## **Banco de Dados**

Segundo Totvs, boa parte dos motivos para se ter um banco de dados está ligado às necessidades e exigências dos consumidores. Eles desejam ter experiências relevantes e que vão além de comprar um item e pagá-lo no caixa.

A personalização do atendimento é muito mais abrangente que isso e pode começar, até mesmo, na seleção de produtos do varejo. Outros motivos que fortalecem sua estratégia são:

* permitir o acesso facilitado dos dados da empresa: seja por não ter uma equipe específica para gestão e análise, seja por ter diversas áreas e filiais que podem usufruir da ferramenta. Um banco de dados garante o compartilhamento de seu acesso e utilização simplificada;
* criar análises e comparativos entre seus dados: o que aumenta consideravelmente seu nível de acerto são os dados estruturados dos bancos, que podem criar relatórios e informações estratégicas para as tomadas de decisão,
* facilitar atualizações e melhorias das informações: com um sistema para gerenciamento de dados, a curadoria dos registros é facilitada. O gestor identifica aqueles que podem ser relevantes para o negócio e outros que podem ser descartados, como antigos parâmetros ou leis que foram modificadas.

A importância do banco de dados é tão indiscutível quanto seu crescimento exponencial. A cada interação entre empresa e cliente, são gerados milhares de registros. Quando organizados, eles podem trazer *insights* fundamentais para a gestão do varejo. Esse crescimento é tão impactante, que a análise de todos esses dados pode ser considerada extremamente complexa.

### **Tipos de banco de dados**

Os bancos de dados podem ser aplicados em diversas áreas de mercado. Além disso, têm algumas tipificações. Para complementar essa análise, é importante entender que eles podem ser do tipo relacional ou não.

**Relacional**

Nesse caso, os registros são tabelados e estruturados em colunas e linhas que podem se relacionar. Tipo muito comum em sistemas de ERP e CRM. Com eles, é possível criar relações valiosas para a tomada de decisão.

Um exemplo seria comparar informações sobre o volume de vendas na loja física e o período de uma campanha promocional na TV. O impacto nos resultados, positivos ou negativos, pode conduzir gestores a insights importantes, como o quanto seu público é sensível àquela mídia.

### **Não relacional**

### Os dados não relacionais não são tão versáteis como os primeiros para análises comparativas, mas são igualmente importantes. Os seus formatos são menos convencionais, como as imagens publicadas nas redes sociais.

Apesar de terem um formato não relacional, também podem ser avaliados. Por exemplo, o número de fotografias tiradas e publicadas nas redes sociais como um sinal de aprovação da marca.

Para o desenvolvimento do projeto foi escolhido o tipo de banco de dados relacional, pois se faz necessários relações entre os dados.

**Linguagem SQL**

Segundo Luís Damas, a linguagem SQL implementa os conceitos definidos no modelo relacional, um modelo largamente aceito e recomendado.

A utilização deste standard internacional reduz as incompatibilidades entre os sistemas e evita que se opte por arquiteturas proprietárias que implicam maiores custos de desenvolvimento e maior esforço financeiro e humano por parte dos intervenientes. Com a linguagem SQL é possível:

* Criar, alterar e remover todas as componentes de uma base de dados, como tabelas, campos, views, índices, etc.;
* Inserir, alterar e apagar dados;
* Interrogar a base de dados, obtendo como resposta o conjunto de registos que obedece às condições indicadas;
* Controlar o acesso dos utilizadores à base de dados e as operações a que cada um deles pode ter acesso;
* Obter a garantia da consistência e integridade dos dados. De notar que a linguagem SQL realiza o conjunto das tarefas enunciadas através de uma linguagem simples, de fácil aprendizagem e implementação.

