04

Separando as responsabilidades

Transcrição

Estamos conseguindo importar, mas a responsabilidade de buscar os dados do servidor não é da `controller`. Por isso, nós vamos isolar o código do `importaNegociacoes()`, numa classe que será especializada em obter as negociações do servidor e que será utilizada pela `controller`. Dentro da pasta `services`, vamos criar `NegociacaoService.js`.

```
importaNegociacoes() {

        let xhr = new XMLHttpRequest();

         xhr.open('GET', 'negociacoes/semana');

         xhr.onreadystatechange = () => {

             if(xhr.readyState == 4) {

                 if(xhr.status == 200) {

                   JSON.parse(xhr.responseText)
                     .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantidade, obje
                     .forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
                     this._mensagem.texto = 'Negociações importadas com sucesso.';

                 } else {
                     console.log(xhr.responseText);
                     this._mensagem.texto = 'Não foi possível obter as negociações.';
                 }
             }
       };

       xhr.send();
    }
}
```

A vantagem de isolarmos o código é que se tivermos outra parte do sistema que precisa obter a lista de negociações do servidor, não iremos cortar o código da `controller`. Nós reutilizaremos em uma nova classe `NegociacaoService`, que terá o método `obterNegociacoesDaSemana()` que retornará as negociações da semana:

```
class NegociacaoService {
    obterNegociacoesDaSemana() {

        let xhr = new XMLHttpRequest();

        xhr.open('GET', 'negociacoes/semana');

        xhr.onreadystatechange = () => {

            if(xhr.readyState == 4) {
```

```
                if(xhr.status == 200) {

                JSON.parse(xhr.responseText)
                    .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantidade, obje
                    .forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
                    this._mensagem.texto = 'Negociações importadas com sucesso.';

            } else {
                console.log(xhr.responseText);
                this._mensagem.texto = 'Não foi possível obter as negociações.';
            }
            }
    };

    xhr.send();
    }
}
```

O service não terá acesso a View, porque ele não tem referência para os elementos da `controller`. Logo, removemos `this._mensagem.texto` a mensagem de erro para o usuário. E importaremos o arquivo em `index.html`:

```
<script src="js/app/models/Negociacao.js"></script>
<script src="js/app/models/ListaNegociacoes.js"></script>
<script src="js/app/models/Mensagem.js"></script>
<script src="js/app/controllers/NegociacaoController.js"></script>
<script src="js/app/helpers/DateHelper.js"></script>
<script src="js/app/views/View.js"></script>
<script src="js/app/views/NegociacoesView.js"></script>
<script src="js/app/views/MensagemView.js"></script>
<script src="js/app/services/ProxyFactory.js"></script>
<script src="js/app/helpers/Bind.js"></script>
<script src="js/app/services/NegociacaoService.js"></script>
<script>
    let negociacaoController = new NegociacaoController();
</script>
```

Em seguida, no arquivo `NegociacaoController.js`, vamos adicionar a variável `service` no `importaNegociacoes()`.

```
importaNegociacoes() {

    let service = new NegociacaoService();

    service.obterNegociacoesDaSemana();
}
```

No método `obterNegociacoesDaSemana()` temos que ter acesso ao retorno, porque será na `controller` que levantaremos os dados com os quais atualizaremos o model e a View ser renderizada. Para isto, o método receberá a função chamada `cb` (*callback*).

```
class NegociacaoService {

    obterNegociacoesDaSemana(cb) {
```

```
        let xhr = new XMLHttpRequest();
        xhr.open('GET', 'negociacoes/semana');
        xhr.onreadystatechange = () => {
            if(xhr.readyState == 4) {
                if(xhr.status == 200) {
                    JSON.parse(xhr.responseText)
                      .map(objeto => new Negociacao(new Date(objeto.data), objeto.quantidade, ob
                      .forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
                } else {
                    console.log(xhr.responseText);
                }
            }
        }

        xhr.send();
    }
}
```

Depois, em `NegociacaoController.js`, usaremos uma *arrow function* com dois valores:

```
importaNegociacoes() {

  let service = new NegociacaoService();

  service.obterNegociacoesDaSemana(() => {

  });
}
```

