UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

FRANCISCO FABRÍCIO NOGUEIRA PINHEIRO

UMA ANÁLISE COMPARATIVA DE PERFORMANCE ENTRE REQUISIÇÕES EM APIS REST E APIS GRAPHQL UTILIZANDO JAVASCRIPT

QUIXADÁ

2022

FRANCISCO FABRÍCIO NOGUEIRA PINHEIRO

UMA ANÁLISE COMPARATIVA DE PERFORMANCE ENTRE REQUISIÇÕES EM APIS REST E APIS GRAPHQL UTILIZANDO JAVASCRIPT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Victor Aguiar Evangelista de Farias

QUIXADÁ

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P719a Pinheiro, Francisco Fabrício Nogueira.
Uma análise comparativa de performance entre requisições em apis rest e apis graphql utilizando javascript / Francisco Fabrício Nogueira Pinheiro. – 2022.
44 f.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2022.
Orientação: Prof. Dr. Victor Aguiar Evangelista de Farias.

1. testes. 2. performance. 3. requisições. 4. javascript (linguagem de programação de computador). 5. framework (arquivo de computador). I. Título.

CDD 005.1

FRANCISCO FABRÍCIO NOGUEIRA PINHEIRO

UMA ANÁLISE COMPARATIVA DE PERFORMANCE ENTRE REQUISIÇÕES EM APIS REST E APIS GRAPHQL UTILIZANDO JAVASCRIPT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: ____/ ____/ ____.

BANCA EXAMINADORA

Prof. Dr. Victor Aguiar Evangelista de Farias (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Bruno Góis Mateus
Universidade Federal do Ceará (UFC)

Prof. Dr. Jefferson de Carvalho Silva
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

À minha mãe, Lizônia Maria Nogueira e meu pai, Antônio Renier Morais Pinheiro, por todo o esforço e trabalho que tiveram para me ajudar nessa jornada, batalhando para que nunca me faltasse nada, fazendo até o que estava além deles para que um dia eu me formasse, e hoje nos formamos juntos, a vitória é nossa.

À Antônia Natália de Lima e Lourdes Glaubia da Hora Tavares, duas pessoas que me acolheram e prestaram todo suporte durante grande parte da minha trajetória acadêmica.

Agradecer a mim mesmo, pela determinação e foco que mantive durante toda a trajetória para dá o melhor de mim e chegar até aqui.

Aos meus professores do ensino fundamental da E.E.F Antônio Honório Pinheiro e do meu ensino médio E.E.M Euclides Pinheiro de Andrade, em especial à Hatynathan Fonseca Felipe, onde tive a oportunidade de tê-lo como professor e ser monitor em suas disciplinas de matemática e física, onde todos os dias ele me incentivava a fazer faculdade e compartilhava muito do seu conhecimento, com sua história de vida me motivou e mostrou que eu conseguiria ir longe para assim conquistar meus objetivos.

À minha família, irmãs, namorada, avós, tios, padrinhos, primos que sempre acreditaram no meu potencial e me deram boas energias para que eu sempre seguisse firme na busca pelo meu sonho.

Aos meus amigos do interior de Milhã-CE, Angicos-RN e da PTF que sempre torceram por mim e me motivaram a acreditar que realizar meu sonho era possível, saiba que carrego vocês sempre em meu coração.

Ao Victor Aguiar Evangelista de Farias, meu orientador, por ter me ajudado e guiado com maestria durante a realização deste presente trabalho.

A Bruno Góis Mateus e Jefferson de Carvalho Silva, minha banca avaliadora, pela disponibilidade em participar da banca desse trabalho, e pelas suas colaborações e sugestões.

A Universidade Federal do Ceará (UFC) e a todos os professores do campus Quixadá, que me auxiliaram nessa caminhada, foi um prazer fazer parte desse campus.

Muito Obrigado!

“Você pode superar qualquer coisa, se e somente se você ama algo suficiente.”

(Lionel Messi)

RESUMO

Com ascensão da tecnologia nos últimos anos foi possível desenvolver diversos sistemas e produtos de software visando aproveitar oportunidades de negócio, conectar e facilitar a comunicação de diversos usuários em uma aplicação de software, como também em automatizar processos, a engenharia de software traz recursos, ferramentas e processos buscando agilizar o tempo de desenvolvimento e visando também auxiliar na qualidade e performance que todo e qualquer software deve apresentar quando estiver disponível para utilização. Portanto, o objetivo desse trabalho é realizar uma análise comparativa de performance de tecnologias de desenvolvimento de software. Para isso, primeiro, pesquisaram-se tecnologias disponíveis no mercado para serem comparadas que atendessem critérios de popularidade, comunidade ativa e quantidade de aparições em artigos e tópicos de pesquisa. Foram identificadas 3 e dado os resultados das pesquisas, foi definido a utilização de 2 tecnologias de uma área específica do desenvolvimento de software, umas delas utilizada no presente no trabalho são os *frameworks* que surgiram com intuito de facilitar o desenvolvimento de aplicações *web*, dessa forma foi selecionado um *framework* para desenvolvimento *back-end*, para serem implementadas aplicações servidoras para realização dos testes de performance. As análises foram realizadas baseando-se em métricas de performance selecionadas na literatura e também disponível na ferramenta de *benchmarking* utilizada. Os testes de carga foram planejados em cima da simulação de cenários reais ao qual ocorre grande quantidade de acessos por parte dos usuários realizando diversas ações em um sistema de software dado um determinado contexto.

