02

A classe ConnectionFactory

Transcrição

Vamos criar a classe `ConnectionFactory` dentro da pasta `services`. Já importaremos a classe no arquivo `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 src="js/app/services/ConnectionFactory.js"></script>
<script>
    let negociacaoController = new NegociacaoController();
</script>
```

Geraremos o método estático `getConnection()`, e para garantir que o desenvolvedor não crie uma instância da classe, lançaremos uma exceção em seu construtor:

```
class ConnectionFactory {

    constructor() {

        throw new Error('Não é possível criar instâncias de ConnectionFactory');
    }

    static getConnection() {

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

        });
    }
}
```

O `getConnection` retornará uma `Promise()` - o que já foi visto na segunda parte do curso. Até aqui, já atendemos duas das regras para fazermos o design da classe `ConnectionFactory`. Criaremos a conexão com base no que vimos: adicionaremos a variável `openRequest` e faremos uma requisição de abertura.

```
static getConnection() {

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

```
            let openRequest = window.indexedDB.open('aluraframe',4);

        });
    }
```

Observe que utilizamos o valor `4`, porque foi a última versão do banco utilizada. Este é o código que fará com que o `onupgradeneeded` seja executado, caso o valor seja maior do que o banco criado. Trabalharemos com a tríade de eventos que são disparados toda vez que uma requisição de abertura de conexão é feita: `onupgradeneeded`, `onsuccess` e `onerror`.

```
    static getConnection() {

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

            let openRequest = window.indexedDB.open('aluraframe',4);

            openRequest.onupgradeneeded = e => {
            };

            openRequest.onsuccess = e => {
            };

            openRequest.onerror = e => {
            };
        });
    }
```

As *stores* são criadas no `onupgradeneeded`, criaremos uma nova Object Store que receberá o nome de `negociacoes`. Mas não poderemos declarar a variável como propriedade do construtor da classe, porque ela só seria acessível por meio de uma instância de classe, por isso, iremos declará-la como uma variável global. Esta será uma saída momentânea. Vamos criar também as variáveis `version` e `dbName` para guardarem a versão do banco e o seu nome. Então, acima do `ConnectionFactory`, adicionaremos as três variáveis:

```
var stores = ['negociacoes'];
var version = 4;
var dbName = 'aluraframe';

//...
```

Nós usaremos essas variáveis no `onupgradeneeded` para criarmos as histórias.