05

Para saber mais: Práticas e patterns

O GraphQL tem algumas práticas comuns e patterns próprios, diferentes do REST. Vamos ver uma lista com algumas delas.

Lembrando que, como todas as convenções e práticas, não estão escritas em pedra e você pode fazer adaptações e tomar decisões diferentes, conforme a necessidade de seu projeto no momento.

Bons cenários para o GraphQL

- Quando o desenvolvimento precisa ser flexível o bastante para produtos que mudam muito rápido, estão em fase de desenvolvimento e com muitas features diferentes para serem testadas. Se a sua API só tem um endpoint (ou poucos) ou o produto não está nessa fase, o REST continua sendo uma boa opção.
- GraphQL é sobre otimizar requisições e queries, diminuindo a quantidade de requisições e evitando os problemas do *over-fetching* ou *under-fetching* que são comuns em REST, quando uma só requisição ou traz muitos dados que não são necessários, ou não traz os dados suficientes;
- GraphQL é uma especificação e você pode utilizar as bibliotecas ou plataformas que quiser para ajudar na implementação; porém o GraphQL também funcionaria sem elas. O Apollo é uma dessas plataformas para desenvolver em GraphQL com NodeJS, mas existem várias outras voltadas para outras linguagens, como o Graphene para Python.
- GraphQL torna o desenvolvimento mais ágil evitando ajustes na API por parte do back-end para cada nova funcionalidade que vai ser implementada; por exemplo, diminui a necessidade da criação de endpoints específicos para uma determinada feature.
- O schema torna o monitoramento de recursos mais fácil e a partir dele a documentação é gerada automaticamente, o que torna o trabalho em times mais fácil.

Vale a pena notar outros pontos da criação de uma API GraphQL:

- O schema reduz bastante a complexidade de adicionar novos tipos e campos da API (mais sobre isso no tópico "versionamento" mais abaixo). Porém, não se recomenda mudanças que possam quebrar as requisições, como:
  - renomear um campo;
  - modificar os argumentos de um campo, ou torná-los obrigatórios;
  - tornar um campo não nulo (como marcador `!` — mais sobre isso abaixo);

Porém é possível depreciar esses campos.

- É perfeitamente OK criar tipos que não reflitam exatamente a estrutura do banco de dados (embora a tendência é que a API tenha uma estrutura de dados diferentes da estrutura do banco à medida em que evolui). Cada tipo deve representar um objeto com dados que clientes possam consumir.
- IDs são "anti-patterns" em GraphQL; deve-se sempre trabalhar com o objeto de referência.
- Em campos com valores específicos — por exemplo, no projeto deste curso onde *role* pode ser somente "estudante", "docente" ou "coordenação', a melhor prática é utilizar um tipo `ENUM`. Enum é conhecido e utilizado em diversas linguagens, porém não no JavaScript. Falaremos sobre esse tipo mais adiante neste curso.
- É aconselhável que Mutations tenha um tratamento de erros que passe informações claras ao cliente.

Boas práticas

A seguir, um resumo de práticas comuns em GraphQL.

- **HTTP**: O GraphQL normalmente utiliza o protocolo HTTP para expor todos os recursos da API através de um único endpoint e todas as requisições utilizam POST - inclusive as de consulta. Ao contrário do REST, que é composto de uma série de endpoints, cada um deles expondo um único recurso da API. É possível utilizar o GraphQL para expor recursos em mais de um endpoint, porém essa não é uma prática comum, além de dificultar o uso de ferramentas como o playground — que apesar do nome é muito importante para acessar a documentação da API.

- **JSON**: Com GraphQL, os dados normalmente são retornados no formato JSON, embora isso não seja obrigatório segundo a especificação do GraphQL (http://spec.graphql.org/draft/#sec-Serialization-Format). JSON é uma notação bastante familiar em desenvolvimento web, tanto para quem desenvolve APIs quanto para clientes, e é de fácil leitura. Para questões de performance, é recomendado o uso da compressão com GZIP e o envio das requisições com o header `accept-encoding: gzip` . Para saber mais sobre HTTP, header e GZIP, caso você precise, o curso de fundamentos do HTTP (https://cursos.alura.com.br/course/http-fundamentos) da Alura vai te dar a base.
- **Versionamento**: Embora nada impeça o versionamento de uma API GraphQL, de forma similar ao versionamento das APIs REST, essa não é uma prática recomendada. Ao invés disso, encoraja-se uma evolução contínua do schema.
    - O versionamento de APIs acaba sendo necessário pois no modelo REST qualquer mudança nos dados retornados pela API podem ser considerada uma mudança considerável, e mudanças consideráveis requerem uma nova versão. Então, uma vez que se torna necessária uma nova versão toda vez que se adicionam novas *features* na API, temos um contraponto entre lançar novas versões com modificações incrementais frequentes *versus* a legibilidade e a manutenção da API. *É importante frisar aqui que a comparação está sendo feita entre GraphQL e APIs REST voltadas somente para operações CRUD, sem considerar APIs RESTful mais complexas.*
    - No caso do GraphQL, como só são retornados dados que são explicitamente requisitados, é possível adicionar novos tipos e campos no schema, inclusive novos campos em tipos já existentes, sem que isso "quebre" a API. Por esse motivo, foi estabelecida a prática de não-versionamento de APIs GraphQL. Além disso, é possivel fazer um monitoramento para saber quando um atributo não está sendo mais usado pelo front-end e quando pode ser retirado do schema.
- **tipo NULL**: Por padrão, em GraphQL todos os campos de um tipo podem ser nulos, a não ser quando explicitamente indicado. Isso porque em um serviço como uma API GraphQL há várias pontas que podem falhar: a conexão com o banco, uma ação assíncrona que falhou, entre outros fatores.
    - Dessa forma, para o GraphQL é preferível que um campo possa retornar nulo do que a requisição falhar. Em vez disso, usa-se o modificador `!` para marcar um campo como "não nulo".
    - Quando criamos o schema, é importante ter em mente quando um campo pode retornar nulo em caso de falha, e quando obrigatoriamente deve retornar um erro.
    - Mais sobre campos nulos e não nulos neste link da documentação do GraphQL (https://graphql.org/learn/schema/#lists-and-non-null).

Ainda há mais práticas listadas na documentação, relacionadas a temas que não estamos abordando neste curso.

Caso queira ver cada uma com mais detalhes, pode conferir a parte de práticas na documentação oficial (https://graphql.org/learn/best-practices/) e este artigo sobre boas práticas e design patterns em GraphQL (https://www.moesif.com/blog/api-guide/graphql-best-practices-resources-and-design-patterns/).