02

Combinando padrões de projeto

Transcrição

Vamos integrar o `NegociacaoDao` com o `NegociacaoController`. Primeiramente, trabalharemos com o método `adiciona()`. Obteremos a conexão, adicionando o `ConnectionFactory`:

```
adiciona(event) {

    event.preventDefault();

    ConnectionFactory
        .getConnection()
        .then(conexao => {
          let negociacao = this._criaNegociacao();
          new NegociacaoDao(conexao)
              .adiciona(negociacao)

        });
```

Chamamos o `_criaNegociacao()`, responsável por criar os dados do formulário. Além de adicionarmos no `IndexedDB`, precisaremos também adicionar a negociação na `_listaNegociacoes` para refletir o que é visto pelo usuário. Porém, tal ação só será possível se gravarmos a negociação no banco. Para termos a confirmação sobre a gravação do dado, usaremos o `then()` do `adiciona()`.

```
    ConnectionFactory
        .getConnection()
        .then(conexao => {
          let negociacao = this._criaNegociacao();
          new NegociacaoDao(conexao)
              .adiciona(negociacao)
              .then(() => {
              })
        });
```

No DAO, nós chamamos o `resolve()` que será o responsável por executar o `then()`.

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

        let request = this
            ._connection
            .transaction([this._store],"readwrite")
            .objectStore(this._store)
            .add(negociacao);
        request.onsuccess = (e) => {
            resolve();
        };
//...
```

Se tivéssemos passado um parâmetro para `resolve()`, seu valor seria repassado para o `then()`. Mas nós não queremos passar valores como parâmetros para nenhum dos dois, queremos apenas resolver a nossa *Promise*. Se tudo der certo, o código no `then()` será executado. Vamos reaproveitar partes do código do antigo `try` do `NegociacaoController`.

```
.then(() => {
    this._listaNegociacoes.adiciona(negociacao());
    this._mensagem.texto = 'Negociação adicionada com sucesso';
    this._limpaFormulario();
  })
```

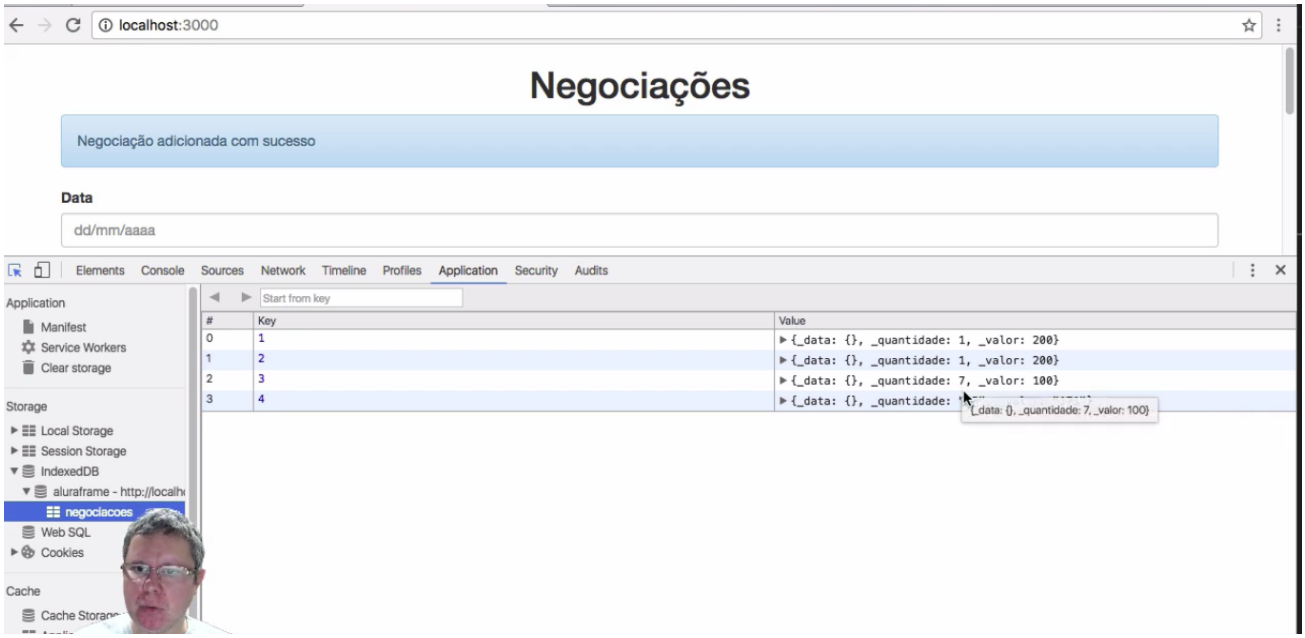
Após alterarmos o texto, limparemos o formulário. Também aproveitaremos o `event.preventDefault()`.