<https://docplayer.com.br/56639763-14-a-edicao-edicao-atualizada.html>

<https://www.totvs.com/blog/negocios/banco-de-dados/>

<https://www.linkedin.com/pulse/o-que-%C3%A9-um-banco-de-dados-e-qual-sua-import%C3%A2ncia-para-william-miranda?src=aff-lilpar&veh=aff_src.aff-lilpar_c.partners_pkw.123201_plc.adgoal%20GmbH_pcrid.449670_learning&trk=aff_src.aff-lilpar_c.partners_pkw.123201_plc.adgoal%20GmbH_pcrid.449670_learning&clickid=2kD2oh0SVxyOW6YwUx0Mo3IUUkiVXQ0HZzDxVQ0&irgwc=1#:~:text=Para%20as%20empresas%2C%20um%20sistema,invent%C3%A1rios%20e%20muitas%20outras%20informa%C3%A7%C3%B5es.>

<https://www.freecodecamp.org/news/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d/>

**Server Side Rendering e Client Side Rendering**

No desenvolvimento web, existem 2 formas principais de entregar conteúdo ao usuário, que são chamadas de: Client Side Rendering (renderização do lado do cliente) e Server Side Rendering (renderização do lado do servidor).

**Server Side Rendering**

A renderização do lado do servidor é o método mais comum para exibir informações na tela. Ele funciona convertendo arquivos HTML no servidor em informações úteis para o navegador. Sempre que você visita um site, seu navegador faz uma solicitação ao servidor que contém o conteúdo do site. Depois que o processamento da solicitação é concluído, seu navegador retorna o HTML totalmente renderizado e o exibe na tela. Se você decidir visitar uma página diferente no site, seu navegador fará novamente uma solicitação para as novas informações. Isso ocorrerá toda vez que você visitar uma página da qual seu navegador não possui uma versão em cache. Não importa se a nova página tem apenas alguns itens que são diferentes da página atual, o navegador solicitará a nova página inteira e renderizará tudo do zero.

**Client Side Rendering**

Quando os desenvolvedores falam sobre renderização do lado do cliente, eles estão falando sobre renderizar conteúdo no navegador usando JavaScript. Portanto, em vez de obter todo o conteúdo do próprio documento HTML, você está obtendo um documento HTML básico com um arquivo JavaScript que renderizará o restante do site usando o navegador.

Essa é uma abordagem relativamente nova para renderizar sites e não se tornou realmente popular até que as bibliotecas JavaScript começaram a incorporá-la em seu estilo de desenvolvimento. Alguns exemplos notáveis ​​são Vue.js, React.js e Angular.js.

Prós do lado do servidor:

* Os mecanismos de pesquisa podem rastrear o site para melhorar o SEO.
* O carregamento da página inicial é mais rápido.
* Ótimo para sites estáticos.

Contras do lado do servidor:

* Solicitações frequentes do servidor.
* Uma renderização geral lenta da página.
* Recarrega a página inteira.
* Interações de sites não ricos.

Prós do lado do cliente:

* Interações ricas em sites
* Renderização rápida do site após o carregamento inicial.
* Ótimo para aplicativos da web.
* Seleção robusta de bibliotecas JavaScript.

Contras do lado do cliente:

* SEO baixo se não for implementado corretamente.
* O carregamento inicial pode exigir mais tempo.
* Na maioria dos casos, requer uma biblioteca externa.

Pensando em uma aplicação MarketPlace, se faz necessário desenvolver uma aplicação que seja Client Side Rendering, pois os usuários precisam de muitas iterações, conteúdos que são direcionados à determinados tipos de usuários e muitos acessos simultâneos. Uma aplicação como esta proporciona todas essas ações com mais performance, gerando melhores resultados e uma melhor experiência para o usuário.

Mas de onde vêm as informações para o JavaScript renderizar?

Entram em cena então as APIs (Application Programming Interface), que significa em tradução para o português "Interface de Programação de Aplicativos".

<https://www.freecodecamp.org/news/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d/> - 01/09/2020

**API**

Segundo a Red Hat, API é um conjunto de definições e protocolos usado no desenvolvimento e na integração de software de aplicações.

