UNIVERSIDADE ESTADUAL DE GOIÁS

CÂMPUS ANÁPOLIS DE CIÊNCIAS EXATAS E TECNOLÓGICAS HENRIQUE SANTILLO

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

LEONARDO RIBEIRO

Graphql: Uma alternativa para webservices

Anápolis  
Maio, 2019

UNIVERSIDADE ESTADUAL DE GOIÁS

CÂMPUS ANÁPOLIS DE CIÊNCIAS EXATAS E TECNOLÓGICAS HENRIQUE SANTILLO

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

LEONARDO RIBEIRO

Graphql: Uma alternativa para webservices

Projeto de TC apresentado ao Departamento de Sistemas de Informação Câmpus Anápolis de Ciências Exatas e Tecnológicas Henrique Santillo da Universidade Estadual de Goiás, para obtenção da nota da primeira VA.

Orientador: Prof. Me. Ronaldo de Castro Del-Fiaco

Anápolis

Maio, 2019

Local reservado para a folha de aprovação (Ata de Defesa)

[na versão final de TC, colocar aqui a Ata de Defesa Escaneada, que será entregue pela banca, com boa qualidade de resolução]

FICHA CATALOGRÁFICA

|  |
| --- |
| RIBEIRO, Leonardo.  Graphql: Uma alternativa para webservices / Orientador: Me. Ronaldo de Castro Del-Fiaco  Anápolis,  Maio, 2019  25 p.  Trabalho de Curso (Graduação, Bacharelado em Sistemas de Informação) -- Universidade Estadual de Goiás, Câmpus Anápolis de Ciências Exatas e Tecnológicas Henrique Santillo, Departamento de Sistemas de Informação.  1. GraphQL 2. API 3. REST 4. SOAP |

REFERÊNCIA BIBLIOGRÁFICA

RIBEIRO, Leonardo. **Graphql: Uma alternativa para webservices** Anápolis, 2019. 26 p. Monografia – Curso de Sistemas de Informação, CCET, Universidade Estadual de Goiás.

CESSÃO DE DIREITOS

É concedida à Universidade Estadual de Goiás a permissão para disponibilizar esse documento por meio eletrônico ou reproduzir cópias, emprestar ou vender tais cópias para propósitos acadêmicos e científicos, conforme termo de autorização assinado pelo autor e arquivado na Biblioteca do Câmpus. O autor reserva outros direitos de publicação e nenhuma parte deste trabalho pode ser reproduzida sem a autorização por escrito do autor.

Anápolis

Maio, 2019

Dedico este trabalho a Deus, por sempre estar do meu lado me ajudando a enfrentar os momentos difíceis que passei em minha vida, aos meus pais por sempre estarem me incentivando, aos meus professores, por sempre serem compreensivos em momentos que tive dificuldade, a minha namorada que sempre me motivou a dar o melhor de mim.

**AGRADECIMENTOS**

Fazer aqui os agradecimentos que o aluno achar adequando.

Observar a linguagem formal e culta.

# RESUMO

Palavras-chave: GraphqQL, APIs, REST, SOAP.

Durante muito tempo desenvolvedores de software têm utilizado tecnologias como REST e SOAP no desenvolvimento de suas APIs. No presente trabalho, será apresentada uma nova tecnologia proposta e utilizada pela empresa Facebook para facilitar a construção de APIs, essa tecnologia é chamada de GraphQL.

O principal objetivo do trabalho é apresentar de maneira sucinta como funciona essa tecnologia e quais suas vantagens comparadas com os já existentes formatos de construção de APIs, como REST e SOAP.

A metodologia utilizada para detalhar o funcionamento dessa nova tecnologia será através de um estudo de caso, onde será construído um pequeno sistema utilizando APIs REST, em seguida a mesma aplicação seria construída utilizando GraphQL, apontando os diferentes pontos entre ambas.

No final desse trabalho será apresentado o resultado comparativo dentre as tecnologias estudadas referente a performance de desenvolvimento, pontos positivos e pontos negativos do ponto de vista do leitor, quantidade de linhas escritas de código para cada metodologia, dificuldade de aprendizado e o que se pode esperar no futuro dessas tecnologias, também no ponto de vista do autor.

A partir dos resultados citados, espera-se que seja o suficiente para ajudar profissionais que trabalham no ramo de desenvolvimento de software a tomarem melhores decisões quanto a definição e construção de APIs, levando uma maior produtividade aos desenvolvedores.

# ABSTRACT

For a long time software developers have been using technologies like REST and SOAP to develop their APIs. In the current work, it’ll be presented a new technology proposed and used by Facebook to make it easier to build APIs, this technology is called GraphQL.

The main goal of this document is to present in a succinct way how this technology works and what’s its advantages compared with the existing ways to build APIs, like REST and SOAP.

The methodology used to detail the operation of this new technology will be through a case study, where will be built a small system using REST APIs following by the construction of the same application built using GraphQL pointing the different points on both applications.

At the end of this work will be presented the comparative results between the studied technologies about development performance, positive and negative details in the author’s point of view, how many code lines written for each methodology, learning difficulty and what can be waited for the future of these technologies.

From the results quoted, it’s expected that it could be enough to help professionals who works in software development area to take better decisions regarding the construction and definition of their APIs, taking more productivity to them.

