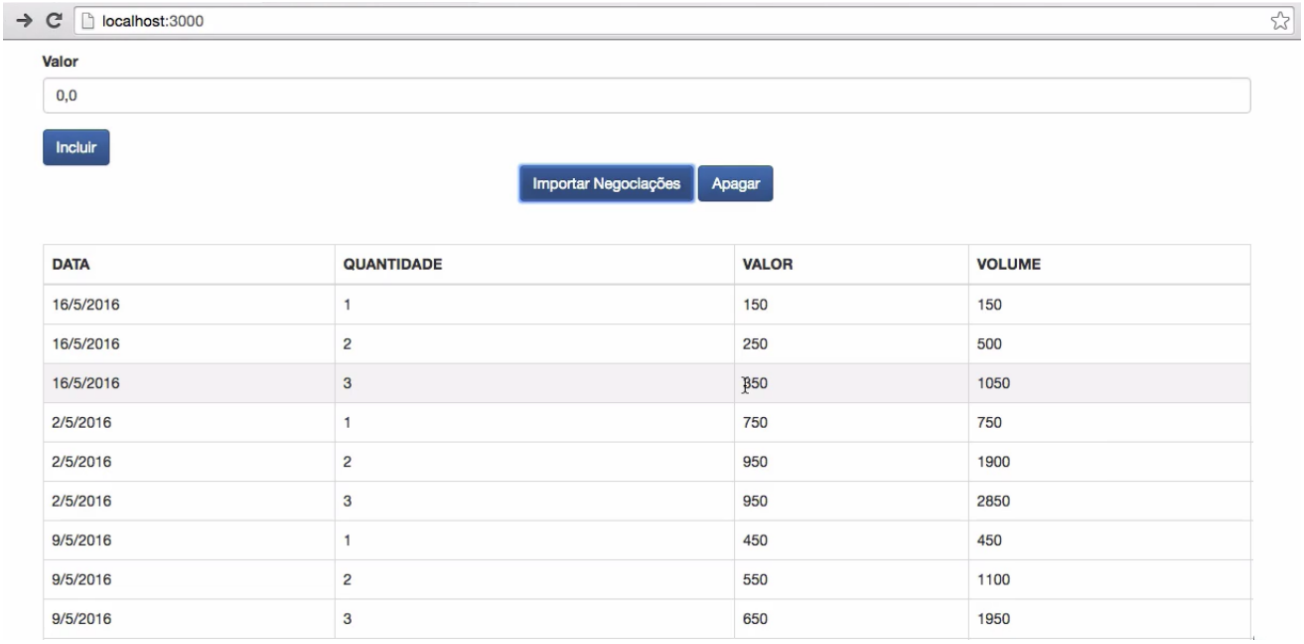
03

Pyramid of Doom novamente? Claro que não, Promise.all nela!

Transcrição

Caímos no mesmo problema enfrentado anteriormente: as negociações são exibidas fora da ordem das datas.



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

Isto ocorreu porque a Promise é assíncrona e as negociações são executadas de forma independente.

```
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);

}
```

Outro problema é que estamos tratando a mensagem de erro em cada uma das promises. Como resolveremos isto? A promise possui um recurso com o qual temos uma sequência de operações assíncronas, que será executada em uma determinada ordem.

Uma maneira de executarmos todas as promises em ordem e obtermos todos os resultado de uma vez só é usar a função `Promise.all`, que receberá um array com as promises. Vamos fazer a segunda refatoração do nosso código:

```
importaNegociacoes() {

    let service = new NegociacaoService();

    Promise.all(
        service.obterNegociacoesDaSemana(),
        service.obterNegociacoesDaSemanaAnterior(),
        service.obterNegociacoesDaSemanaRetrasada()
    ).then(negociacoes => {
        console.log(negociacoes)
    })
    .catch(erro => this._mensagem.texto = erro);
}
```

Pedimos para que o `Promise.all()` resolvesse todas as promises na ordem indicada. Mas iremos obter os dados da `Promise` com o `then()`. Caso ocorra um `erro`, trataremos com o `catch()`. E se der uma mensagem de erro específica de `obterNegociacoesDaSemana()`, o `catch()` será chamado - sem precisar ser chamado diversas vezes.