Uma API permite que sua solução ou serviço se comunique com outros produtos e serviços sem precisar saber como eles foram implementados. Isso simplifica o desenvolvimento de aplicações, gerando economia de tempo e dinheiro. Ao desenvolver novas ferramentas e soluções (ou ao gerenciar aquelas já existentes), as APIs oferecem a flexibilidade necessária para simplificar o design, a administração e o uso, além de fornecer oportunidades de inovação.

Uma API é responsável por fornecer apenas dados de acordo com a solicitação que foi feita. Para esta solicitação ser interpretada, é necessário um protocolo de comunicação para enviá-la e para recebê-la, para que seja possível essa comunicação.

Estudaremos 2 formas de comunicação: API Rest e GraphQL.

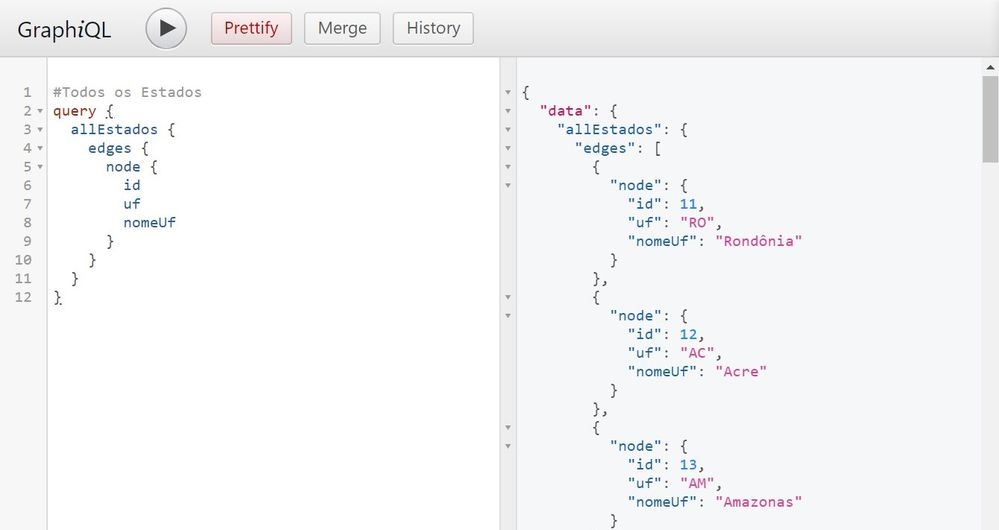
<https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces> - 01/09/2020

**GraphQL**

Segundo GraphQL Fundation, GraphQL é uma linguagem de consulta de dados e especificação desenvolvida internamente pelo Facebook em 2012 antes de ser disponibilizada publicamente em 2015. Ela oferece uma alternativa às arquiteturas baseadas em REST com o objetivo de aumentar a produtividade do desenvolvedor e minimizar a quantidade de dados transferidos.

Segundo a própria documentação, GraphQL é uma linguagem de consulta para APIs e um tempo de execução para atender a essas consultas com seus dados existentes. GraphQL fornece uma descrição completa e compreensível dos dados em sua API, dá aos clientes o poder de pedir exatamente o que precisam e nada mais, torna mais fácil evoluir APIs ao longo do tempo e permite ferramentas poderosas de desenvolvedor.

Como assim os clientes podem pedir exatamente o que precisam? Isso mesmo! Veremos um exemplo abaixo:



