03

Exibindo todas as negociações

Transcrição

Vamos agora implementar o método `listaTodos()` no `NegociacaoDAO`.

```
listaTodos() {

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

    });
}
```

Em seguida, aproveitaremos o código que criamos no `listaTodos` do `aprendendo_indexeddb.html`:

```
listaTodos() {

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

                let cursor = this._connection
                    .transaction([this._store],"readwrite")
                    .objectStore(this._store)
                    .openCursor();

                    // o restante do código vem aqui!
            });
}
```

O `cursor` é o responsável por passear pelos dados da Object Store. Ele tem um ponteiro para o primeiro, segundo e os demais elementos ordenados. Sabemos que depois, ele chamará os elementos para cada item do banco. Se temos dez negociações, o `onsuccess` será chamado o mesmo número de vezes. E quando este evento é chamado em `e.target.result`, temos o ponteiro `atual` do cursor. No `if`, testaremos se o ponteiro existe, caso o resultado seja positivo, vamos pedir o dado para o ponteiro.

```
let negociacoes = [];
cursor.onsuccess = e => {
    let atual = e.target.result;

    if(atual) {

        let dado = atual.value;

        negociacoes.push(new Negociacao(dado._data, dado._quantidade, dado._valor));

        atual.continue();
    } else {
        console.log(negociacoes);
    }
}
```

Os dados do ponteiro são da negociação, mas teremos que construir uma nova negociação com as informações, que serão colocadas dentro de um array. Depois, continuaremos com o processo até ele varrer todas as negociações. Quando ele terminar, `atual` será nulo. Ou seja, ao chegar na linha do `console.log(negociacoes)` já teremos a lista de negociação totalmente preenchida - vimos isto na primeira aula. Será esta linha que passaremos para o `resolve()`. Com a alteração, o trecho do código ficará da seguinte forma:

```
cursor.onsuccess = e => {
    let atual = e.target.result;

    if(atual) {

        let dado = atual.value;

        negociacoes.push(new Negociacao(dado._data, dado._quantidade, dado._valor));

        atual.continue();
    } else {

        resolve(negociacoes);
    }
}
```

Quando o cursor entra no `else`, ele passará a lista de negociações para o `resolve()`. E nos casos de erro, adicionaremos uma mensagem de alto nível no `reject()`:

```
cursor.onerror = e => {

    console.log(e.target.error);
    reject('Não foi possível listar as negociações');
}
```

Geramos o método `listaTodos`. O nosso objetivo é que ao recarregarmos as páginas, seja possível recuperar todas as negociações gravadas no banco, por isso, colocaremos o código que busca as negociações no construtor da classe `NegociacaoController`. Se observarmos o arquivo `index.html`, veremos que a classe é instanciada assim que a aplicação é iniciada.

```
<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 src="js/app/services/ConnectionFactory.js"></script>
<script src="js/app/dao/NegociacaoDao.js"></script>
<script>
```

```
    var negociacaoController = new NegociacaoController();
</script>
```

Então, colocaremos o `ConnectionFactory` também dentro do `constructor`.

```
ConnectionFactory
    .getConnection()
    .then(connection => {

        new NegociacaoDao(connection)
            .listaTodos()
            .then(negociacoes => {

                negociacoes.forEach(negociacao)

            });
    });
```

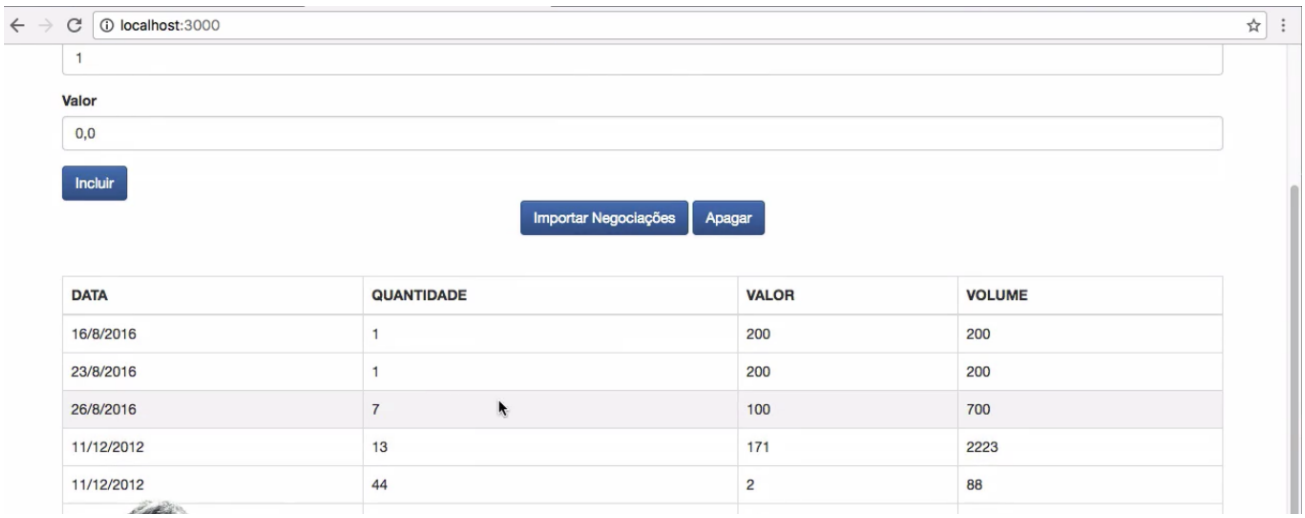
Como não queremos trabalhar diretamente com a conexão, passamos o `NegociacaoDAO`. Agora, para cada negociação, adicionaremos o `this._listaNegociacoes()`.

```
ConnectionFactory
    .getConnection()
    .then(connection => {

        new NegociacaoDao(connection)
            .listaTodos()
            .then(negociacoes => {

                negociacoes.forEach(negociacao => {
                    this._listaNegociacoes.adiciona(negociacao);
                });
            });
    });
```

Até aqui, temos que buscar todas as negociações do banco e adicionar na lista de negociações. Veremos que todas as negociações gravadas serão exibidas na tabela quando recarregarmos a página no navegador.



Se atualizarmos a página novamente, confirmaremos que os dados persistiram. Porém, podemos simplificar o código escrito até aqui. Vamos fazer a seguinte alteração no `then()` do `ConnectionFactory` :

```
ConnectionFactory
    .getConnection()
    .then(connection => new NegociacaoDao(connection))
    .then(dao => {
    })
```

O retorno estará disponível para a próxima chamada do `then()` . E se fizermos o `dao.listaTodos` ? Qual será o retorno?

```
ConnectionFactory
    .getConnection()
    .then(connection => new NegociacaoDao(connection))
    .then(dao => dao.listaTodos());
```

O retorno será uma lista de negociações. Aproveitaremos o `negociacoes` e depois, faremos um `forEach()` .

```
    ConnectionFactory
        .getConnection()
        .then(connection => new NegociacaoDao(connection))
        .then(dao => dao.listaTodos())
        .then(negociacoes =>
            negociacoes.forEach(negociacao =>
                this._listaNegociacoes.adiciona(negociacao)))
```

Conseguimos resumir ainda mais o código, usando Promises.

No entanto, se testarmos o botão "Apagar" da nossa página, veremos que ele só apagará os dados do modelo. Ao recarregarmos a página, todos os dados serão exibidos na nossa tabela. A seguir, implementaremos no DAO, a capacidade de apagar os dados da Object Store para completarmos as funcionalidades da aplicação.