5º Seminário Internacional do Mês da Pesquisa 2020
Volume 2021
http://dx.doi.org/10.11594/nstp.2021.0908

NST PROCEEDINGS

OPEN ACCESS

Conferência

Análise comparativa da tecnologia Rest e GraphQL na API baseada em Nodejs Desenvolvimento

Gede Susrama Mas Diyasa1*, Gideon Setya Budiwitjaksono2, Hafidz Amarul Ma'rufi1, Ilham Ade Widya Sampurno1

1Departamento de Informática, Faculdade de Ciências da Computação, Universitas Pembangunan Nasional "Veteran" Jawa Timur, Indonésia
2Departamentos de Contabilidade, Faculdade de Economia e Negócios, Universitas Pembangunan Nasional "Veterano" Jawa Timur, Indonésia

Autor para correspondência:
E-mail:
igsusrama.if@upnjatim.ac.id

RESUMO

O serviço da Web é um método de conectar servidores e aplicativos clientes. Existem vários tipos de tecnologia no desenvolvimento de um serviço web, como REST e Graph-QL. Graph-QL é uma tecnologia alternativa criada pelo Facebook para corrigir as deficiências da tecnologia REST, especialmente na seção de apresentação de dados. O Graph-QL fornece uma alternativa onde o aplicativo cliente pode determinar para eles quais dados são necessários. Este artigo analisa o desempenho das duas tecnologias para determinar qual tecnologia é adequada para suas necessidades. A análise realizada é comparar a velocidade de resposta e a eficiência dos dados para otimizar a largura de banda disponível. O modelo de desenvolvimento usa o modelo cascata, que consiste em pesquisa, design, implementação e teste. Como objeto de teste, dois aplicativos baseados em Node-JS foram desenvolvidos com o Express Framework, que aplicou os conceitos REST e Graph-QL em cada objeto de teste. Os resultados obtidos são que o REST tem melhor desempenho que o Graph-QL em sua velocidade de resposta. Por outro lado, o Graph-QL também se destaca na apresentação de dados por solicitações de aplicativos clientes para otimizar a largura de banda disponível.

Palavras-chave: REST, Graph-QL, API, Node-JS, Web Service

Introdução

Um aplicativo tem o termo back-end, que é uma fonte de dados e uma fonte lógica do aplicativo. A fonte de dados de back-end do aplicativo é acessada pelo aplicativo cliente usando uma API (Application Programming Interfaces) (Meng et al., 2018). A API é usada para distribuir os aplicativos cliente e de back-end entre si; em outras palavras, um back-end pode ser acessado por vários aplicativos clientes diferentes.

Na implementação da API, diversas tecnologias podem ser utilizadas, como REST (Representational State Transfer) (Ghebremicael, 2017) ou GraphQL (Brito et al., 2019). Ambas as tecnologias apresentam os mesmos dados em JSON (JavaScript Object Notation) (Lv et al., 2018). GraphQL é uma tecnologia alternativa ao REST introduzida pelo Facebook. O GraphQL (Brito e Valente, 2020) foi criado para melhorar as deficiências do REST em termos de apresentação de dados, onde os dados retornados pelo servidor podem ser ajustados de acordo com as necessidades do cliente (ÿechák, 2017).

Vários estudos relacionados à implementação da tecnologia REST e GraphQL incluem referências [7] [8] [9] [10], em relação (Masdiyasa et al., 2020) Graph-QL testado foi sobre a criação de um sistema integrado de competência teste de certificação, que é baseado na web e comparado com a eficiência do uso da API REST. Os resultados dos testes constataram que o Graph-QL tem vantagens na apresentação de dados no servidor, e a API REST tem melhor desempenho na velocidade de resposta, mas não aborda o NODEJS. Referência (Sayan & Shreyasi, 2020) compara Graph-QL com serviços repousantes em

Como citar:

Diyasa, IS M et al. (2021). Análise comparativa das tecnologias rest e graphQL no desenvolvimento de APIs baseadas em nodejs. *5º Seminário Internacional do Mês de Pesquisa 2020.* NST Proceedings. páginas 1-9. doi: 10.11594/ nstp.2021.0908

gerenciamento de API arquitetônica e explica em detalhes os métodos de gerenciamento e design de arquitetura de APIs GraphQL e REST e analisando os benefícios potenciais do GraphQL em comparação com REST em design de arquitetura de API sem estado.

Na referência (Brito & Valente, 2020), os serviços nas arquiteturas REST e GraphQL fornecem desempenho semelhante à pesquisa de referência (Seabra et al., 2019), onde o valor entre os serviços REST e GraphQL sem discutir o NODEJS. Em contraste, a referência (Hartig & Pérez, 2017) discute o Espaço Logarítmico Não Determinístico (NL) analisando o Graph-QL Facebook para fornecer um novo tipo de interface de acesso a dados na Web.