Keywords: GraphqQL, APIs, REST, SOAP.

# LISTA DE FIGURAS

[Figura 1 - Pesquisas do google sobre GraphQL e REST API 19](#_Toc10492750)

[Figura 2 - Corpo de um documento XML para mensagens SOAP Fonte: Adaptado de https://www.w3schools.com/XML/xml\_soap.asp 25](#_Toc10492751)

[Figura 3 - Linguagem GraphQL para consulta de um](#_Toc10492752) *[endpoint](#_Toc10492752)* [chamado](#_Toc10492752) *[search](#_Toc10492752)* [27](#_Toc10492752)

[Figura 4 - Definindo tipos num](#_Toc10492753) *[schema](#_Toc10492753)* [do GraphQL 28](#_Toc10492753)

[Figura 5 - Query para solicitar um artigo em GraphQL 28](#_Toc10492754)

[Figura 6 - Resposta da requisição GraphQL na Figura 4 29](#_Toc10492755)

[Figura 7 -](#_Toc10492756) *[Mutation](#_Toc10492756)* [no GraphQL 29](#_Toc10492756)

# LISTA DE ABREVIATURAS E SIGLAS

**Siglas Descrição**

API *Application Programming Interface* - Interface de programação da aplicação

REST *Representational State Transfer* - Transferência de estado representacional

SOAP *Simple Object Access Protocol* - Protocolo de acesso de objeto simples

JSON *JavaScript Object Notation* – Notação de objetos javascript

SLOC *Source Lines of Code –* Linhas de código fonte

SUMÁRIO

[RESUMO 9](#_Toc10492838)

[ABSTRACT 10](#_Toc10492839)

[LISTA DE FIGURAS 11](#_Toc10492840)

[LISTA DE ABREVIATURAS E SIGLAS 12](#_Toc10492841)

[INTRODUÇÃO 14](#_Toc10492842)

[2. Referencial Teórico 15](#_Toc10492843)

[3. World Wide Web 17](#_Toc10492844)

**[3.1.](#_Toc10492845)****[Uma breve história](#_Toc10492845)** [18](#_Toc10492845)

[4. Webservices 19](#_Toc10492846)

[4.1. O que é um Web Service? 19](#_Toc10492847)

[4.2. Tipos de Web Services 21](#_Toc10492848)

[4.2.2. REST 22](#_Toc10492849)

[5. Node.js 22](#_Toc10492850)

[6. GraphQL 23](#_Toc10492851)

[6.1. O que é GraphQL? 23](#_Toc10492852)

[6.2. GraphQL Schema 25](#_Toc10492853)

[6.3. Query 25](#_Toc10492854)

[6.4. Mutation 26](#_Toc10492855)

[CONCLUSÃO 27](#_Toc10492856)

[REFERÊNCIAS 28](#_Toc10492857)

[GLOSSÁRIO 29](#_Toc10492858)

[APÊNDICES 30](#_Toc10492859)

[Apêndice A – Cronograma previsto e realizado 30](#_Toc10492860)

[Apêndice B - Banner [ou Artigo] apresentado na Semana de Comunicação Científica 31](#_Toc10492861)

[ANEXOS 33](#_Toc10492862)

[Anexo A 33](#_Toc10492863)

# INTRODUÇÃO

A evolução da internet tem se manifestado de muitas formas: as características de tráfego, a interconexão entre topologias, o relacionamento entre negócios e componentes autônomos, a facilidade do acesso a informação, entre outros. Até hoje essa evolução vem acontecendo de maneiras inesperadas em muitos aspectos. Começando com a forma que as aplicações web eram moldadas nos anos 2000 e como são construídas atualmente. É importante entender o por quê (e como) dessa evolução estar acontecendo, e como isso pode afetar drasticamente como os usuários navegam, a interação das empresas que lidam com aplicações web e com os desenvolvedores dessas aplicações (MA; LUI; MISRA, 2015).

Graças ao trabalho árduo de desenvolvedores, a internet foi criada, e através dela é possível facilitar o compartilhamento de informações. Porém, esse era o objetivo inicial de quando a internet surgiu. Hoje, além de compartilhar informações entre alguns departamentos dentro das empresas, também usa-se a internet como forma de entretenimento, criar novos negócios, realizar compras e vendas de produtos e/ou serviços, ou até mesmo gerir a internet em si (MA; LUI; MISRA, 2015).

A maneira mais comum que é utilizado para fazer isso é através do desenvolvimento de aplicações web, o qual o usuário é capaz de interagir através de entrada de dados (clicks, textos em formulários, arquivos) e esperam receber uma saída de dados ou um feedback dessa aplicação. Esse feedback podem ser informações já inseridas anteriormente, download de algum arquivo, ou qualquer informações que supra a necessidade do usuário (MULLOY, 2012).

Para desenvolver essas aplicações, é utilizado os chamados *webservices, que servem*  para gerir essas requisições que o usuário faz através de uma aplicação web. Esse *webservice* nada mais é do que um serviço com um ponto de entrada (uma *url*) onde são mandadas informações pelo usuários. Essas informações são processadas e retornam um *feedback* sobre aquela requisição (BOREK *et al.*, 2015).