Quando o nosso servidor, via AJAX, buscar a negociação e estiver tudo pronto, ele chamará a função que adicionamos. Agora, vamos inserir um `if` para o caso em que ocorrer um erro.

```
importaNegociacoes() {

  let service = new NegociacaoService();

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

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

Se o erro não retornar preenchido, o `if` não será executado. Com o `forEach()`, para cada negociação, vamos incorrer em `this._listaNegociacoes.adiciona()`.

Também vamos adotar um convenção: em casos de erro, ele será descoberto sempre no primeiro parâmetro e o resultado da operação virá no segundo. Estamos aplicando um padrão que vem do mundo NodeJS, e que recebe o nome de *Error-First-Callback*.

Então, se ocorrer um erro, exibiremos a mensagem e daremos o retorno. Desta forma, as linhas abaixo do `return` não serão executadas. Mas no caso em que venha uma negociação, faremos o `forEach()`.

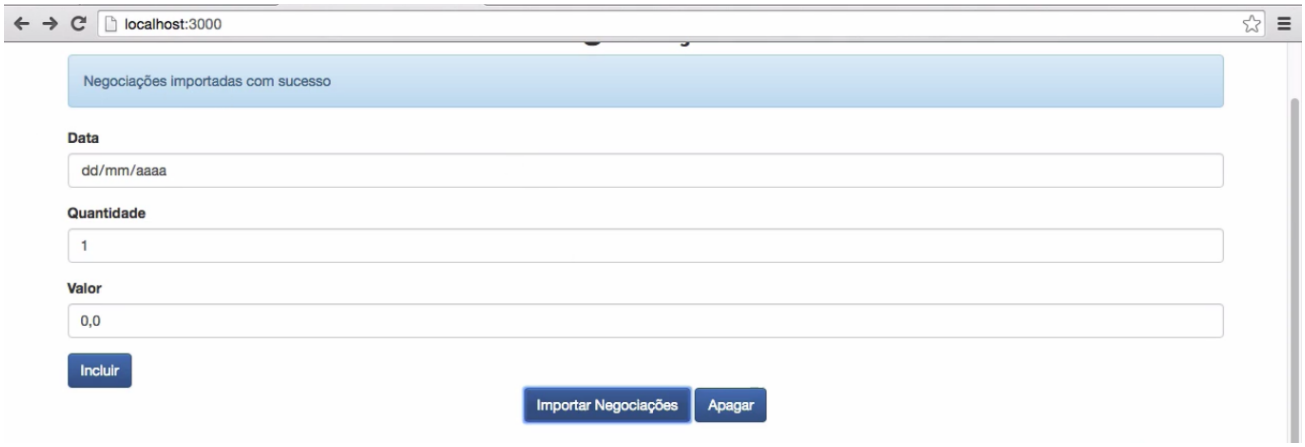
Agora, em `NegociacaoService`, vamos implementar o *callback*(`cb`):

```
class NegociacaoService {

    obterNegociacoesDaSemana(cb) {

        let xhr = new XMLHttpRequest();
        xhr.open('GET', 'negociacoes/semana');
        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,

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

        xhr.send();
    }
}
```

Se ocorrer um erro, executaremos o `cb` de alto nível, informando para o usuário que não foi possível obter as negociações.

Esta estratégia de *Error-First* significa que passaremos a função `obterNegociacoesDaSemana()`, se tiver sucesso receberá o primeiro parâmetro `null`, indicando que não teve o erro, e no segundo parâmetro, teremos o retorno. Em caso de erro, o primeiro parâmetro passarem será o erro, e o segundo, será o valor `null`. Temos a opção de deixar o segundo parâmetro em branco também.

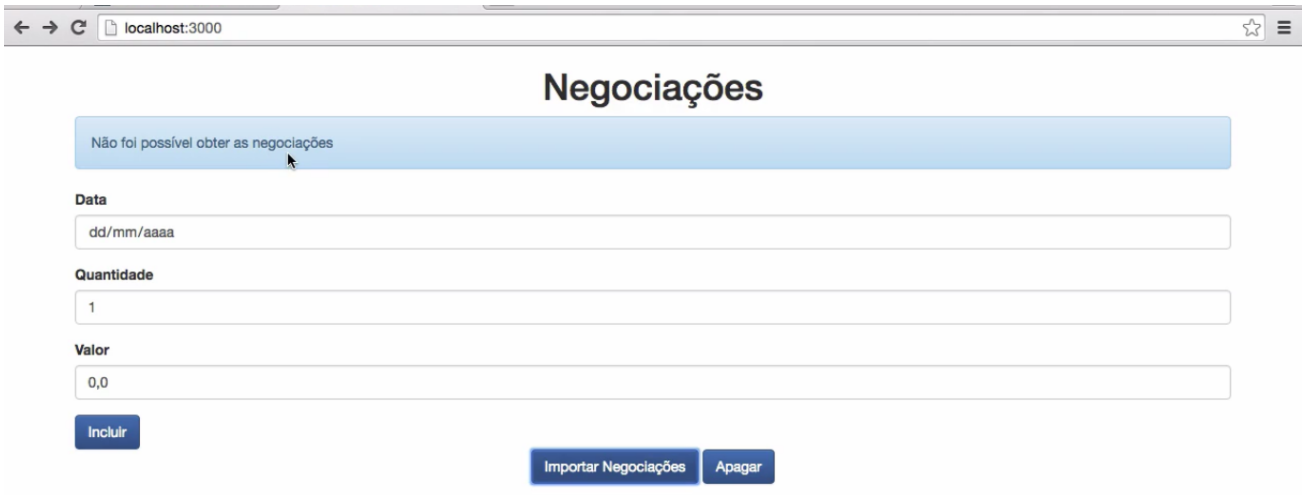
Ao recarregar a página e importar as negociações, veremos a mensagem de sucesso.



Nós isolamos a URL no serviço.

```
xhr.open('GET', 'negociacoes/semana');
```

Se tivermos diversas controllers utilizando o serviço, só precisaremos alterar a URL uma única vez. Caso a URL esteja errada, veremos a mensagem de erro:



O código de `NegociacaoController` ficou mais limpo. Faremos um pequeno ajuste substituindo o `err` por `erro`:

```
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';
  });
}
```

Fizemos uma pequena revisão de AJAX. Trabalhamos com requisições assíncronas, usando JavaScript puro. Aprendemos a isolar a lógica que geramos para realizar a requisição do servidor. Aprendemos a trabalhar com a ideia

do callback e *Error-First*.