Este artigo analisará a comparação entre a tecnologia REST e GraphQL no tratamento de uma solicitação de um aplicativo de sistema de teste de competência on-line integrado. Algumas das coisas analisadas são a velocidade de resposta e a eficiência dos dados retornados pelo servidor (Khan & Mian, 2020).
Em sua implementação, serão criadas duas aplicações com diferentes tecnologias de API e acessando o mesmo banco de dados. Aplicativos construídos em execução no NodeJS como recurso de objeto (Tran, 2019). Espera-se que este documento possa ajudar a determinar a tecnologia apropriada a ser usada para fazer uma aplicação.

Revisão da literatura

API (interface de programação de aplicativos)

API significa Application Programming Interface e permite que os desenvolvedores integrem duas partes de um aplicativo ou com diferentes aplicativos simultaneamente (Hou et al., 2017). A API consiste em vários elementos, como funções, protocolos e outras ferramentas que permitem aos desenvolvedores criar aplicativos. O objetivo de usar a API é acelerar o processo de desenvolvimento, fornecendo funções separadas para que os desenvolvedores não precisem construir recursos semelhantes (Mattia et al., 2019). será significativamente sentida Implementação da API se a qualidade desejada for muito complicada. Claro, leva tempo para criar algo semelhante a ele. Por exemplo, integração com um gateway de pagamento. Vários tipos de APIs de sistema podem ser usados, incluindo sistemas operacionais, bibliotecas e a Web (Cutler & Dickenson, 2020).

REST

REST é um modelo de arquitetura que conecta dois aplicativos cliente e servidor baseados na Web usando HTTP (HyperText Transfer Protocol) é um protocolo padrão (Halili & Ramadani, 2018). Assim, o servidor pode ser acessado por diversas aplicações clientes mesmo em diferentes plataformas como mobile, desktop e website. Aplicativos de sites que seguem a arquitetura REST são chamados de serviços da Web REST ful. Alguns dos métodos HTTP frequentemente usados são GET para recuperar dados, POST para adicionar dados, PUT para editar dados e DELETE para excluir dados (Melnichuk et al., 2018). Para lidar com a resposta, um servidor com arquitetura REST geralmente envia códigos HTTP para indicar o status da solicitação.
Os códigos comumente usados incluem 200 (OK), 400 (Solicitação inválida), 401 (Não autorizado), 403 (Forbid den), 404 (Não encontrado), 429 (Excesso de solicitações) e 500 (Erro interno do servidor) (Dell, 2018) .

GraphQL

GraphQL é uma linguagem de consulta para implementações de API baseadas em sites. Ao contrário do REST, o GraphQL tem a vantagem de apresentar os dados dinamicamente de acordo com as aplicações do cliente (Nogatz & Seipel, 2017). O GraphQL fornece um endpoint para acessar todas as fontes de dados no servidor. Para descrever o GraphQL comparado ao REST, podemos usar um exemplo de dados do usuário com doze atributos. No REST, serão retornadas todas as funcionalidades existentes para a aplicação cliente. Enquanto isso, no GraphQL, o aplicativo cliente pode determinar quais dados são necessários para que os dados produzidos sejam muito precisos, dependendo do aplicativo cliente (Ritsilä, 2017). Alguns termos básicos no GraphQL são tipos que definem dados, consultas são um modelo de transação do servidor para o cliente usado para recuperação de dados e mutações são modelos de transação que permitem alterar dados no banco de dados (Mukhiya et al., 2019)

JSON

JSON é um formato de dados independente em JavaScript com Padrão ECMA-262 3ª Edição - Dezembro de 1999 (Bray, 2014). O formato de dados JSON é amplamente utilizado na troca de dados em um serviço da web. JSON é usado porque é um formato universal para várias plataformas. Em outras palavras, esses dados para mat podem ser lidos por qualquer meio, por isso é muito adequado para uso na troca de dados, especialmente entre servidor e aplicativos cliente.

NodeJS

NodeJS é um software usado para construir aplicativos baseados em sites (Bangare et al., 2016). O NodeJS é executado no lado do servidor (Rimal, 2019). O NodeJS foi introduzido por Rayn Dahl em 2009 para fornecer um modelo de entrada/saída mais leve e eficiente. O NodeJS é escrito em C, C++ e JavaScript construído no mecanismo V8 do Google Chrome (Brown, 2016).

Métodos

O fluxo de desenvolvimento utiliza o método Waterfall, um dos ciclos de vida clássicos no desenvolvimento de software. Este método descreve uma abordagem razoavelmente sistemática e sequencial para o desenvolvimento de software, começando por: especificação dos requisitos do usuário, planejamento, modelagem, construção, submissão de sistemas aos usuários e manutenção do sistema.

