13

Para saber mais: Igualdade de objetos

Vamos lembrar rapidamente como criamos o nosso filtro para saber se já existe uma negociação dentro da nossa lista de negociações.

Segue o esboço de código do filtro, lembrando que o filtro recebe o critério de inclusão. Se o critério devolve `true` a negociação deve fazer parte da lista:

```
//classe NegociacaoService no método importa


negociacoes.filter(negociacao =>
    !this._listaNegociacoes.negociacoes.some(negociacaoExistente =>
        JSON.stringify(negociacao) == JSON.stringify(negociacaoExistente) //criterio
    )
)
```

Para os amantes do paradigma OO, usarmos a comparação com `JSON.stringify(..)` pode representar uma quebra de um princípio importante: o *Encapsulamento*.

A pergunta que devemos sempre fazer quando pensamos no mundo OO é: Quem possui essa responsabilidade? Nesse caso especifico, quem deve saber quando uma negociação é igual a uma outra negociação?

Imagine que você precisa verificar a igualdade entre negociações em uma outra classe. Devemos realmente repetir `JSON.stringify(..)` ? A resposta é **não**, pois devemos colocar esse comportamento em um único lugar, dentro da nossa classe `Negociação` ! Em outras palavras, a classe `Negociacao` sabe quando uma negociação é igual a outra, essa **regra fica encapsulada** dentro da classe.

No mundo JavaScript, não existe um nome padrão para esse tipo de método - o mais utilizado é `equals` ou `isEquals` . Vamos implementar esse método na classe `Negociacao` :

```
class Negociacao {

    //construtor e métodos omitidos

    isEquals(outraNegociacao) {
        return JSON.stringify(this) == JSON.stringify(outraNegociacao)
    }
}
```

Repare que usamos `JSON.stringify(..)` dentro do método `isEquals` . Assim poderemos aproveitar esse método em qualquer lugar e melhorar no código do nosso filtro:

```
class NegociacaoService {

    constructor() {
```

```
            this._http = new HttpService();
        }

        // código anterior omitido

        importa(listaAtual) {

            return this.obterNegociacoes()
                .then(negociacoes =>
                    negociacoes.filter(negociacao =>
                        !listaAtual.some(negociacaoExistente =>
                            negociacao.isEquals(negociacaoExistente)))
                )
                .catch(erro => {
                    console.log(erro);
                    throw new Error('Não foi possível buscar negociações para importar');
                })
        }
    }
```

Mais elegante e principalmente encapsulado!