A maneira com a qual esses *webservices* foram desenvolvidos também evoluíram com o tempo. Sua principal mudança sempre teve como foco a melhoria da produtividade dos desenvolvedores para construção dos *webservices*, performance de processamento dos dados enviados nas requisições, automação na produção de documentação e processamento mais amigável tanto para máquinas quanto para humanos que interagem com o *webservice* (BOREK *et al.*, 2015).

Até o presente momento, essas aplicações eram desenvolvidas utilizando tecnologias como *REST* e *SOAP*. O problema de utilizar essas tecnologias é que às vezes era necessário um nível abstração baixo e sem controle na resposta das requisições. Imagine, por exemplo, que um usuário fizesse uma requisição em uma *url* e essa retornasse uma quantidade imensa de dados, dos quais não seriam usados nem a metade. Além disso, o desenvolvedor teria uma resposta no formato *JSON*, que deveria ser interpretada pelo cliente (*frontend* da aplicação, parte que o usuário vê), para depois então descartar a maioria desses dados. Por causa dessa requisição, foram consumidos recursos de internet, cliente e servidor (BRITO; MOMBACH; VALENTE, 2019).

Neste trabalho, o objetivo é apresentar uma das tecnologias desenvolvidas para construção de *webservices*, proposta e desenvolvida recentemente pelo Facebook em 2016, chamada GraphQL. Essa tecnologia é uma linguagem de consulta proposta para implementar APIs (*Application Programming Interface*) baseadas em aplicações web. Essa tecnologia é capaz de solicitar, em uma única requisição, respostas complexas apenas com os dados necessários para o cliente, evitando consumo desnecessário de recursos, consequentemente melhorando a performance da requisição. Além disso, essa tecnologia trás uma abordagem diferente na construção de *APIs*, melhorando também a produtividade do desenvolvedor, que será detalhada mais adiante (BRITO; MOMBACH; VALENTE, 2019).

# 2. Referencial Teórico

No Brasil ainda não há muitas fontes com abordagens introdutórias e comparativas aos modelos existentes de *web services*. Portanto no presente trabalho serão utilizadas fontes estrangeiras na maioria, analisando e abstraindo dos autores detalhes significativos para quem ainda não conheça a tecnologia mas gostaria de uma alternativa produtiva para desenvolvimento de APIs.

Kambona (2016) faz uma abordagem mais introdutória sobre GraphQL, além de também apresentar ferramentas auxiliares para se trabalhar em conjunto com a tecnologia. Brito (2019), por outro lado, tem o objetivo de auxiliar técnicos que já trabalham com tecnologia da informação a migrarem de suas arquiteturas REST para GraphQL. Hartig (2017) leva o assunto de uma forma mais analítica sobre como GraphQL pode ser benéfico ou não para desenvolvedores.

O que se pode ver em comum na maioria das fontes utilizadas neste trabalho é que há uma tendência de uso da ferramenta para os próximos anos, considerando que, caso contrário, não haveriam livros e trabalhos acadêmicos com tais abordagens desde a publicação da proposta do uso da ferramenta pelo facebook em 2016.

Também é possível validar essa tendência através da ferramenta Google Trends:

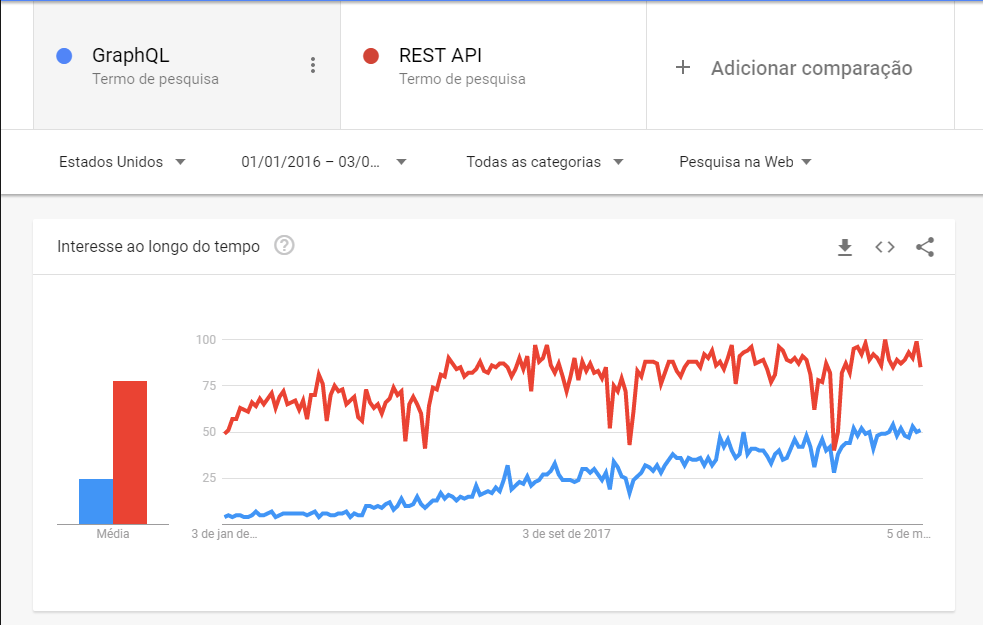


Figura 1 - Pesquisas do google sobre GraphQL e REST API

Fonte: Produzido pelo Autor em https://trends.google.com

Uma problemática interessante que os autores deixam implícito em seus trabalhos é como a produtividade dos desenvolvedores que utilizam a tecnologia GraphQL aumentam em relação ao desenvolvimento de *web services* com arquitetura baseada em REST. Porém isso não é apresentado de forma direta.