No método cascata. Se a etapa 1 não foi concluída (Bassil, 2012), a etapa 2 não pode ser executada e assim por diante. Todos os conjuntos estão inter-relacionados e cada um deve ser feito em detalhes e documentado. O método cascata exige que todas as especificações, requisitos e objetivos do sistema sejam definidos no estágio inicial (requisitos e projeto) antes de entrar no processo (implementação).
Isso porque o método cascata não acomoda mudanças no meio do processo de desenvolvimento (Dima & Maassen, 2018). Então, o que o analista e a equipe do cliente concordaram nas etapas iniciais será o resultado final. Tudo deve estar de acordo e ser consistente com ele até que o pedido seja concluído e enviado ao cliente. O próprio cliente não pode intervir com os programadores. Isso é diferente de alguns outros modelos de desenvolvimento que permitem que os dois se comuniquem para determinar ou revisar o trabalho no estágio de codificação. O método cascata é geralmente utilizado em grandes projetos de criação de sistemas com alta complexidade e que demandam muitos recursos humanos em seu desenvolvimento (Yu, 2018). As etapas do método cascata podem ser vistas na Figura 1.

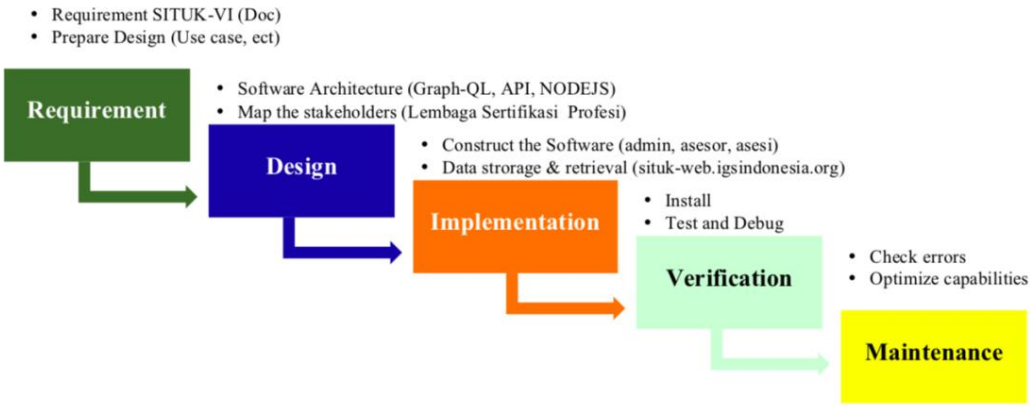


Figura 1. O fluxo de desenvolvimento do sistema com o método cascata

Requisito de análise

Nesta fase, vários requisitos do sistema são analisados (Casteren, 2017). Necessidade é o processo de investigação ou coleta de dados relativos ao sistema a ser criado. Essa coleta de dados pode ser feita por entrevista, estudo da literatura, observação ou pesquisa direta. Nesta fase, o analista

A equipe irá desenterrar o máximo de informações possível do cliente ou usuário sobre qual software eles desejam e outros requisitos do sistema. Os resultados desta etapa produzirão um documento denominado "Documento de requisitos do usuário" ou "Especificação dos requisitos do usuário". O estágio de análise de requisitos tem vários outros termos, incluindo requisitos do sistema, coleta de requisitos do cliente, análise ou análise das necessidades do usuário (Buchori et al., 2017).

A análise realizada nesta etapa analisa a tecnologia a ser utilizada buscando algumas fontes literárias, principalmente sobre GraphQL e REST. Nesta etapa também é realizada a análise das ferramentas, bibliotecas e frameworks utilizados para dar suporte às necessidades de desenvolvimento do sistema. Como objetos de teste, serão criados ¸podem ser vistos com diferentes tecnologias de API, nomeadamente GraphQL e REST. A aplicação será construída com base em NodeJS com o Express Framework e utilizará o MongoDB como banco de dados que roda no servidor local.

Etapas de design e implementação Este

processo de design se concentrará na construção de estruturas de dados, arquitetura de software, design de interfaces, design de funções internas e externas e os detalhes de cada algoritmo processual (Eason, 2016). A fase de projeto produzirá um documento chamado "Requisitos de Software", que se tornará a base para os programadores na criação de códigos de aplicativos.

Já o processo de implementação é a etapa de confecção de aplicações por programadores utilizando códigos específicos de linguagens de programação. O processo de codificação do aplicativo refere-se aos documentos previamente criados. Neste documento, geralmente há uma solução para os módulos do sistema. Vários programadores podem fazer o trabalho do aplicativo ao mesmo tempo sem perturbar o outro sistema como um todo. A etapa de implementação também é chamada de etapa de código e depuração, chamada de etapa de integração e teste do sistema (Salim et al., 2014).

