07

Para saber mais: Unions e interfaces

O último dos tipos do GraphQL, `Union`, é útil em casos em que existem campos similares em mais de um tipo.

Usando como base o exemplo da documentação:

```
union Resultado = Livro | Autoria

type Livro {
    titulo: String
}

type Autoria {
    nome: String
}

type Query {
    busca: [Resultado]
}
```

No exemplo acima foi definido um campo `busca` no tipo `Query`, que pode retornar tanto `Livro` quanto `Autoria`. Como passamos isso para o resolver?

```
const resolvers = {
    Resultado: {
        __resolveType(obj, context, info){
            if(obj.nome){
                return 'Autoria'
            }

            if(obj.titulo){
                return 'Livro'
            }

            return null
        },
    },
    Query: {
        busca: () => { // código aqui }
    },
}
```

No exemplo de resolver acima, foi adicionada uma nova propriedade no objeto `resolvers`: a propriedade `Resultado`, que é o nome do tipo `Union` que criamos no schema.

Nessa propriedade, chamamos o campo `__resolveType`, um campo do GraphQL que vai ser usado para resolver se o objeto retornado será `Livro` ou `Autoria`.

Dessa forma, é possível fazer queries que retornam objetos diferentes dependendo dos subcampos pedidos:

```
{
    busca(contains: "") {
        ... on Livro {
            titulo
        }
        ... on Autoria {
            nome
        }
    }
}
```

Nós vimos anteriormente um uso do tipo `Interface` para customizar os dados retornados ao cliente em uma Mutation. Um outro uso possível para `Interface` é lidar com casos onde existe mais de um tipo com campos similares, assim como `Union`.

Neste curso introdutório de GraphQL, não vamos abordar este uso específico, mas vai ficar o desafio: você pode ver os exemplos na documentação (https://www.apollographql.com/docs/apollo-server/schema/unions-interfaces/) e implementar esta funcionalidade.