Portanto, o objetivo deste trabalho é apresentar, quantitativamente utilizando técnica de SLOC (*Source Lines of Code* – Linhas de código fonte), uma metodologia sugerida no trabalho de Nguyen (2007) para medir o desenvolvimento de um software.

Considerando que quanto menos linhas de código escritas por um desenvolvedor significa que ele é capaz de desenvolver mais rápido, essa foi a maneira escolhida para validar a hipótese de que GraphQL pode ser mais produtivo, comparado a aplicações que utilizam arquitetura REST. Também serão comparados tempo de requisição e resposta do servidor e tamanho de bytes retornados, que são métricas disponíveis em qualquer browser moderno.

A metologia utilizada para testar e provar a hipótese do presente trabalho será o estudo de caso, onde o autor desenvolverá 2 (duas) aplicações similares, sendo uma utilizando arquitetura REST e outra utilizando GraphQL, assim será possível fazer uma comparação mais realística de como ambas apresentam as métricas citadas.

A coleta dos dados serão feitas, na maioria, manualmente. Exceto pela quantidade de linhas de código, que pode ser adquirida de através de uma ferramenta *git* com o comando “git ls-files | xargs wc -l”, que poderá listar todos os arquivos do projeto e a quantidade de linhas de código de cada um, em uma ferramenta de linha de comando como *bash* (do Linux), terminal (do MacOS) ou *cmd* e *powershel* (do Windows). Após a listagem desses documentos, basta fazer a somatória da quantidade de linhas de cada arquivo.

Para coletar o tempo de requisição e tamanho de bytes, será utilizado o *browser* Google Chrome, que possui ferramentas para desenvolvedores analisarem as atividades de rede, que contém dados de cada requisição feita do cliente para o servidor, incluindo o tempo de requisição e tamanho da resposta do servidor.

Após a coleta dos dados, serão criados gráficos comparando ambas as aplicações lado a lado, tendo assim uma maneira fácil de validar as hipóteses apresentadas.

# World Wide Web

Para entendermos melhor o funcionamento das aplicações web, precisamos antes entender o que é exatamente web e como surgiu o famoso “*www*”. Para isso, a seguir, vamos abordar uma breve história sobre o surgimento da internet, sua evolução até o dia de hoje e depois, também abordaremos um pouco sobre como são criados os web sites e aplicações web que acessamos diariamente.

* 1. **Uma breve história**

O primeiro passo para o que hoje conhecemos como internet começou no dia 29 de Agosto de 1949, quando a União Soviética detonou sua primeira arma nuclear, um evento marcante no pós-guerra entre seu confronto com os Estados Unidos, na segunda guerra mundial. Como os aviões eram a única maneira de levar uma bomba nuclear naquela época, a defesa aérea se tornou uma preocupação e prioridade imediata para os Estados Unidos. Em 1948, a força aérea já havia estudado a necessidade de um sistema de redes para notificação antecipada para a defesa aérea. Para isso, o professor do *Massachussetts Institute of Technology* (Instituto Tecnológico de Massachussetts), George Valley foi nomeado para recomendar um sistema melhorado que resolvesse as deficiências da rede de radares da época, o qual deveria ser montado as pressas. Os dados gerados pelos radares eram escritos a mão utilizando lápis de graxa em sentido inverso em placas transparentes (LUKASIK, 2011).

Para automatizar o processo manual, a ADSEC (*Air Defense System Engineering Committee* - Comitê de Engenharia do Sistema de Defesa Aérea) recomendou a automação desse processo através da digitalização dessas informações. Essas informações seriam enviadas através das redes de telefones ao invés de serem escritas manualmente, e esses dados seriam analisados pelos novos computadores digitais sendo desenvolvidos. Porém isso ainda era só uma proposta e não uma realidade na época. (LUKASIK, 2011).

Enquanto isso, no mesmo período a Coréia do Norte invadia a Coréia do Sul em 25 de Junho de 1950 e quando a China interveio em 26 de Novembro do mesmo ano, trouxe um espectro de guerra nuclear com um mundo comunista ainda mais próximo, a urgência da força aérea americana para um sistema de radares mais efetivo aumentou. Vários projetos foram sugeridos, porém nenhum tão efetivo até 1951, quando foi criado o Projeto Lincoln, onde, para validar as hipóteses apresentadas, um “bombardeiro” foi interceptado em tempo real por um “lutador” utilizando dados transferidos dos radares por linha telefônica (LUKASIK, 2011).

A pesquisa dos Estados Unidos continuou e em 1958, foi estabelecida a ARPA (*Advanced Research Projects Agency* – Agência de Projetos de Pesquisa Avançados) pelo governo dos Estados Unidos como resposta a crise de segurança do pós-guerra da segunda guerra mundial. A ARPA era responsável por desenvolver pesquisas relacionadas a comunicação, navegação, defesa de mísseis balísticos, sistemas de rastreamentos por satélite, lançamento de mísseis, dentre outras disciplinas acadêmicas que oferecessem suporte ao meio militar do país (LUKASIK, 2011).