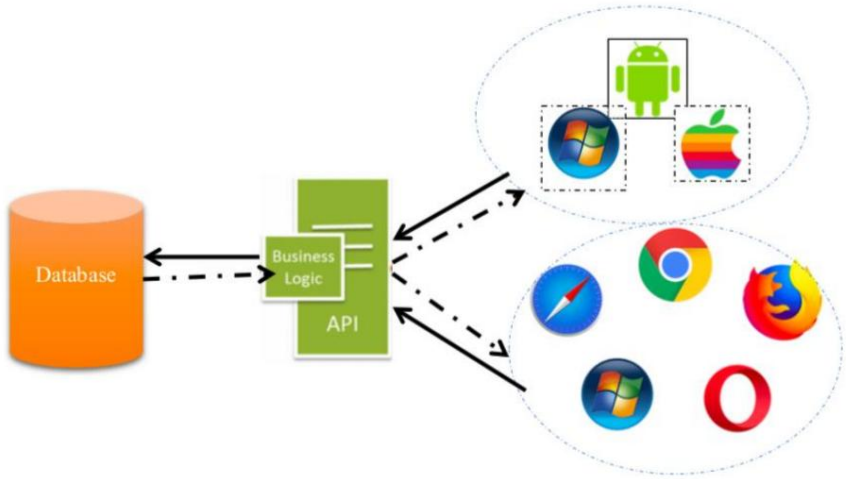


Figura 2. Fluxo do processo da API

As APIs que funcionam no nível do sistema operacional ajudam os aplicativos a se comunicarem com a camada base e entre si seguindo uma série de protocolos e especificações. Um exemplo que pode ilustrar essa especificação é o POSIX (Portable Operating System Interface). Usando o padrão POSIX, os aplicativos compilados para rodar em um determinado sistema operacional também podem rodar em outros sistemas com os mesmos critérios. A biblioteca de software também desempenha um papel essencial na criação de compatibilidade entre diferentes sistemas (Walli, 1995).

Os aplicativos que interagem com a biblioteca devem seguir um conjunto de regras definidas pela API. Essa abordagem facilita para os desenvolvedores de software criar aplicativos que se comuniquem com várias bibliotecas sem repensar as estratégias usadas, desde que todas as bibliotecas sigam a mesma API.

Outra vantagem desse método é a facilidade de usar a mesma biblioteca em diferentes linguagens de programação. Como o nome sugere, a API da Web é acessada por meio do protocolo HTTP. Este é um conceito, não uma tecnologia. E as APIs da Web podem ser feitas usando diferentes tecnologias, como PHP, Java, .NET, etc. Por exemplo, a API Rest do Twitter fornece acesso de leitura e gravação aos dados integrando o Twitter em nossos aplicativos.

Neste estudo, o fluxo começa a partir de uma solicitação de entrada para o API Gateway, que está no endpoint "/api", conforme mostrado na Figura 2. Nesse endpoint, usaremos GraphQL e REST com alguma lógica de recuperação de dados para o banco de dados . O banco de dados utilizado contém uma tabela chamada "todos" com uma estrutura, conforme a Figura 3.

Várias APIs serão feitas em REST como candidatas, conforme mostrado na Tabela 1. Enquanto o tipo GraphQL A API fará diversas consultas e mutações como candidatas, conforme a Tabela 2.

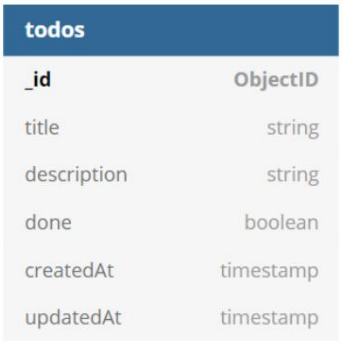


Figura 3. Estrutura da tabela "todos".

Tabela 1. API candidata em REST

Método	Ponto final	Descrição
OBTER	/api/todos /api/	Buscar todos os dados de tarefas
OBTER	todos/:_id /api/todos /	Recupera dados de tarefas com base em _id
PUBLICAR	api/todos/:_id /api/	Criar uma nova tarefa
COLOCAR	todos/:_id	Alterar dados de tarefas com base em _id
EXCLUIR		Excluir dados de tarefas com base em _id

Tabela 2. Consultas candidatas na consulta Graph-

Tipo	QL	Descrição
Consulta	todos	Buscar todos os dados de tarefas
Consulta	todo(_id: ID)	Buscar dados para fazer com base no id
Mutação criada (dados: TodoData)		Criar uma nova tarefa
Mutação updateTodo(_id: ID,data: TodoData)		Alterar dados de tarefas com base no id
Mutação excluída (_id: ID)		Excluir dados de tarefas com base no id

Etapas de Teste (Verificação) e Manutenção A etapa de teste (verificação) inclui a integração do sistema e também o teste das aplicações realizadas. O sistema será verificado para ser testado até que ponto é viável. Nesta etapa, todos os módulos que são trabalhados por diferentes programadores serão combinados e então testados se estão de acordo com as especificações especificadas ou erros/erros no sistema antes de serem corrigidos novamente

O estágio de manutenção geralmente inclui as etapas de instalação de software e teste de aplicativos. A manutenção também é uma forma de responsabilidade da equipe de desenvolvimento para garantir que o aplicativo possa funcionar sem problemas após ser entregue ao cliente por um determinado período. Em uma definição mais ampla, a manutenção é o processo de corrigir um aplicativo de quaisquer erros ou falhas de segurança, melhorando o desempenho do aplicativo, garantindo que o aplicativo possa ser executado em um novo escopo e adicionando mais módulos de desenvolvimento de aplicativos

Neste estudo, o teste foi realizado utilizando o aplicativo Postman solicitando uma API. Neste teste, o tempo de resposta ou a velocidade de uma API é calculado ao lidar com uma solicitação. Neste teste, houve 100 iterações de uma API.