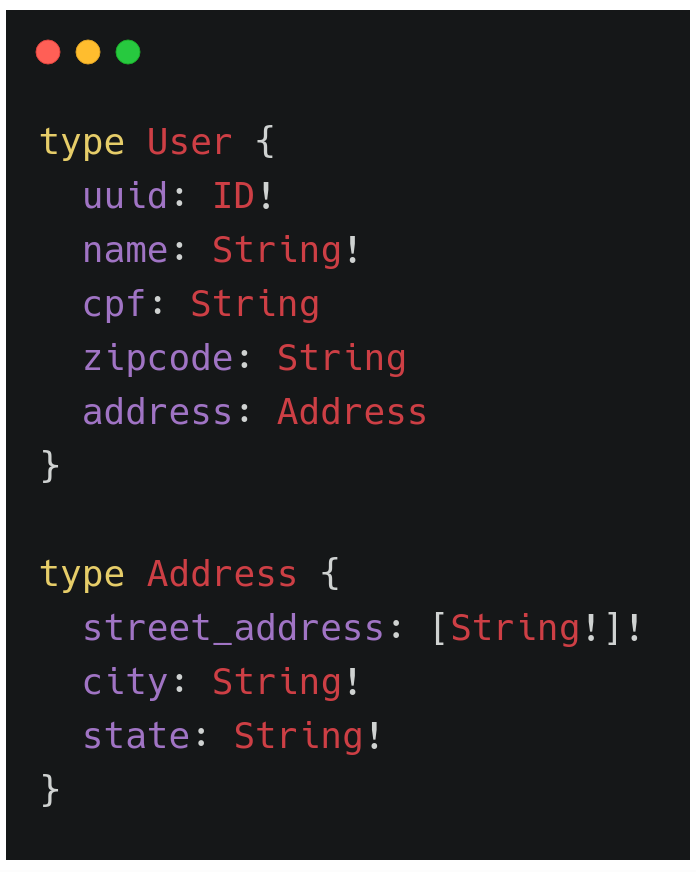
<https://community.qlik.com/t5/image/serverpage/image-id/15003i4BC2FD2C96310D4A/image-size/large?v=1.0&px=999>

Do lado esquerdo vemos um JSON (JavaScript Object Notation) que é enviado no corpo da requisição, e do lado direito temos a resposta dessa requisição. Na requisição, é solicitado todos os estados e os seguintes dados de cada estado: id, uf, nomeUf. Na resposta, temos exatamente isso: um JSON contendo todos os estados e as informações requisitadas de cada estado. Se quiséssemos apenas o id e a uf de cada estado, precisaríamos apenas desconsiderar do exemplo anterior o nomeUf, e ele deixaria de aparecer em nossa resposta.

Isso permite uma liberdade maior para os clientes, e segundo a própria documentação, os aplicativos que usam GraphQL são rápidos e estáveis ​​porque controlam os dados que obtêm, não o servidor.

Estas requisições feitas com GraphQL são todas do tipo POST, e o que vai mudar de uma para a oura é apenas o corpo da requisição, ou seja, nossa query.

Esta query precisa seguir um modelo, que é chamado de Type. Abaixo temos dois exemplos de definição de Type:



https://miro.medium.com/max/1392/1\*QUtpN4hPVp0gR\_3J\_Szf5w.png

Como vemos, temos dois Types, User e Address. Em User, podemos notar que existe um relacionamento com Address, pois foi definida uma propriedade com o tipo Address. Para montarmos uma query solicitando todas as informações de um User com todas as informações de seu Address, podemos fazer da seguinte forma:

