03

Faça como eu fiz: Paginação no SQL

Existem algumas técnicas para paginação de APIs. No caso de `User` o próprio json-server já usa os *query params* `_page` e `_limit` e faz o gerenciamento dos registros exibidos a cada requisição.

Agora vamos implementar a paginação na query `turmas` e ver outra forma de fazer isso com SQL.

> Antes de continuarmos, utilize a mutation `incluirTurma` e crie mais alguns registros na tabela `turmas`, se ainda não tiver feito isso — a partir de 6 ou 8 registros já funciona bem para testes.

> Você pode seguir o passo-a-passo detalhado com o que está sendo implementado, mas também pode pegar o código pronto neste

commit (https://github.com/alura-cursos/1982-graphql/commit/2f89512a47e8a7d2f1d46a95a6e0f350220c6a74).

Começando pela definição dos argumentos em `./api/turma/schema/turma.graphql`:

```
type Query {
  turmas (page: Int, pageOffset: Int):
  # . . . outras queries
}
```

COPIAR CÓDIGO

O primeiro, `page`, se refere ao "número da página", e `pageOffset`, a quantos registros serão exibidos por vez.

Definido o schema, a implementação no resolver é a mesma feita para `User`, que é utilizar o segundo parâmetro do resolver para passar `args` para o método `getTurmas`. Em `./api/turma/resolvers/turmaResolvers.js`:

```
Query: {
  turmas: (_, args, { dataSources }) =
  // . . . outros resolvers
},
```

COPIAR CÓDIGO

Agora, na classe `TurmasAPI`, falta atualizar o método `getTurmas` com os novos parâmetros que estão sendo recebidos via `args`.

Em `./api/turma/resolvers/turmaResolvers.js`:

```
async getTurmas({ page = 0, pageOffset

return this.db
  .select('*')
  .from('turmas')
  .offset(page)
  .limit(pageOffset)
}
```

COPIAR CÓDIGO

A primeira coisa que temos de diferente é o valor inicial dado para a propriedade `pageOffset`. Porque no caso da query que está sendo feita ao SQLite, o retorno será uma array; temos que garantir que, caso o cliente não passe nenhum valor em `pageOffset` — ou seja, não passe quantidade de registros por página — seja exibida a array inteira de resultados, independente de quantos sejam. Por isso o valor inicial está definido como `Infinity`.

Os métodos do Knex `.offset()` e `.limit()` se referem às cláusulas SQL com os mesmos nomes; porém, **em SQL o termo `offset` tem um sentido um pouco diferente, vai especificar a quantidade de linhas da tabela que serão "puladas"**. A cláusula `limit` informa a quantidade (limite) de registros retornados por requisição.

Antes de continuarmos, faça este teste no playground:

```
query {
  turmas (page: 0, pageOffset: 2) {
    descricao
  }
}
```

COPIAR CÓDIGO

Esse teste deve retornar os dois primeiros registros de `Turma`. Aparentemente tudo certo! Então faça um segundo teste:

```
query {
  turmas (page: 1, pageOffset: 2) {
    descricao
  }
}
```

COPIAR CÓDIGO

O primeiro registro dessa query deve estar repetindo o último da query anterior. Ou seja, primeira query:

```
{
  "data": {
    "turmas": [
      {
        "descricao": "básico"
      },
      {
        "descricao": "intermediário"
      }
    ]
  }
}
```


COPIAR CÓDIGO

Segunda query:


```
{
  "data": {
    "turmas": [
      {
        "descricao": "intermediário"
      },
      {
        "descricao": "conversação"
      }
```

```
    ]
  }
}
```

COPIAR CÓDIGO

Consegue identificar o que está acontecendo? Lembre-se de que 1) estamos lidando com uma array de resultados, e 2) o parâmetro `page` está sendo passado como valor de `offset` na query SQL — o valor de registros que o SQL deve "pular" antes de trazer os resultados.

Então, no primeiro teste, passamos `page: 0` (offset 0, não vai pular registro nenhum) e no segundo teste, `page: 1` (offset 1, vai pular 1 registro).

Traduzindo para uma array de exemplo:

```
['id-1', 'id-2', id-3', 'id-4']
```

COPIAR CÓDIGO

Com os parâmetros `(page: 0, pageOffset: 2)` o *start* está no índice `0` da array e traz o seguinte resultado:

```
['id-1', 'id-2', id-3', 'id-4'] // 'id-
```

COPIAR CÓDIGO

Passando `(page: 1, pageOffset: 2)`, o *start* está no índice `1` da array e o resultado é:

```
['id-1', 'id-2', id-3', 'id-4'] // 'id-
```

COPIAR CÓDIGO

Ok, como resolver isso e fazer com que `page` realmente se comporte como "página"? Adicionando no método um cálculo para que o número em `page` considere a quantidade de registros passados em `pageOffset` e "pule" a quantidade certa de registros:

```
async getTurmas({ page = 0, pageOffset

  const registroInicial = page === 0 |
    ? 0
    : (page * pageOffset) - 1

  return this.db
    .select('*')
    .from('turmas')
    .offset(registroInicial)
    .limit(pageOffset)
}
```

COPIAR CÓDIGO

A const `registroInicial` faz as seguintes verificações:

1. Se o valor de `page` é `0` ou `1` — nesse caso, vai retornar o que seria a "primeira página" de registros;
2. Se for passado em `page` qualquer valor diferente de `0` ou `1`) — o valor que será

passado como `offset` para a query SQL será `(page * pageOffset) - 1`, ou seja: `(page: 2, pageOffset: 2)` retornará o terceiro e quarto registros.

A query agora recebe `registroInicial` como valor a cada requisição. Faça novos testes no playground, com valores diferentes!

O tema paginação é complexo e ainda há outras formas de abordar a questão, sempre dependendo do framework e da base de dados! Você pode testar outras, se quiser.