Teste de **resultados e**

discussão A resposta resulta

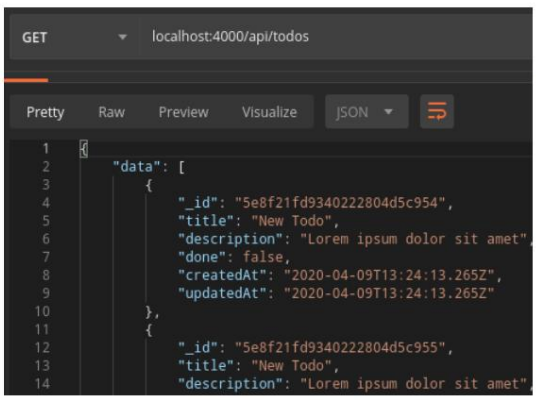
da API baseada em REST na Figura 4.a. Os dados de resposta na Figura 4.a. é obtido solicitando o endpoint "/api/todos" usando o método GET. Nesse caso, a API retorna 1.000 dados com um tamanho de resposta de 183,82 KB. Enquanto isso, os resultados da resposta da API baseada em GraphQL são apresentados na Figura 4.b.

Os dados de resposta na Figura 5 são obtidos solicitando a API baseada em GraphQL enviando o corpo da solicitação no atributo de consulta como a solicitação de dados solicitados para o GraphQL. Nesse caso, a API retorna 1.000 dados com um tamanho de resposta de 183,83 KB. A estrutura do corpo da solicitação ordenada é mostrada na Figura 4.c.

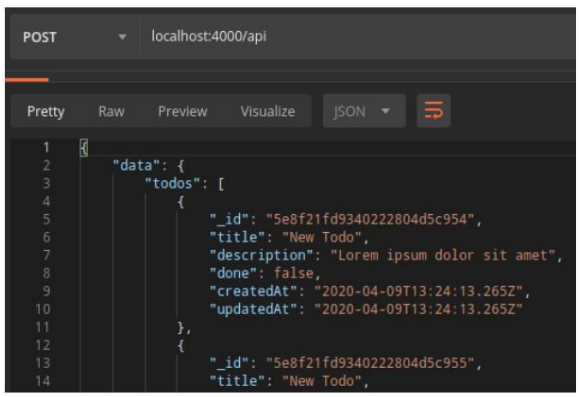
O corpo da solicitação GraphQL é feito enviando os dados e atributos do tipo de API necessários. Neste caso, a API solicitada é "todos" com o tipo "query" e os atributos necessários incluem "_id", "ti tle", "description", "done", "createdAt" e "updatedAt".

Por outro lado, uma API construída usando GraphQL pode tratar requisições com atributos que são definidos de forma customizada pela aplicação cliente com resultados, conforme mostrado na Figura 4.d abaixo. Ao contrário do lance anterior, nos atributos de negócios, o aplicativo cliente pode determinar quais recursos são necessários. O tamanho da resposta retornado pela API é 75,43 KB. A estrutura do corpo da requisição é solicitada, conforme Figura 4.e.

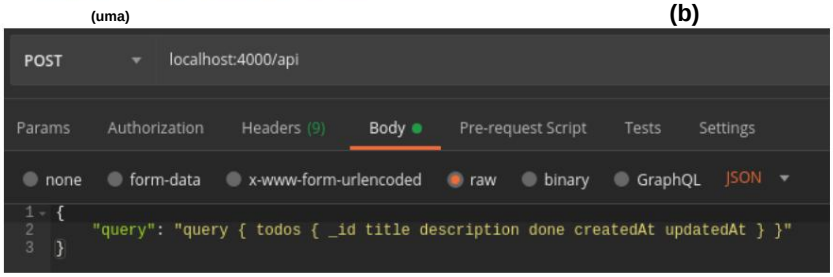
Nesse caso, o aplicativo cliente solicita apenas três atributos customizados para a API, que inclui os atributos "título", "descrição" e "concluído".