query{

user{

uuid,

name,

cpf,

zipcode,

address{

street\_address,

city,

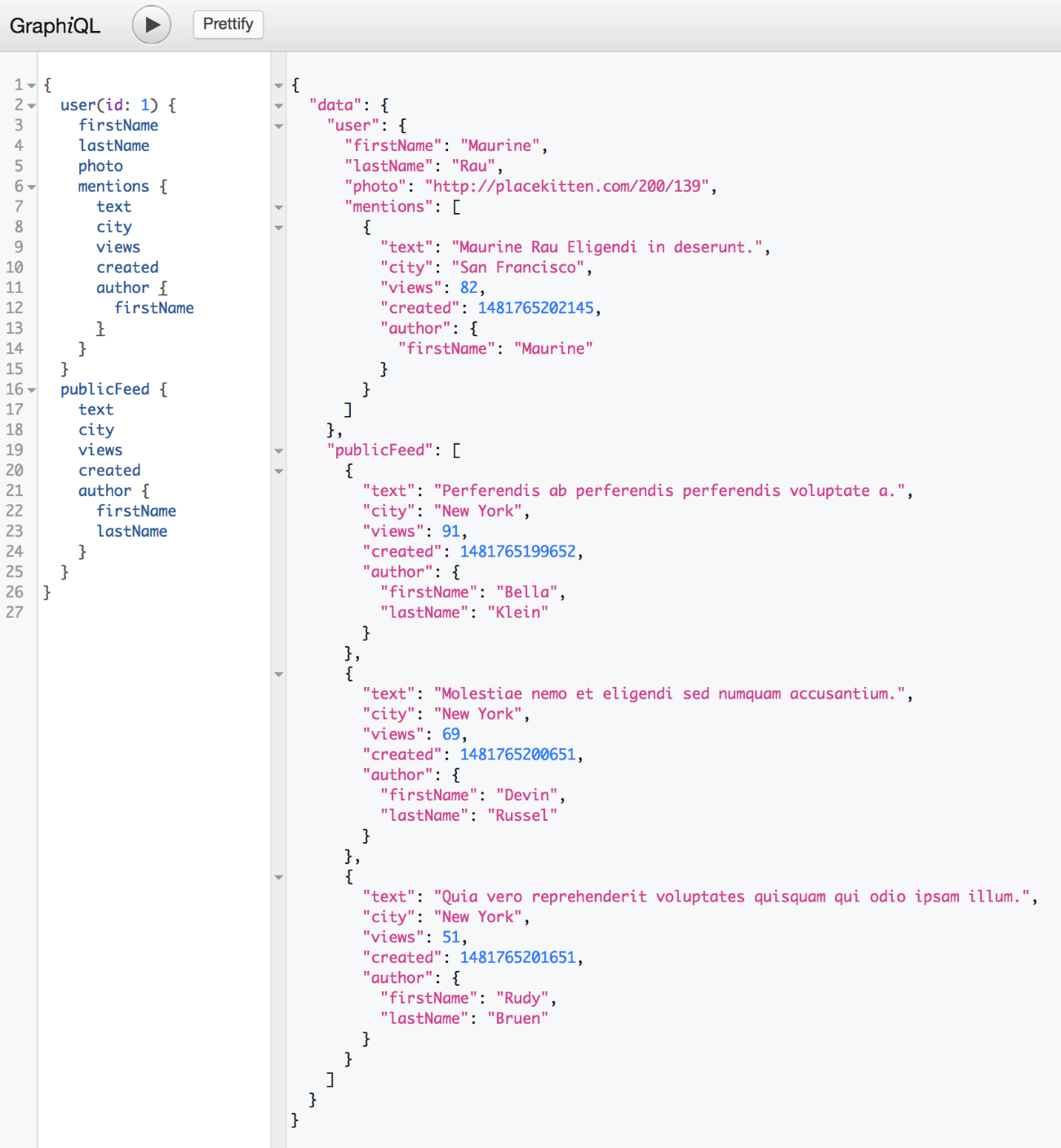
state

}

}

}

Ainda segundo a documentação, as consultas GraphQL acessam não apenas as propriedades de um recurso, mas também seguem suavemente as referências entre eles. Enquanto as APIs REST típicas requerem o carregamento de vários URLs, as APIs GraphQL obtêm todos os dados de que seu aplicativo precisa em uma única solicitação. Abaixo temos um exemplo do que em REST seriam necessárias duas requisições para termos como resposta os mesmos resultados:



https://res.cloudinary.com/dyyck73ly/image/upload/v1487945501/iq7kavaunkt77jaqlxsm.png

Isso torna a aplicação mais rápida, pois conseguimos fazer mais do que uma consulta em apenas uma requisição.

<https://graphql.org/> - 01/09/2020

https://foundation.graphql.org/