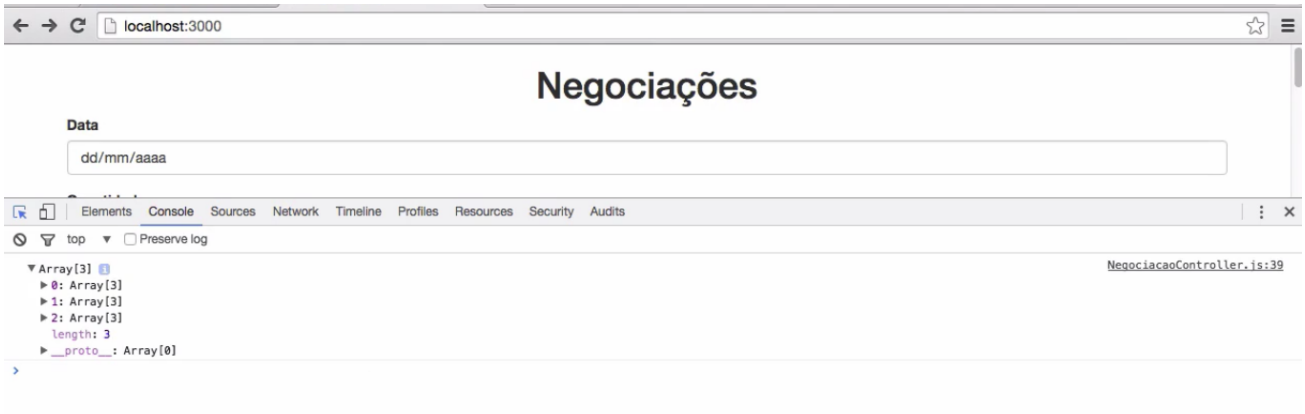
No entanto, se executarmos nosso código no navegador, teremos uma mensagem de erro, porque o `Promise.all` receberá a lista de promises dentro de um array, ou seja, elas deverão estar entre colchetes (`[]`). Com a pequena alteração, o trecho do código ficará assim:

```
importaNegociacoes() {

    let service = new NegociacaoService();

    Promise.all([
        service.obterNegociacoesDaSemana(),
        service.obterNegociacoesDaSemanaAnterior(),
        service.obterNegociacoesDaSemanaRetrasada()]
    ).then(negociacoes => {
        console.log(negociacoes)
    })
    .catch(erro => this._mensagem.texto = erro);
}
```

Agora, faremos um teste para verificar se a página está funcionando. Após clicarmos em "Importar Negociações", o array será exibido no Console.



A grande vantagem da função `Promise.all()` é que todas as promises do array serão exibidos na sequência e o resultado estará em `negociacoes`, e em caso de erro, ele será capturado uma única vez. No entanto, a `negociacao` retornada não é equivalente à lista de negociações, mas sim, cada posição do array será uma lista de negociações. Para resolver a questão, usaremos o `forEach()`, mas teremos que pensar bem no que faremos.

```
importaNegociacoes() {

    let service = new NegociacaoService();

    Promise.all([
        service.obterNegociacoesDaSemana(),
        service.obterNegociacoesDaSemanaAnterior(),
        service.obterNegociacoesDaSemanaRetrasada()]
    ).then(negociacoes => {
        negociacoes.forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações importadas com sucesso';
    })
    .catch(erro => this._mensagem.texto = erro);
```

Se usarmos esta opção, o `negociacao` dentro do `forEach()` será uma lista. Seguiremos por outro caminho... Antes de chegarmos até o `forEach()`, executaremos uma **transformação** do array que possui outros três dentro de si. Com o `reduce()`, criaremos um array que contem apenas um elemento, contendo todos as negociações. Nós faremos *flatten* - achatar - o array.

