Graphql é uma ferramenta para construirmos de forma ágil APIs que são rápidas e versáteis.

Vamos começar então a entender o que é o Graphql e o que ele faz, o que ele não faz e em que casos ele é bem utilizado. O Graphql é uma especificação para criar e usar APIs que têm sua própria linguagem de query.

O que isso significa então? É que às vezes o Graphql é entendido, é percebido como sendo uma tecnologia voltada para bancos de dados, ele não é. Ele é uma especificação para APIs, para escrever, criar e utilizar APIs e não está ligado a nenhum tipo de banco, inclusive ele pode ser usado com qualquer base de dados, ou mesmo nenhuma base de dados.

O Graphql fornece um ambiente para executarmos essas queries usando os dados que fornecermos para ele, não importa de onde esses dados venham. Eles podem vir de um banco SQL, de um banco não SQL, de uma API de terceiros via endpoint rest, ou até mesmo da memória do seu computador. E também de todos esses juntos, para o SQL não faz diferença, isso não é o ponto principal dele.

O Graphql também normalmente utilizamos http para fazer a comunicação, mas na verdade ele é agnóstico com relação a protocolos de comunicação. Os schemas do Graphql são baseados em como os dados são usados, e não como estão armazenados. Isso é um conceito chave para entender o Graphql, não importa se os dados vêm de objetos, de APIs, o que acabamos de falar, o que importa é que o cliente consiga utilizar esses dados da melhor forma possível.

Quando falo cliente normalmente estou falando do front ou de quem vai consumir nossa API Graphql. Essa é uma questão chave. Os schemas são baseados em como os dados são usados, e não importa de onde eles vêm.

Por exemplo, se usássemos um banco de tabelas para criar um usuário, por exemplo, um usuário de uma escola de inglês, que é o que vamos trabalhar durante o curso, esses usuários têm nome, e-mail, se eles estão ativos no sistema ou não, e eles podem ser professores, podem ser alunos, podem ser coordenação.

Se pensarmos, por exemplo, numa tabela SQL, conseguimos montar um usuário completo a partir de duas tabelas, uma com os dados principais, nome, e-mail, etc, e outra para definir os possíveis roles, os possíveis papeis desse usuário no sistema.

Então você teria uma tabela de roles que tem professor, aluno, coordenação, e por aí vai. Com o Graphql pensamos em como esses dados seriam usados pelo lado cliente e o cliente pode montar as queries a partir dessa premissa.

Vamos supor que numa feature qualquer desse sistema de escola de cliente, o lado cliente precisa receber somente o nome do usuários da tabela users e da tabela roles ele só precisa receber uma string com o tipo de role, professor, aluno, etc. Ele não precisa receber mais nada.

Ele quer fazer uma query que pegue de users somente o nome, e do role desse user, do papel desse usuário, somente a string de tipo, ele não quer receber mais nada, somente isso.

A partir dessa query que o cliente consegue fazer em Graphql ele recebe um JSON somente com o que ele quer no formato agregado de uma forma que para o cliente faz mais sentido, são dados mais concisos e somente com o que o cliente pediu.

O Graphql com isso procura resolver um problema em rest, uma questão do rest que costumamos chamar de overfitting, que seria super requisição, e também o underfitting, que é sub requisição. É quando o endpoint ou traz muitos dados que não precisamos numa requisição ou o contrário, precisamos de mais de uma requisição para ter os dados que precisamos.

O Graphql vem aí para resolver essa questão e fazer com que nosso cliente peça, em uma requisição só ele receba somente o que ele quer e mais nada. Podemos dizer que o Graphql é uma tecnologia voltada para front? Podemos, porque a ideia é melhorar, otimizar essa relação dos clientes com os dados que ele recebe do back, que é uma tecnologia focada no front, mas claro que vamos desenvolver nosso servidor Graphql no back.

Para o cliente, para a parte de front, ele vai ter menos requisições, vai visualizar os dados de uma forma melhor, os dados agregados de uma forma melhor para ele, mas dados mais enxutos, inclusive, de forma que faça mais sentido para ele, mas também tem bastante benefícios para a parte do back, para o lado do servidor, uma vez que livra o backend de fazer muitas implementações de muitos endpoints.

O desenvolvimento fica mais ágil. Se você tem que desenvolver um novo produto ou uma nova feature para o seu produto, evita que o backend caia naquela situação de ter que desenvolver endpoints sem fim para cada coisa nova que pode ou não entrar no sistema.

Fica mais ágil porque o front não fica dependendo tanto do back para criar um endpoint para cada coisa que ele precisa fazer, e o backend fica mais agilizado também sem ter que ficar fazendo todas essas implementações para uma coisa que no final às vezes pode até entrar no sistema ou não.

O GraphQL faz uma separação clara entre estrutura e comportamento.

A estrutura do GraphQL está no schema, no qual especifica-se o que o servidor GraphQL está estruturado para fazer, com seus tipos e objetos.

Essa estrutura precisa ser implementada de alguma forma para que possa funcionar. No GraphQL, isso se dá através do que chamamos de funções resolver, ou só resolvers. É nos resolvers que implementamos o comportamento. Cada campo em um schema GraphQL é implementado através de um resolver.

É aqui que entram ferramentas como Apollo, que vamos utilizar no curso. Elas servem para nos ajudar a implementar a especificação GraphQL em nossa aplicação.

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.