08

Para saber mais: Tipos básicos do GraphQL

O GraphQL tem sua própria linguagem, chamada de SDL, ou *Schema Definition Language*, linguagem de definição de schema. Isso porque é possível implementar o GraphQL em conjunto com qualquer outra linguagem, então a SDL serve para fazer essa integração de forma *agnóstica*.

Para entender como essa linguagem funciona, sempre temos que ter em mente que o GraphQL trabalha com tipos, e saber quais tipos são esses.

SCALAR TYPES

São tipos que refletem alguns dos tipos de dados que já conhecemos. Para o GraphQL, são os tipos que se resolvem em *dados concretos* (ao contrário de objetos, por exemplo, que são conjuntos de dados). São eles:

- **Int** - inteiro de 32 bits
- **Float** - tipo ponto flutuante
- **String** - sequência de caracteres no formato UTF-8
- **Boolean** - true ou false
- **ID** - identificador único, usado normalmente para localizar dados É possível criar tipos scalar customizados, estudaremos mais adiante neste curso.

OBJECT TYPE

Quando trabalhamos com GraphQL, o ideal é pensarmos no uso dos dados, mais do que na forma em que estão armazenados. Pensando nisso, nem sempre queremos retornar um dado concreto, mas sim um conjunto de dados com propriedades específicas — ou seja, um objeto.

Um exemplo de tipo Objeto (*Object type*) em GraphQL:

```
type Livro {
    id: ID!
    titulo: String!
    autoria: String!
    paginas: Int!
    colecoes: [Colecao!]!
}
```

No exemplo acima, estamos definindo o tipo Objeto `Livro`.

As propriedades — que no GraphQL são chamadas de campos — retornam tipos scalar, como strings e inteiros, e também podem retornar arrays compostas de outros objetos, como no caso de `colecoes: [Colecao!]!`.

Note que na definição do objeto não está especificado de qual base de dados virão esses dados, apenas *quais dados o GraphQL espera receber, e de que tipos*.

Os campos marcados com exclamação `!` são campos que não podem ser nulos. Ou seja, qualquer query que envolva estes campos sempre devem ter algum valor do tipo esperado. No caso de `colecoes: [Colecao!]!` a exclamação após o

fechamento da array significa que o campo `colecoes` sempre vai receber uma array (tendo ou não elementos dentro dela); a exclamação em `Colecao!` significa que qualquer elemento dentro da array sempre vai ser um objeto `Colecao`.

QUERY TYPE

Os tipos Query definem os pontos de entrada (*entry points*) da API; indicam quais dados o cliente pode receber e de que forma — de certa forma, são como queries do tipo GET quando trabalhamos com REST, a diferença aqui é que o cliente tem mais liberdade para montar as queries para receber apenas os dados que precisa — lembrando que, para o GraphQL e também para o cliente, não importa a *origem* desses dados. os dados podem vir de diversas fontes: endpoints REST, bancos SQL e NoSQL, outro servidor GraphQL.

Um exemplo de tipo Query:

```
type Query {
    livros: [Livro!]!
    livro(id: ID!): Livro!
}
```

Aqui definimos a query `livros`, que retorna uma array composta por tipos objeto `Livro`, e a query `livros`, que recebe um número de ID por parâmetro e retorna um objeto `Livro` referente ao ID informado.

Uma vez que as queries são os pontos de entrada de uma API GraphQL, toda aplicação vai ter pelo menos uma Query em seu schema.

MUTATION TYPE

Mutations são os tipos GraphQL utilizados para adicionar, alterar e deletar dados, de forma similar às operações de POST, PUT e DELETE nos CRUDs desenvolvidos em REST.

Os tipos Query são obrigatórios em qualquer serviço GraphQL, porém Mutations são opcionais. Um exemplo de tipo Mutation para adicionar um novo livro:

```
type Mutation {
    adicionaLivro(titulo: String!, autoria: String!, paginas: Int!, colecoes: Colecao!): Livro!
}
```

Neste exemplo temos somente uma Mutation, que chamamos de `adicionaLivro` e recebe por parâmetro os dados necessários. Confira os parâmetros com o tipo `Livro` definido anteriormente!

Além dos tipos acima, o GraphQL ainda tem mais tipos básicos que trabalharemos com mais detalhes durante o curso:

- Enum,
- Input,
- Interface,
- Union.