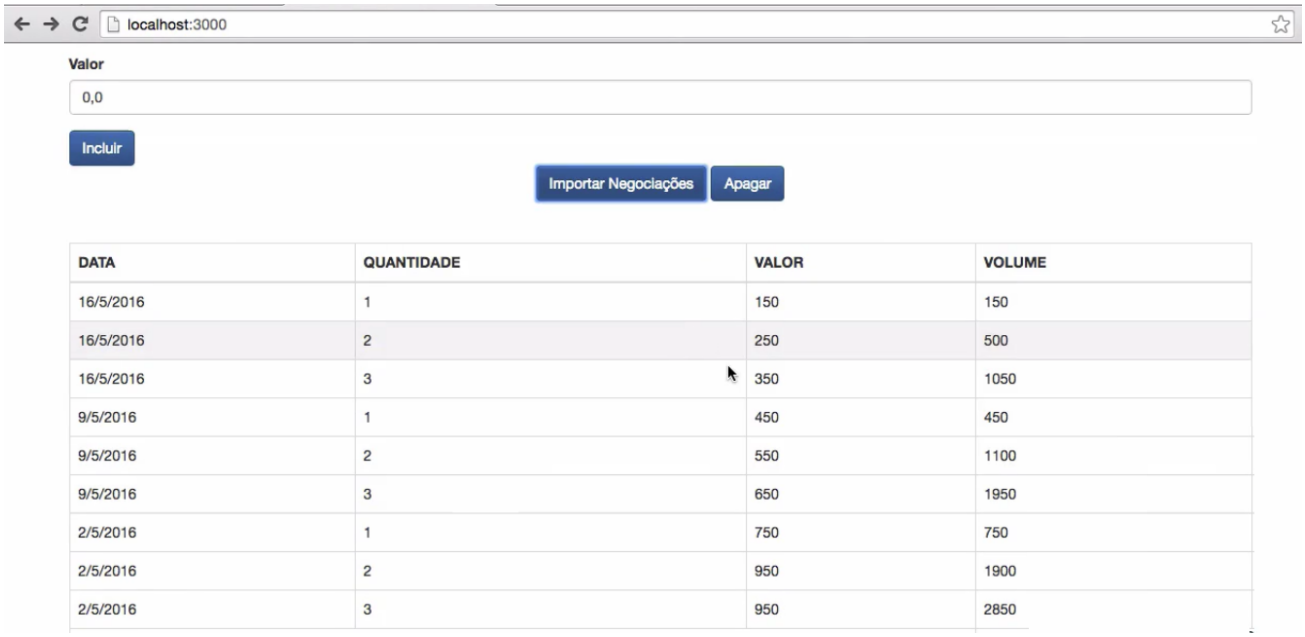
```
importaNegociacoes() {

    let service = new NegociacaoService();

    Promise.all([
        service.obterNegociacoesDaSemana(),
        service.obterNegociacoesDaSemanaAnterior(),
        service.obterNegociacoesDaSemanaRetrasada()]
    ).then(negociacoes => {
        negociacoes
          .reduce((arrayAchatado, array) => arrayAchatado.concat(array), [])
          .forEach(negociacao => this._listaNegociacoes.adiciona(negociacao));
        this._mensagem.texto = 'Negociações importadas com sucesso';
    })
    .catch(erro => this._mensagem.texto = erro);

}
```

Observe que também usamos a função `concat()`, que concatenará o array da primeira posição de `negociacoes`. No fim, o `reduce` devolverá uma única lista cheia de negociações e o `forEach()` será executado sem problemas.



Primeiro ele resolveu as negociações da semana atual, depois da semana anterior e por último, da semana retrasada. E como só tratamos o erro em um único lugar, se provocarmos o erro modificando a URL da semana anterior, em `NegociacaoService.js`, por exemplo:

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

A mensagem de erro será exibida para o usuário.

O mesmo ocorreria se o problema estivesse relacionado com as demais semanas.

Vimos como solucionar questões assíncronas com o padrão de projeto `Promise`. Mas ainda podemos extrair do `NegociacaoService`, a parte que lida com `XMLHttpRequest`.