Em 1960 a ARPA deu seus primeiros passos, quando Joseph Carl Robnett Licklider – um cientista da computação, juntamente com a companhia Bolt, Berenek and Newman (BBN) – formularem ideias para a construção de uma rede global em vários memorandos, descrevendo uma “Rede Intergaláctica de Computadores”. Essas ideias levaram Licklider a se tornar diretor da ARPA. Sua ambição era que todos os computadores existentes pudessem ter alguma forma de se conectar por todo o globo, e todos próximos a um computador poderiam compartilhar informações. (PALERMO; COX, 2014).

Em 1969, a primeira mensagem através de uma rede de computadores foi enviado utilizando o protocolo IMP (Interface Message Processor) por Leonard Kleinrock – um dos contribuidores para o projeto de desenvolvimento da Arpanet – na UCLA (*University of California, Los Angeles*), onde o conteúdo era apenas “LO”. A mensagem foi recebida por Elizabeth Feinler no SRI (*Standford Research Institute*), em seguida a rede caiu, porém essa foi uma mensagem simples, que entrou para a história (LUKASIK, 2011).

Depois dessa primeira mensagem, muitas melhorias foram feitas até a rede ficar mais estável. Em 1983, a Arpanet foi dividia em duas partes, uma rede de pesquisa que continuou sendo chamada de Arpanet, e a Milnet utilizada para atividades militares. Em 1989 a Arpanet foi descontinuada, porém não foi evitada sua metamorfose no que conhecemos hoje como “Internet”. A intenção da Arpanet era apenas demonstrar sua utilidade para comutação de pacotes no meio militar, o qual foi um sucesso. Como consequência inesperada, a Arpanet e sua sucessora, a Internet, demonstraram ainda mais a utilidade geral do trabalho em rede para "comando e controle" muito além das necessidades militares (LUKASIK, 2011).

# 4. Webservices

Neste tópico será abordado o uso de *webservices* (serviços web), que são utilizados para realizar operações para aplicativos *web*, *mobile* (para celulares) e *desktop*. Também será abordado como funcionam essas operações e como elas interagem com os aplicativos de forma independente.

### 4.1. O que é um Web Service?

Segundo Fensel (2002), *web services* conectam computadores e dispositivos um com o outro utilizando a internet para trocar e combinar dados. *Web* *service* pode ser definido como um conjunto de objetos de software que podem ser montado de maneira distribuída pela internet utilizando protocolos padrões de comunicação para executarem funções ou processos de negócio completos.

Para que isso aconteça, portanto, devemos ter um computador ou dispositivo que faz uma requisição (cliente) e um computador que processa essa requisição (servidor), assim retornando o conjunto de dados processados a quem fez a requisição.

De acordo com Bouguettaya (2013), uma aplicação distribuída é geralmente criada baseada em processos de negócio, que consistem de uma sequência lógica de ações que podem incluir a invocação de um *Web Service*. Esses processos podem ou não estar diretamente ligados ao sistema sendo construído, ou seja, um sistema pode utilizar um serviço de um outro sistema não relacionado. Mumbaikar (2013) ainda define que:

“…estamos utilizando um *web service* que é uma maneira padrão de distribuir serviços pela internet. Os clientes não têm nenhum conhecimento prévio sobre o *web service* antes deles realmente usarem, assim sendo *WEB SERVICE* é uma plataforma independente e fracamente acoplada”.

Isso dá a entender que um *Web Service* permite a um provedor encapsular funcionalidades e tornar isso disponível por uso de uma rede de internet. Um cliente pode, por sua vez, invocar um *web service* e usar suas funcionalidades. Se for combinado vários *web services* existentes de diferentes provedores, é possível criar uma nova e mais complexa aplicação distribuída, além de poder oferecer um novo serviço composto com maior valor agregado.

Bouguettaya (2013) ainda evidencia que os *web services* podem ser um conjunto de transações, que são abstraídas de processos de negócios. E cada transação consiste de um conjunto de operações (“unidades de trabalho”) que são executadas dentro de um sistema. Para que essas transações sejam executadas, é necessário que o sistema provedor do serviço esteja em um estado consistente para garantir a integridade dos dados processados.

Levando em consideração as definições de Bouguettaya (2013), pode-se então entender que os serviços oferecidos pelos provedores são, nada menos que um conjunto de operações funcionando em conjunto para realizar uma transação, sendo essa transação uma abstração de regras de negócio.

### 4.2. Tipos de Web Services

Atualmente há vários tipos de *web services* existentes, porém para o contexto do presente trabalho, falaremos sobre SOAP e REST, que são os tipos mais comumente utilizados pelas companhias.

#### 4.2.1. SOAP

Segundo Fensel (2002, p. 115), SOAP (Simple Object Access Protocol) é uma especificação de *layout* que define uma maneira uniforme de passar dados através de um documento no formato XML. Além disso, também define uma forma de configurar o protocolo HTTP como a maneira mais básica de comunicação de mensagens SOAP entre dois *endpoints*, sendo esses o cliente e o servidor.

