07

Para saber mais: Campos nulos

Já vimos que, quando queremos que o valor de um campo nunca seja nulo, utilizamos exclamação (`!`). Como no campo `nome` no tipo `User` :

```
type User {
    nome: String!
    .
    .
    .
}
```

Aqui deixamos explícito que o valor de `nome` sempre deve retornar algum valor; ou seja, nunca pode ser `null` . O contrário fará com que o GraphQL lance um erro.

E quando trabalhamos com listas de dados? Nesse caso, existe uma diferença na parte do código onde declaramos `!` .

Primeiro caso: uma query que pede uma lista de usuários. O ponto de exclamação está somente depois dos colchetes `[]` . Ou seja, a query `users` em si não pode retornar `null` , **mas pode conter** `null` **entre os itens retornados na lista.**

```
type Query {
    users: [User]!
}
```

Exemplos de retorno:

```
users: [{user}, null, {user}] //retorno válido
users: null //retorna erro
```

Segundo caso: o ponto de exclamação está dentro dos colchetes da lista, junto ao tipo que queremos retornar na lista. Ou seja, **a lista em si pode ser nula, mas não pode retornar itens nulos**.

```
type Query {
    users: [User!]
}
```

Exemplos de retorno:

```
users: null //retorno válido
users: [{user}, {user}] //retorno válido
users: [{user}, null, {user}] //retorna erro
```

Terceiro caso: o ponto de exclamação aparece junto ao tipo de retorno e também junto ao fechamento dos colchetes da lista. Ou seja, deve retornar obrigatoriamente uma lista que não contenha `null` entre seus itens.

Atenção: uma lista vazia é um retorno válido!

```
type Query {
    users: [User!]!
}
```

Exemplos de retorno:

```
users: [] //retorno válido
users: null //retorna erro
users: [{user}, {user}] //retorno válido
users: [{user}, null, {user}] //retorna erro
```

No GraphQL, todos os campos são definidos como "anuláveis" por padrão. Ao definir os campos de um tipo como obrigatórios (não-nulos), devemos pensar em como os dados serão utilizados e também em como estão definidos na base de dados. Por exemplo, definir como obrigatório um campo cuja origem dos dados seja uma coluna SQL sem a restrição `NOT NULL` pode nos trazer erros.

Pode parecer uma boa ideia sempre declarar todos os campos possíveis como não-nulos, mas isso pode fazer com que fique difícil evoluir o schema, especialmente quando trabalhamos com diversas fontes de dados.