(uma)



(b)



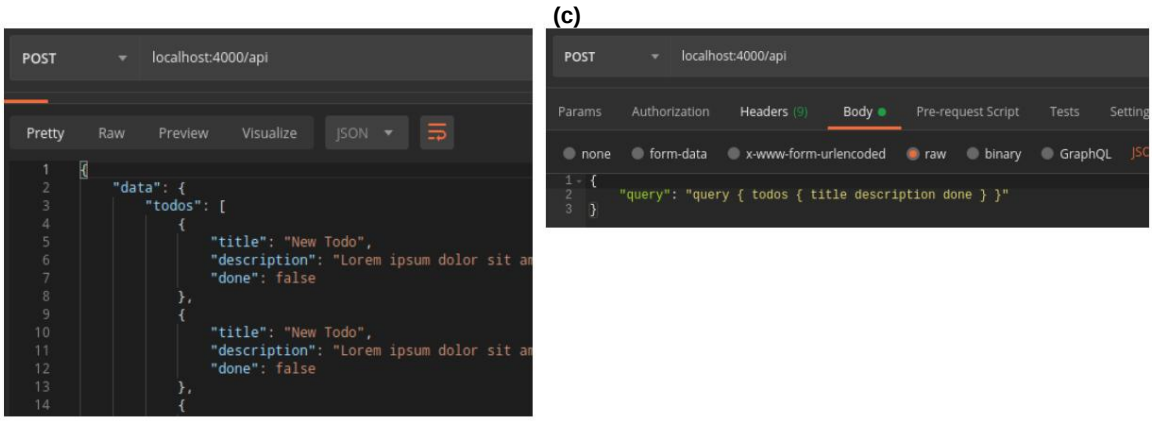


Figura 4. (a) Dados de resposta da API baseada em REST. (b) Dados de resposta da API baseada em GraphQL. (c) Corpo da solicitação na API baseada em GraphQL, (d) Dados de resposta da API baseada em GraphQL com atributos personalizados, (e) Corpo da missão FRe na API baseada em GraphQL com atributos personalizados

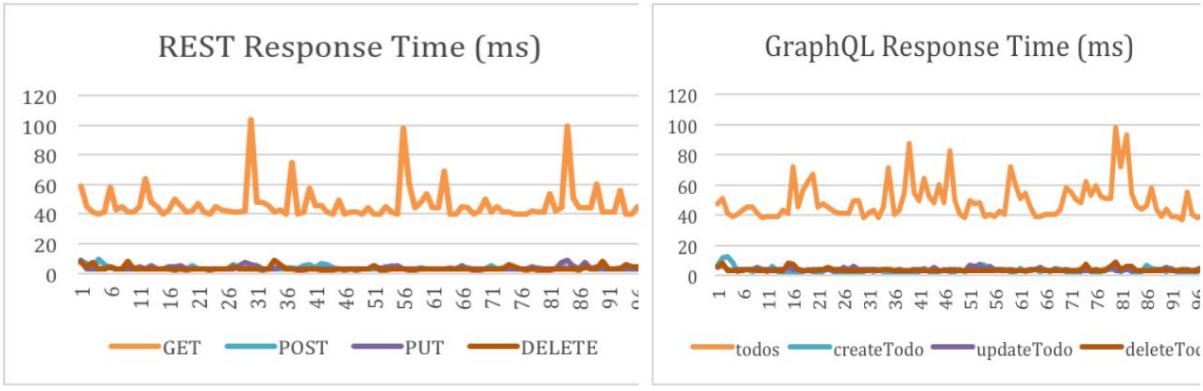
Resultados

Ao fazer 100 solicitações repetidas usando o aplicativo Postman, os resultados da velocidade de resposta na API baseada em REST são mostrados na Figura 5.a. Na API baseada em REST, a velocidade média para cada tipo de API é obtida com 1000 dados no banco de dados da seguinte forma:

1. GET: 46,5 ms 2.
POST: 3,67 ms
3. PUT: 3,51 ms 4.
DELETE: 3,25 ms O

mesmo método de teste é executado na API baseada em GraphQL com os resultados do teste na Figura 5.b. Na API baseada em GraphQL, a velocidade média de resposta para cada tipo de API é obtida ao tratar uma requisição com 1000 dados no banco de dados da seguinte forma: 1. todos: 49 ms 2. createTodo: 3,88 ms 3. updateTodo: 3,81 ms 4 . delete Todo: 3,6 ms

Para que se possa comparar a velocidade entre os dois, que é apresentada na Tabela 3., e na Fig. ure 6 é mostrado em forma de gráfico.



(uma) (b)

Figura 5. (a) Velocidade de resposta na API baseada em REST. (b) Velocidade de resposta na API baseada em GraphQL.

Tabela 3. Comparação das taxas médias de resposta

Tipo	Média (ms)	
	DESCANSO	Gráfico-QL
Ler	46,5	49
Crio	3,67	3,88
Atualizar	3.51	3.81
Excluir	3.25	3.6