Para que essa comunicação aconteça então, é necessário que o sistema desenvolvido seja capaz de abstrair as informações necessárias e serializá-las em um documento no formato especificado, no caso, XML. Em seguida, esse documento seria enviado através de uma conexão utilizando o protocolo HTTP para um cliente ou servidor, dependendo de quem faz a requisição e quem processa.

Mumbaikar (2013, p. 1) ainda especifica que um *web service* baseado em SOAP necessita de 3 entidades, sendo elas: o provedor do serviço, que é responsável pro oferecer e executar os serviços; O registro de serviços, que é também oferecido pelo provedor como forma de deixar explícito ao consumidor quais serviços têm disponível; E, por fim, o consumidor do serviço que é quem faz a solicitação dos serviços.

Abaixo segue um exemplo do formato de documento de mensagem SOAP:

Uma imagem contendo captura de tela

Descrição gerada automaticamente

Figura 2 - Corpo de um documento XML para mensagens SOAP  
Fonte: Adaptado de https://www.w3schools.com/XML/xml\_soap.asp

### 4.2.2. REST

REST (*representational state transfer*) foi apresentado por Roy Fielding como uma forma alternativa a arquiteturas SOAP, em que um cliente envia uma requisição ao servidor, o servidor então processa essa requisição e retorna uma resposta. Essas requisições e respostas giram em torno de “transferência de representações de recursos” (MUMBAIKAR; PADIYA, 2013).

Mumbaikar (2013) ainda afirma que esse recurso é identificado por uma URI (*Uniform Resource Identifier*). Geralmente REST possui menos detalhes do que o SOAP em seu documento, além de não precisar que seja definido cabeçalhos, assim como é com SOAP. O formato do documento também não precisa ser necessariamente XML, sendo assim utilizado formatos mais simples como JSON.

Geralmente *web services* desenvolvidos com arquitetura REST, utilizam os métodos do protocolo HTTP: GET, PUT, POST e DELETE. Através destes métodos, o cliente solicita as respectivas operações para criar, atualizar, listar e deletar recursos.

# 5. Node.js

No presente trabalho será utilizada a tecnologia Node.js para desenvolver um servidor de aplicação no tópico de estudo de caso, portanto, faremos uma breve introdução sobre o que é e como funciona essa tecnologia. Porém, segundo a documentação do GraphQL encontrada em <https://graphql.org/code/>, há implementações da tecnologia nas linguagens: C# / .NET, Clojure, Elixir, Erlang, Go, Groovy, Java, JavaScript, PHP, Python, Scala e Ruby.

De acordo com Tilkov (2010, p. 1), *Node.js*, também chamado de *Node*, é um ambiente javascript para desenvolvimento de aplicações no lado do servidor. Foi baseado na implementação do *runtime* da Google, chamado motor “V8”, utilizado para interpretação da linguagem javascript no famoso browser Google Chrome. O Node e o V8 são ambos implementados nas linguagens de programação C e C++, focando em performance e baixo consumo de memória. Com o uso do V8 portanto, foi possível para desenvolvedores utilizarem uma linguagem de programação, inicialmente criara para desenvolvimento em *frontend* (cliente), agora no *backend* (servidor) também.

# 6. GraphQL

Neste tópico será evidenciado as principais motivações para a criação de mais uma tecnologia para lidar com requisições em aplicações que utilizam a rede para comunicação e execução de serviços. Serão abordados detalhes sobre o que é e quais os principais propósitos dessa proposta criada pela companhia bilionária e rede social mais utilizada do mundo, o Facebook (BRITO; MOMBACH; VALENTE, 2019).

### 6.1. O que é GraphQL?

Brito (2019) afirma que GraphQL é uma linguagem de consulta, proposta pelo Facebook em 2016, representando uma alternativa para APIs basedas em arquitetura REST. Para melhor entendimento sobre o que é essa proposta, o autor apresenta o seguinte exemplo: Suponha-se que uma API REST utilizada por uma universidade possua um *endpoint* chamado *search*, que permite ao usuário buscar metadados sobre uma pré-impressão dado como parâmetro seu título. O resultado é um complexo e gigantesco documento no formato JSON, com pelo menos 33 campos. Entretanto, o cliente talvez precise apenas de alguns campos específicos (o formato do papel, por exemplo). Apesar disso, o *endpoint* irá retornar todos os 33 campos mesmo assim, que em seguida deve ser analisado e formatado pelo cliente. Agora vamos supor que a API decida suportar GraphQL, assim utilizando-se a linguagem, o cliente poderia formular uma consulta da seguinte forma.



Figura 3 - Linguagem GraphQL para consulta de um *endpoint* chamado *search*

Fonte: BRITO; MOMBACH; VALENTE, 2019, p. 1

De acordo com Bryant (2018), entretanto, a linguagem de consulta GraphQL foi desenvolvida e passou a ser utilizada pelo Facebook em 2012, sendo então levada a público em 2015, com um lançamento nomeando a mesma como uma especificação de rascunho de linguagem. Dando assim a entender que a companhia já estava trabalhando na proposta e tecnologia há algum tempo, porém ainda havia melhorias e correções a serem abordadas, já que ainda era um “rascunho”. O autor ainda aponta os principais objetivos da proposta, que eram: reduzir a sobrecarga de dados relacionados a *web services* do tipo REST, em termos de quantidade de dados desnecessários transferidos e a quantidade de requisições para poder fazer isso; reduzir o potencial de erros causados por requisições inválidas pelo cliente; suportar um modelo evolutivo de dados sem a necessidade de versionamento da API.

