve`:

```
class NegociacaoService {

  obterNegociacoesDaSemana() {

    return new Promise((resolve, reject) => {
        let xhr = new XMLHttpRequest();
        xhr.open('GET', 'negociacoes/semana');
        xhr.onreadystatechange = () => {
            if(xhr.readyState == 4) {
                if(xhr.status == 200) {

                  resolve(JSON.parse(xhr.responseText)
                        .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantidade,

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

                }
            }
        };
        xhr.send();

    });
```

O `resolve` passará diretamente o resultado de `JSON.parse()`. Observe que não precisaremos mais utilizar o parâmetro (`null`) do *Error-First-Callback*. Nós passamos direto para o `resolve` o resultado que será disponibilizado para a função `then`. E se tivermos algum tipo de erro, chamaremos a função `reject()`. O que está no `resolve`, pegaremos dentro do método `then()` do arquivo `NegociacaoController.js` e o erro passado para o `reject`, pegaremos no `catch`.

```
importaNegociacoes() {

    let service = new NegociacaoService();

    let promise = service.obterNegociacoesDaSemana();
    promise
        .then(negociacoes => {
        negociacoes.forEach(negociacao => {
            this._listaNegociacoes.adiciona(negociacao);
        });
        this._mensagem.texto = 'Negociações importadas com sucesso'
    })
    .catch(erro => this._mensagem.texto = erro);
}
```

Se testarmos no navegador, veremos que tudo está funcionando e conseguiremos importar as negociações corretamente.



Aplicaremos o padrão Promise nos outros métodos do `NegociacaoService.js` .

```
obterNegociacoesDaSemanaAnterior() {

        return new Promise((resolve, reject) => {

            let xhr = new XMLHttpRequest();

            xhr.open('GET', 'negociacoes/anterior');
            xhr.onreadystatechange = () => {
                if(xhr.readyState == 4) {
                    if(xhr.status == 200) {
                        resolve(JSON.parse(xhr.responseText)
                            .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantida

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

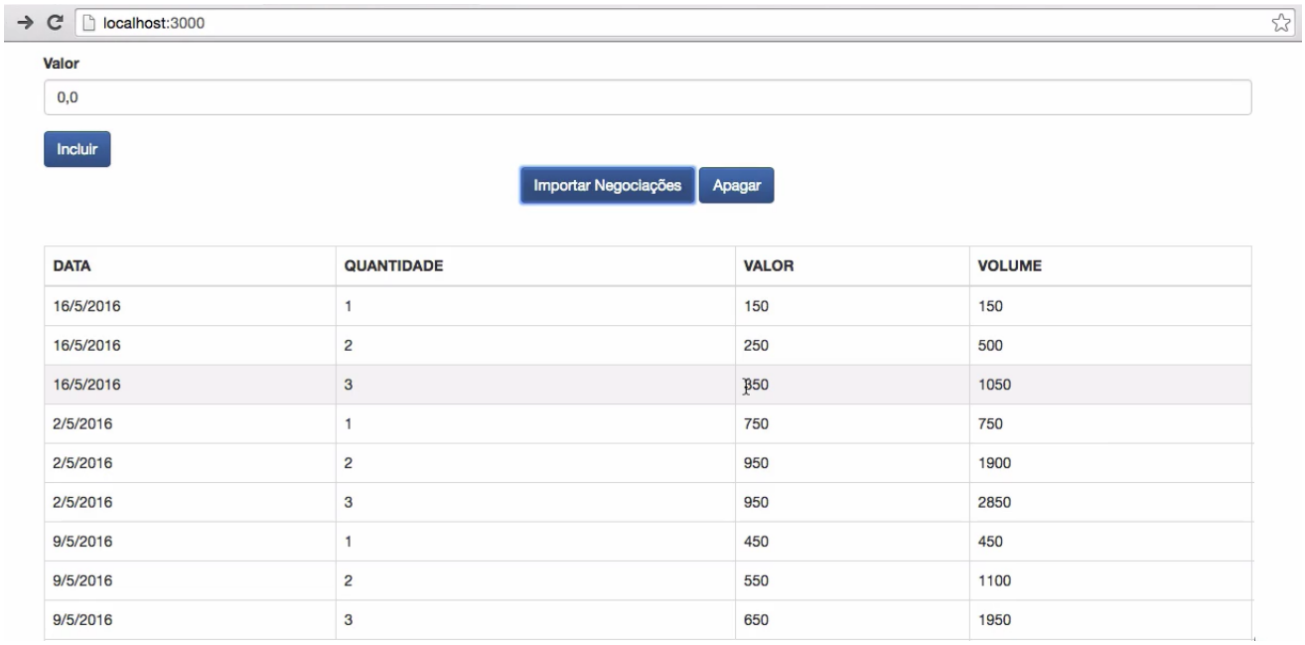
```
                    }
                }
            }
            xhr.send();
        });
    }

    obterNegociacoesDaSemanaRetrasada() {

        return new Promise((resolve, reject) => {

          let xhr = new XMLHttpRequest();

            xhr.open('GET', 'negociacoes/retrasada');
            xhr.onreadystatechange = () => {
                if(xhr.readyState == 4) {
                    if(xhr.status == 200) {
                      resolve(JSON.parse(xhr.responseText)
                          .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantidade

                    } else {
                        console.log(xhr.responseText);
                        reject('Não foi possível obter as negociações da semana retrasada');
                    }
                }
            }
            xhr.send();
        });
    }
```

Em seguida, no `NegociacoesController.js`, não iremos mais declarar a variável `Promise`:

```
importaNegociacoes() {

    let service = new NegociacaoService();

    service.obterNegociacoesDaSemana()
      .then(negociacoes => {
        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações da semana obtidas com sucesso';
    })
    .catch(erro => this._mensagem.texto = erro);

    service.obterNegociacoesDaSemanaAnterior()
      .then(negociacoes => {
        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações da semana obtidas com sucesso';
    })
    .catch(erro => this._mensagem.texto = erro);

    service.obterNegociacoesDaSemanaRetrasada()
      .then(negociacoes => {
        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações da semana obtidas com sucesso';
    })
```

```
        .catch(erro => this._mensagem.texto = erro);

    }
```

No navegador, dessa vez, teremos problemas com as negociações importadas. Novamente, as negociações estão fora da ordem.



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

A ordem de execução das Promises está incorreta. Mais adiante, entenderemos por que isso aconteceu e descobriremos a solução.