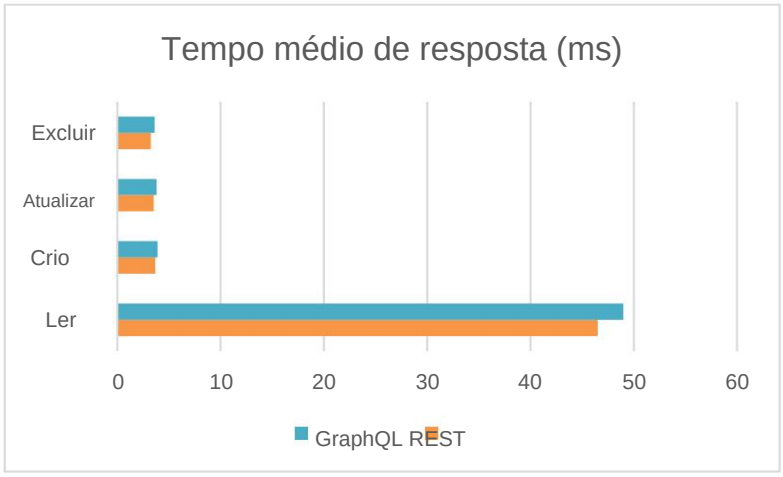


Figura 6. Gráfico da velocidade média de resposta

Os dados médios obtidos entre REST e GraphQL em termos de velocidade de resposta a uma requisição podem dizer que o REST tem um desempenho melhor que o GraphQL, embora com uma ligeira diferença de desempenho.

Conclusão

A cepa *S. prasinopilosus* Pn-TN2 que isolou da amostra de ninho de cupim, mostrou um amplo intervalo de 1. Os dados de velocidade de resposta média na API baseada em REST e GraphQL baseado em NodeJS com o Express Framework podem concluir que REST tem melhor desempenho do que GraphQL em termos de velocidade de resposta no servidor ao lidar com uma solicitação.

2. O GraphQL é superior na apresentação de dados para aplicativos cliente, onde os aplicativos cliente podem definir seus atributos personalizados para que nenhum dado seja inútil. Isso torna o uso da largura de banda mais otimizado

Reconhecimento

Obrigado ao LPPM UPN Veteran East Java, pela pesquisa que nos foi dada, para que possamos publicar artigos em vários periódicos

Referências

Bangare, S, Gupta, S., Dalal, M., & Inamdar, A. (2016). Usando Node.Js para construir um servidor de banco de dados back-end escalonável e de alta velocidade. *interno National Journal of Research in Advent Technology,* 4, 19.

Bassil, Y. (2012). Um modelo de simulação para o ciclo de vida de desenvolvimento de software em cascata. *Jornal Internacional de Engenharia e Tecnologia ogy (iJET),* 2(5), 1–7.

Bray, T. (2014). O formato de intercâmbio de dados JavaScript Object Notation (JSON). *Internet Engineering Task Force (IETF), ECMA International, 1ª edição (outubro),* p. 8. Disponível em: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA 404.pdf%5Cn https://www.rfc-editor.org/info/rfc7158.

Brito, G., & Valente, MT (2020). *REST vs GraphQL: Um experimento controlado', Proceedings - IEEE 17th International Conference on Soft ware Architecture, ICSA 2020, (fevereiro), 81–91.* doi: 10.1109/ICSA47634.2020.00016.

Brito, G., Mombach, T., & Valente, MT (2019). Migrando para o GraphQL: uma avaliação prática. *SANER 2019 - Anais do 2019 IEEE 26ª Conferência Internacional sobre Análise, Evolução e Reengenharia de Software,* pp. 140–150. doi: 10.1109/SANER.2019.8667986.

Brown, E, (2016). Desenvolvimento Web com Node e Express. *Journal of Chemical Information and Modeling, 53* (9), 1689-1699

Buchori, A., Setyosari, P., Dasna, IW, & Ulfa, S. (2017). Design de mídia de realidade aumentada móvel com modelo em cascata para aprender ge ometria na faculdade. *Jornal Internacional de Pesquisa em Engenharia Aplicada,* 12(13), 3773–3780.

Casteren, WV (2017). O modelo cascata e as metodologias ágeis: uma comparação por características de projeto - curta a cascata modelos e metodologias ágeis. *Competências Acadêmicas na Licenciatura, (Fevereiro),* 10–13. doi: 10.13140/RG.2.2.36825.72805. ÿechák, D. (2017). *Usando GraphQL para entrega de conteúdo em Kentico Cloud', Is.Muni.Cz.* Disponível em: https://is.muni.cz/th/qm0cs/thesis.pdf.

Cutler, J., & Dickenson, M. (2020). *Interfaces de programação de aplicativos,* 87–97. doi: 10.1007/978-3-030-36826-5_7.

Dell, Em. (2018). *Unisphere ® Management REST API Programmer* Dima, AM, & *s Guide', em Dell EMC Unity TM Family Versão 4.3,* 1–122.

Maassen, MA (2018). Da cascata ao software ágil: modelos de desenvolvimento no setor de TI, 2006 a 2018. impactos na gestão da companhia. *Jornal de Estudos Internacionais,* 11(2), 315–326. doi: 10.14254/2071-8330.2018/11-2/21.

Eason, OK (2016). *Transições de metodologias de desenvolvimento de sistemas de informação: uma análise da metodologia cascata para ágil.* Universidade de New Hampshire, pp. 1–23. Disponível em: http://scholars.unh.edu/honors%0Ahttp://scholars.unh.edu/honors.

Ghebremicael, ES (2017). *Transformação da API REST para GraphQL para OpenTOSCA.* doi: 10.18419/opus-9352.