Com base no tamanho da companhia e da rede social em si, faz sentido o desenvolvimento de uma ferramenta que pudesse facilitar a validação de requisições e a manutenção evolutiva de APIs existentes.

Além de ser uma linguagem, também temos o lado técnico do GraphQL, sendo apontado por Kambona (*et al.*, 2016, p. 18) como uma camada de execução que um servidor da aplicação pode utilizar para entender e responder às várias requisições feitas utilizando a linguagem GraphQL. Ou seja, além da linguagem também temos um interpretador, afinal, todos podemos inventar uma nova linguagem e começar a falar, mas ninguém entenderia sem aprender e interpretar a linguagem antes.

Para realizar uma operação utilizando GraphQL, podemos ter uma consulta (operação de leitura), ou uma mutação (operação de escrita). Para ambos os casos temos uma string em que a camada de execução interpreta e executa no nosso servidor, então respondendo em um formato específico, sendo este o JSON, o formato mais utilizado por aplicações *mobile* e web. (KAMBONA *et al.*, 2016)

### 6.2. GraphQL Schema

Define a estrutura e o significado do modelo de dados exposto, assim como, as funções de manipulação dos dados. É onde definimos tipos e diretivas sobre o que queremos que o servidor seja capaz de suportar. (KAMBONA *et al.*, 2016)

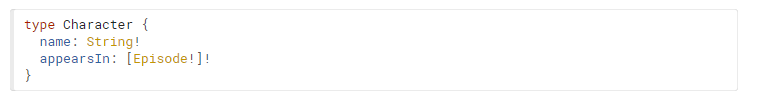


Figura 4 - Definindo tipos num *schema* do GraphQL

Fonte: TEAM (2016)

Como pode-se ver na imagem acima, também é possível assumir que GraphQL trabalha com atributos com tipos diferentes. No caso de *name*, temos um tipo *string* (texto) obrigatório, marcado pelo ponto de exclamação. No campo *appearsIn*, temos um tipo *array* de *Episode* também obrigatório. Kambona (2016, p. 65) confirma isso afirmando que GraphQL é uma linguagem de forte tipagem, o que ajuda o servidor GraphQL determinar se uma *query* é válida em tempo de execução, facilitando assim para o desenvolvedor algumas validações que ele teria que escrever manualmente.

### 6.3. Query

Uma *query*, de acordo com Frisendal (2018), é um documento contendo a linguagem GraphQL com o formato da resposta desejada. Kambona (2016) apresenta um exemplo interessante, onde é possível delimitar exatamente o que o desenvolvedor quer receber como resposta do servidor:

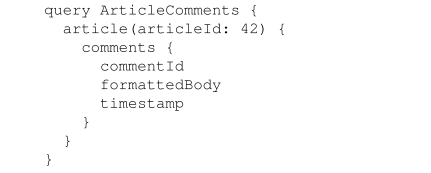


Figura 5 - Query para solicitar um artigo em GraphQL

Fonte: KAMBONA (*et al.*, 2016, p. 43)

E a seguir a resposta do servidor baseada no que foi definido na *query* escrita na imagem acima:

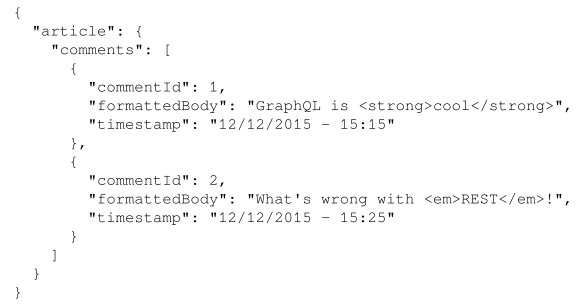


Figura 6 - Resposta da requisição GraphQL na Figura 4

Fonte: KAMBONA (*et al.*, 2016, p. 44)

### 6.4. Mutation

Uma *mutation* (mutação) é similar a uma *query*, mas no tempo de execução temos alguns efeitos colaterais responsáveis por fazer mutações numa estrutura dados, como solicitado no cliente. Essas mutações podem ser a criação de um novo item no banco de dados, a atualização dos dados dessa estrutura ou a exclusão desse item. Essa mutação é capaz de executar uma série de ações em paralelo, como é o caso de alteração de itens interligados através de uma chave-estrangeira num banco de dados relacional. (KAMBONA *et al.*, 2016)



Figura 7 - *Mutation* no GraphQL

Fonte: KAMBONA (*et al.*, 2016, p. 54)

# CONCLUSÃO PARCIAL

Até o presente momento, foi possível ser aprimorado o conhecimento do autor no tópico abordado no presente trabalho, considerando que o mesmo tinha apenas conhecimento superficial e havia utilizado a tecnologia em apenas algumas ocasiões. Num ponto de vista pessoal, GraphQL dá a sensação de ser mais amigável para o desenvolvedor, afinal quando se escreve uma *query* é como se tivesse conversando com a máquina de uma forma mais inteligível.

