04

Isolando a complexidade em HttpService

Transcrição

Se quisermos realizar um requisição do tipo `GET`, precisaremos repetir isso no código e preparar o `onreadystatechange`...

```
class NegociacaoService

  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.quant

                      }

//...
```

Para facilitar o processo, criaremos um serviço que chamaremos de `HttpService.js`, que ficará na pasta `Service`. A nova classe não terá nenhum atributo ou método estático. Ele terá o método `get()`, com a URL que queremos conectar, e que retornará uma `Promise`.

```
class HttpService {

    get(url) {

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

            let xhr = new XMLHttpRequest();
            xhr.open('GET', url);
            xhr.onreadystatechange = () => {
                if(xhr.readyState == 4) {
                    if(xhr.status == 200) {
                        resolve(JSON.parse(xhr.responseText));
                    } else {
                        console.log(xhr.responseText);
                        reject(xhr.responseText);
                    }
                }
```

```
            }

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

Observe que aproveitamos o padrão `Promise` e, em caso de erro, repassaremos o que veio do servidor. Também vamos passar o que veio do servidor em caso de sucesso, `JSON.parse(xhr.responseText)`. Se quisermos usar o `HttpService`, faremos o seguinte:

```
service = new HttpService();

service.get('negociacoes/semana').then(negociacoes => ????);
```

Com `then()`, teremos acesso à lista de negociações e poderemos fazer as negociações que desejamos. Deixaremos desta forma genérica, e você poderá fazer melhorias... Nossa intenção era mostrar o conceito para isolarmos este código.

Agora, `NegociacaoService` é dependente de `HttpService`, vamos declarar isto no construtor da classe.

```
class NegociacaoService {

      constructor() {

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

Em seguida, faremos grandes alterações no `return` do `obterNegociacoesDaSemana()`:

```
class NegociacaoService {

    constructor() {
        this.http = new HttpService();
    }

    obterNegociacoesDaSemana() {

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

            this.http
                .get('negociacoes/semana')
                .then(negociacoes => {
                    resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obj
                })
                .catch(erro => {
                    console.log(erro);
                    reject('Não foi possível obter as negociações da semana');
                })
        });
  }
```

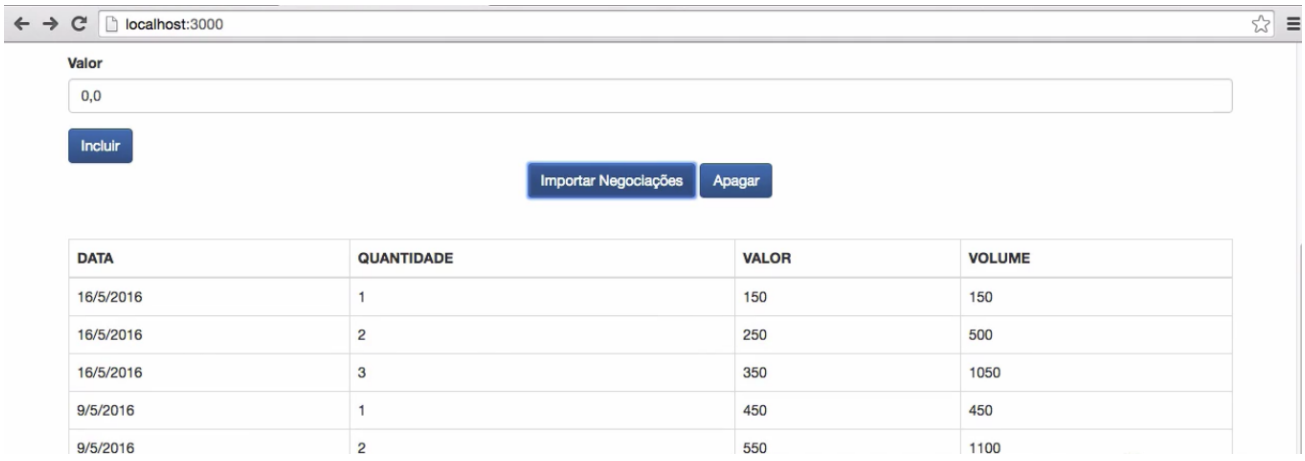
O `reject` será responsável pela mensagem que será exibida para o usuário.

Vamos rever o que fizemos até aqui... Pedimos para o serviço `_http` buscar `negociacoes/semana` e, no retorno, já teremos objetos "parseados". Mas no caso do `NegociacaoService` que quando usamos este endereço, trata-se de uma lista de negociações com um objeto - que contem dado, quantidade e valor. Nós converteremos esta lista para outra em que teremos instâncias de negociações e passaremos para o `resolve` .

Agora, só falta importarmos o `HttpService.js` no `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 src="js/app/services/HttpService.js"></script>
<script>
    let negociacaoController = new NegociacaoController();
</script>
```

Carregaremos novamente a página e clicaremos em "Importar Negociações".



Continuaremos o trabalho de conversão para usarmos o `HttpService` . As alterações serão feitas em `obterNegociacoesDaSemanaAnterior` e `obterNegociacoesDaSemanaRetrasada` :

```
obterNegociacoesDaSemanaAnterior() {

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

        this.http
            .get('negociacoes/anterior')
            .then(negociacoes => {
                console.log(negociacoes);
```

```
                resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), objeto.

            })
            .catch(erro => {
                console.log(erro);
                reject('Não foi possível obter as negociações da semana anterior');
    })
});

obterNegociacoesDaSemanaRetrasada() {

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

        this.http
            .get('negociacoes/retrasada')
            .then(negociacoes => {
                console.log(negociacoes);
                resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), objeto.

            })
            .catch(erro => {
                console.log(erro);
                reject('Não foi possível obter as negociações da semana retrasada');
    })
});
```

Depois de salvar as mudanças, podemos executar o código e a página funcionará corretamente.

Nós conseguimos desmembrar a parte de `HttpService`. Caso você queira criar um `POST` ou `PUT` ou `DELETE`, basta acrescentar estes métodos no `HttpService`, usando o padrão `Promise`.