03

Criando stores

Transcrição

Criaremos as nossas *stores*, por enquanto, temos um array com apenas uma. Atualmente o nosso código está assim:

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

class ConnectionFactory {


    constructor() {

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

    static getConnection() {

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

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

            openRequest.onupgradeneeded = e => {

                stores.forEach(store => {

                });

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

Começaremos adicionando `e.target.result` no `stores.forEach()`:

```
static getConnection() {

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

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

            openRequest.onupgradeneeded = e => {

                stores.forEach(store => {

                    if(e.target.result.objectStoreNames.contains(store)) e.target.result.delete(
```

```
                    e.target.result.createObjectStore(store, { autoIncrement: true });
                });
            };
```

Com o `if` recém adicionado, apagaremos as Objects Stores que já existem. Caso tudo esteja correto, criaremos uma nova store e o `autoIncrement` será `true`. Mas se você observar o nosso código, perceberá que temos parte repetidas no código. Para solucionarmos isto, adicionaremos um método estático privado `_createStores()`, e depois, moveremos o `forEach()` para ele:

```
static _createStores(connection) {

    // criando nossos stores!

    stores.forEach(store => {

        if(connection.objectStoreNames.contains(store)) connection.deleteObjectStore(store);
        connection.createObjectStore(store, { autoIncrement: true });
    });
}
```

Nos substituímos `e.target.result` pelo método `_createStores()` porque ele recebe uma `connection`. Depois, vamos inserir o método no `onupgradeneeded`:

```
static getConnection() {

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

            let openRequest = window.indexedDB.open(dbName,version);

            openRequest.onupgradeneeded = e => {

                ConnectionFactory._createStores(e.target.result);
            };
  //...
```

Chamaremos o método auxiliar privado da classe - vimos anteriormente, que o método poderá ser chamado por qualquer um, mas quando utilizamos o prefixo `_`, significa que este só deverá ser chamado pela própria classe. Então, fechamos o `onupgradeneeded`. Vamos agora trabalhar com o `onsuccess`:

```
openRequest.onsuccess = e => {

   resolve(e.target.result);
};
```

Para os casos de erro, teremos o `onerror`:

```
openRequest.onerror = e => {

    console.log(e.target.error);
```

```
        reject(e.target.error.name);
    };
```

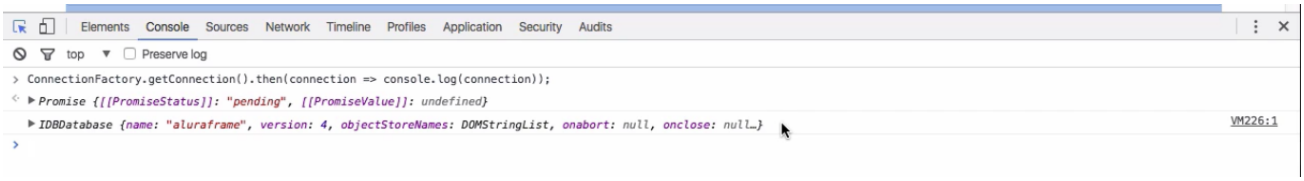
No `reject`, em vez de passarmos o objeto `error`, retornaremos um *string* com a mensagem de erro. Nosso código ficou assim:

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

class ConnectionFactory {

    constructor() {

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

    static getConnection() {

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

            let openRequest = window.indexedDB.open(dbName,version);

            openRequest.onupgradeneeded = e => {

                ConnectionFactory._createStores(e.target.result);
            };

            openRequest.onsuccess = e => {

                resolve(e.target.result);
            };

            openRequest.onerror = e => {

                console.log(e.target.error);

                reject(e.target.error.name);
            };
        });
    }
    static _createStores(connection) {

        stores.forEach(store => {

            if(connection.objectStoreNames.contains(store)) connection.deleteObjectStore(store)
            connection.createObjectStore(store, { autoIncrement: true });
        });
    }
}
```

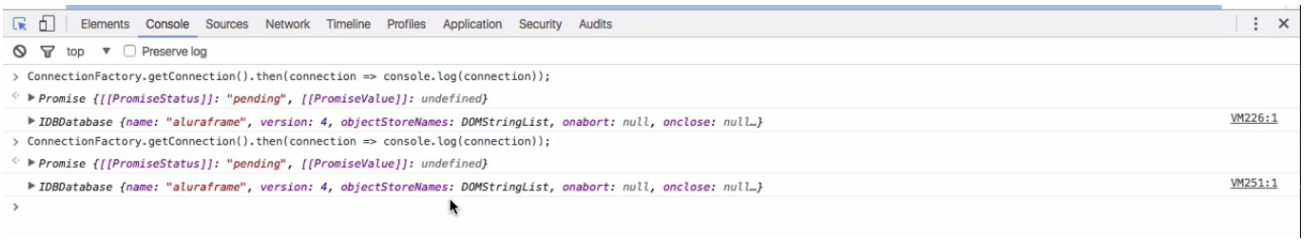
Vamos realizar um teste na página do navegador. No Console, chamaremos o método `ConnectionFactory.getConnection`:

```
ConnectionFactory.getConnection().then(connection => console.log(connection));
```

Ele irá nos retornar o `IDBDatabase` .



Está tudo funcionando corretamente. Mas se chamarmos o mesmo método novamente, veremos que uma nova conexão será criada.



Nós queremos ter apenas **uma** conexão, independentemente do número de vezes que o método `getConnection` for chamado. Veremos como solucionar isto.