Atualmente, várias companhias já estão utilizando GraphQL, entre elas, gigantes como o próprio Facebook (criador da tecnologia), Atlassian, GetNinjas, Coursera, GitHub, PayPal, Pinterest, Product Hunt, Twitter, Yelp, entre outras. (TEAM, 2016)

Embora muitas empresas estejam utilizando a tecnologia, não significa que todos devem utilizar agora, pois REST e SOAP ainda são as alternativas de *web services* mais utilizadas do mundo e continuarão assim por um bom tempo. E para quem está começando é possível utilizar essas arquiteturas em conjunto, e caso haja interesse, realizar a migração de forma incremental, caso haja vantagem para o negócio. (BRITO; MOMBACH; VALENTE, 2019)

# REFERÊNCIAS

BOREK, Marian *et al.* *Correct Software in Web Applications and Web Services*. [S.l: s.n.], 2015. Disponível em: <http://link.springer.com/10.1007/978-3-319-17112-8>.

BOUGUETTAYA, Athman; SHENG, Quan Z.; DANIEL, Florian. *Advanced web services*. [S.l: s.n.], 2013. v. 9781461475.

BRITO, Gleison; MOMBACH, Thais; VALENTE, Marco Tulio. Migrating to GraphQL: A Practical Assessment. *SANER 2019 - 26th edition of the IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2019.

BRYANT, Mike. GraphQL for archival metadata: An overview of the EHRI GraphQL API. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, v. 2018- Janua, p. 2225–2230, 2018.

FENSEL, Dieter; BUSSLER, Christoph. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, v. 1, n. 2, p. 113–137, 2002.

FRISENDAL, Thomas; FRISENDAL, Thomas. *Visual Design of GraphQL Data*. [S.l: s.n.], 2018.

KAMBONA, Kennedy *et al.* *Learning GraphQL and Relay*. [S.l: s.n.], 2016. Disponível em: <http://dl.acm.org/citation.cfm?id=2307823%5Cnhttp://doi.acm.org/10.1145/2489798.2489802>.

LUKASIK, S J. “Why the Arpanet Was Built” (vol 33, pg 3, 2011). *Ieee Annals of the History of Computing*, v. 33, n. 4, p. 3, 2011.

MA, Richard T.B.; LUI, John C.S.; MISRA, Vishal. Evolution of the internet economic ecosystem. *IEEE/ACM Transactions on Networking*, v. 23, n. 1, p. 85–98, 2015.

MULLOY, Brian. Web API Design. 2012.

MUMBAIKAR, Snehal; PADIYA, Puja. Web Services Based On SOAP and REST Principles. *International Journal of Scientific and Research Publications*, v. 3, n. 5, p. 3–6, 2013. Disponível em: <www.ijsrp.org>.

NGUYEN, Vu *et al.* A SLOC Counting Standard. *COCOMO II Forum.*, v. 2007, p. 1–15, 2007.

PALERMO, Elizabeth; COX, Lauren. *Who Invented the Internet? | ARPANET*. Disponível em: <https://www.livescience.com/42604-who-invented-the-internet.html>. Acesso em: 27 abr. 2019.

TEAM, Facebook. *GraphQL Documentation*. Disponível em: <https://graphql.org/learn/schema/>.

TILKOV, Stefan; VINOSKI, Steve. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, v. 14, n. 6, p. 80–83, 2010.

# APÊNDICES

## Apêndice A – Cronograma previsto e realizado

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N.º | ATIVIDADE | Cronograma de execução do Trabalho de Curso. | | | | | | | | | | |
| FEV | MAR | ABR | MAI | JUN | JUL | AGO | SET | OUT | NOV | DEZ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 01 | Atividade | X |  |  |  |  |  |  |  |  |  |  |
| 02 | Atividade | X | X |  |  |  |  |  |  |  |  |  |
| 03 | Atividade |  | X | X |  |  |  |  |  |  |  |  |
| 04 | Atividade |  | X | X | X |  |  |  |  |  |  |  |
| 05 | Atividade |  |  |  | X | X |  |  |  |  |  |  |
| 06 | Atividade |  |  |  |  | X |  |  |  |  |  |  |
| 07 | Atividade |  |  |  |  |  |  | X | X |  |  |  |
| 08 | Atividade |  |  |  |  |  |  | X | X |  |  |  |
| 09 | Atividade |  |  |  |  |  |  |  | X | X |  |  |
| 10 | Atividade |  |  |  |  |  |  |  |  | X |  |  |
| 11 | Atividade |  |  |  |  |  |  |  |  | X |  |  |
| 12 | Atividade |  |  |  |  |  |  |  |  | X |  |  |
| 13 | Atividade |  |  |  |  |  |  |  |  | X | X |  |
| 14 | Atividade |  |  |  |  |  |  |  |  |  | X |  |
| 15 | Atividade |  |  |  |  |  |  |  |  |  | X | X |

## Apêndice B - Banner [ou Artigo] apresentado na Semana de Comunicação Científica

# ANEXOS

## Anexo A