```
adiciona(event) {

        event.preventDefault();

        ConnectionFactory
            .getConnection()
            .then(conexao => {

                let negociacao = this._criaNegociacao();

                new NegociacaoDao(conexao)
                    .adiciona(negociacao)
                    .then(() => {
                        this._listaNegociacoes.adiciona(negociacao);
                        this._mensagem.texto = 'Negociação adicionada com sucesso';
                        this._limpaFormulario();
                    });
        })
        .catch(erro => this._mensagem.texto = erro);
```

Nós não teremos um `try/catch`. Para o tratamento de erro usaremos o `catch` e o erro capturado será exibido para o usuário. Como só trabalhamos com uma instrução, não precisamos adicionar o bloco. Mas se conseguirmos obter a conexão, e dela instanciaremos um DAO, por meio dele vamos instanciar a `negociacao`. Logo, poderemos adicionar a negociação no modelo. Mas o fato de gravarmos no banco, não significa que os dados serão exibidos na tela, teremos que gravar também na `_listaNegociacoes`. Com o *data binding* criado por nós, depois de adicionarmos a negociação na lista, ela será automaticamente refletida para o usuário.

Vamos recarregar a página e preencher os dados nos campos do formulário.

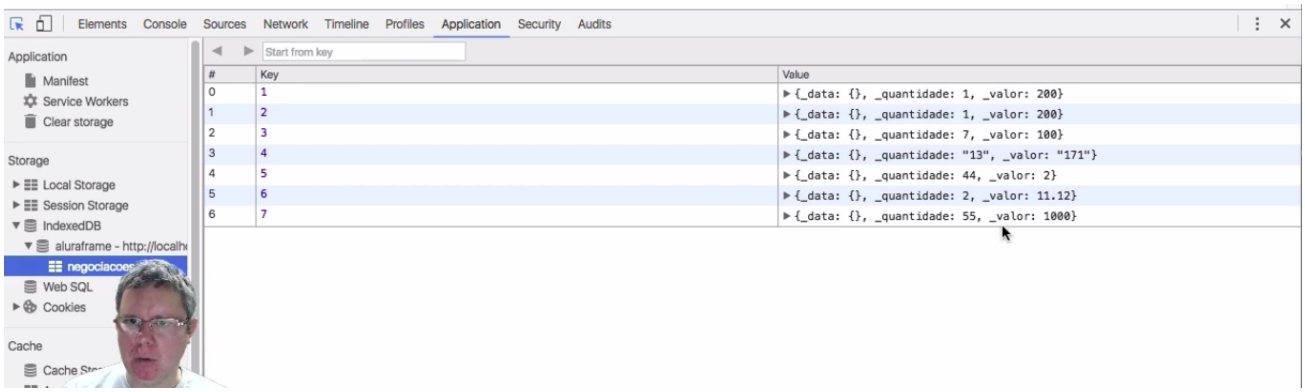
Podemos visualizar a mensagem e os dados sendo exibidos na tabela. Isto significa que conseguimos gravar as informações no banco. Se acessarmos a aba "Application", veremos que os dados da nova negociação também foram adicionados.



Observe que ele salvou os dados como *String*, o ideal seria salvá-los como números. Para resolvermos isto, no `_criaNegociacao` , vamos garantir que o valor de "quantidade" será inteiro, adicionando o `parseInt()` . E com o `parseFloat` no `value` , podemos trabalhar um número decimal:

```
_criaNegociacao() {

    return new Negociacao(
        DateHelper.textoParaData(this._inputData.value),
        parseInt(this._inputQuantidade.value),
        parseFloat(this._inputValor.value));
}
```

Se testarmos no navegador, veremos que os dados serão salvos no banco já usando as novas configurações.



Se cadastrarmos valores decimais no campo de "Valor", veremos que ele será gravado inclusive com os valores decimais.

Agora, vamos resolver o problema das inclusões. Queremos que ao recarregarmos a página no navegador, as negociações salvas no banco sejam reexibidas. Ou seja, traremos as negociações do banco e, depois, estas serão exibidas para o usuário.