Halili, F. & Ramadani, E. (2018). Serviços da Web: uma comparação de serviços de sabão e descanso. *Ciência aplicada moderna,* 12(3), 175. doi: 10.5539/mas.v12n3p175.

Hartig, O., & Pérez, J. (2017). Uma análise inicial da linguagem GraphQL do Facebook. *CEUR Workshop Proceedings, 1912 (maio).*

Hou, L., Zhao, S., Li, X., Chatzimisios, P., & Zheng, K. (2017). Projeto e implementação de interface de programação de aplicativos para Inter rede de nuvens de coisas. *Jornal Internacional de Gerenciamento de Rede,* 27(3), 1-10. doi: 10.1002/nem.1936.

Khan, R., & Mian, AN (2020). Desenvolvimento de aplicações de sensoriamento IoT sustentável através da camada de abstração baseada em graphQL. *Eletrônicos (Suíça),* 9(4), 1–23. doi: 10.3390/electronics9040564.

Lv, T., Yan, P., & He, W. (2018). Pesquisa sobre modelagem de dados JSON. *Journal of Physics: Conference Series,* 1069(1), 1-10. doi: 10.1088/1742-6596/1069/1/012101.

Masdiyasa, IGS, Budiwitjaksono, GS, Amarul, MH, Sampurno, IAW e Mandenni, NMIM (2020). 'Graph-QL Responsabilidade Análise na Base do Sistema de Teste de Certificação Integrada de Competências na Web. *Lontar Komputer,* 11(2), 114–123. doi: https://doi.org/10.24843/LKJITI.2020.v11.i02.p05.

Mattia, S., Lorenzino, V., Dimitrios, M., Robin, S., Monica, PS, & Dietmar, G. (2019). *Interfaces de programação de aplicativos da Web (APIs): Normas de uso geral, termos e iniciativas da Comissão Europeia.* doi: 10.2760/675.

Melnichuk, M., Kornienko, Y., & Boytsova, O. (2018). Serviço de internet. Arquitetura Repousante. *Automação de profissionais tecnológicos e de negócios processos,* 10(1), 17–22. doi: 10.15673/atbp.v10i1.876.

Meng, M., Steinhardt, S., & Schubert, A. (2018). Documentação da interface de programação de aplicativos: O que os desenvolvedores de software desejam?. *Journal of Technical Writing and Communication,* 48(3), 295–330. doi: 10.1177/0047281617721853.

Mukhiya, SK, Rabino, F., Pun, VKI, Rutle, A., & Lamo, Y. (2019). Uma abordagem graphQL para troca de informações de saúde com HL7 FHIR. *Procedia Computer Science, 160,* 338–345. doi: 10.1016/j.procs.2019.11.082.

Nogatz, F., & Seipel, D. (2017). Implementando GraphQL como linguagem de consulta para bancos de dados dedutivos em SWI-Prolog usando DCGs, quasi citações e ditados', Electronic Proceedings in Theoretical Computer Science, *EPTCS, 234 (janeiro de 2017),* 42–56. doi: 10.4204/EPTCS.234.4.

Rimal, A. (2019). *Desenvolvendo um aplicativo da Web em NodeJS e MongoDB usando ES6 e além. Bacharel em Engenharia, Informação Tecnologia, Universidade de Ciências Aplicadas.* Disponível em: https://www.theseus.fi/bitstream/handle/10024/159951/Ri mal_Aashis.pdf?sequence=1&isAllowed=y.

Ritsilä, A. (2017). *GraphQL: A Revolução do Design de APIs.* Programa de Tese de Bacharelado em Bit, University of Applied Science, Haage-helia.

Salim, AP, Chithra, P., & Sreeja, S, (2014). Levantamento sobre Diferentes Modelos de Processos Utilizados no Desenvolvimento de Software. *Jornal Internacional de Ciência da Computação e Tecnologias da Informação,* 5(3), 2855–2860.

Sayan, G., & Shreyasi, M. (2020). Um estudo comparativo entre Graph-QL e serviços Restful no gerenciamento de API do Stateless Architec turas. *Jornal Internacional de Computação de Serviços Web (IJWSC),* 11(02), 1–16. doi: 10.5121/ijwsc.2020.11201.

Seabra, M., Nazário, MF, & Pinto, G. (2019). REST ou GraphQL? Um estudo comparativo de desempenho. *ACM International Conference Pro Série anterior, (junho),* 123–132. doi: 10.1145/3357141.3357149.

Tran, T. (2019). *Crie um aplicativo GraphQL com Node. js e React', (abril).*

Walli, SR (1995). A família de padrões POSIX. *StandardView,* 3(1), 11–17. doi: 10.1145/210308.210315.

Yu, J. (2018). Processo de Pesquisa em Modelo de Desenvolvimento de Software. *Série de Conferências IOP: Ciência e Engenharia de Materiais,* 394(3). 1-10. doi: 10.1088/1757-899X/394/3/032045.