Palavras-chave: testes; performance; requisições; javascript (linguagem de programação de computador); framework (arquivo de computador).

ABSTRACT

With the rise of technology in recent years it was possible to develop several systems and software products aiming to take advantage of business opportunities, connect and facilitate the communication of various users in a software application, as well as automate processes, software engineering brings resources, tools and processes seeking to speed up the development time and also aiming to assist in the quality and performance that any software must present when it is available for use. Therefore, the objective of this work is to perform a comparative analysis of the performance of software development technologies. To do this, first, we searched for technologies available in the market to be compared that met the criteria of popularity, active community, and number of appearances in articles and research topics. Three technologies were identified and given the results of the research, it was defined the use of two technologies from a specific area of software development, one of them used in this work are the frameworks that appeared with the intention of facilitating the development of web applications, this way a framework was selected for back-end development, to be implemented server applications for the performance tests. The analyses were performed based on performance metrics selected in the literature and also available in the benchmarking tool used. The load tests were planned based on the simulation of real scenarios in which a large number of accesses are made by users performing several actions on a software system given a certain context.

Keywords: testing; performance; requests; javascript (computer programming language) framework (computer file).

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A crescente ascensão do mercado de tecnologia fez com que uma infinidade de sistemas e aplicativos fossem desenvolvidas com o intuito de atender os mais diferentes setores econômicos. Esse crescimento desenfreado se deu muito pela evolução da *internet* nos últimos anos, que 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 à informação, entre outros (RIBEIRO, 2021).

Muitos empreendedores buscam criar soluções tecnológicas de modo a expandir seus negócios e levar seu produto ou serviço ao maior número de usuários possíveis, assim gerando receita e reconhecimento no mercado. A *internet* quando criada tinha como objetivo inicial facilitar o compartilhamento de informações, porém atualmente além de compartilhar informações entre alguns departamentos nas empresas, também é usada para entretenimento, criar negócios, realizar compras e vendas de produtos e/ou serviços, ou até mesmo para gerá-la (RIBEIRO, 2021).

Ao que se refere a criação de produtos e/ou serviços em sua maioria se dá por meio de *software* com odesenvolvimento de aplicações *web*, onde o usuário consegue trocar dados e informações através da *internet* com auxílio das aplicações desenvolvidas. Existem diversas tecnologias e abordagens para acessos à dados por aplicações *web*, como: REST *(Representational State Transfer*) e GraphQL *(Graph Query Language)* (LOPES, 2019), ambas possuem suas particularidades e especificações na troca de dados e informações, assim como a maneira ao qual é implementando, sendo o principal responsável pelo acesso e tráfego de dados entre *client-side* (Lado do Cliente) e *server-side* (Lado do Servidor) dos sistemas de software. Uma comparação de performance do REST e GraphQL para acesso a dados em aplicações *web* já foi relatada por alguns especialistas, no entanto, os resultados foram em sua maioria de simulações ou execução de desenvolvimento de aplicações *web*, porém os resultados ou foram inconclusivos ou inconsistentes (LAWI, 2021). Em um cenário ao qual o número de acessos de usuários nos mais diversos sistemas e aplicativos possa se multiplicar, a performance se torna um requisito chave para que os serviços *web* operem como esperado com grande quantidade de dados e informações.

As aplicações *web* desenvolvidas também tiveram sua evolução com o tempo para atender a abundante quantidade de usuários, pois a facilidade no acesso à comunicação com a *internet* teve aumentos exponenciais de usuários. Além do crescente número de acessos de usuá-

rios em sistema de software e aplicativos, a gama de *frameworks* e bibliotecas disponíveis para facilitar no desenvolvimento também podem gerar sistemas lentos, tornando assim um ambiente de trabalho impraticável, criando problemas de custos aos desenvolvedores e organizações, visto que o tempo dos funcionários nas empresas, por exemplo, implica na produtividade e pode estar sendo despendido em esperas desnecessárias (COUTO, 2015).

A performance de uma aplicação impacta diretamente na experiência do usuário, em situações de aplicações instáveis, como: sistema de compras fora do ar em datas festivas; sites de eventos indisponíveis para compra de ingressos de shows com alta popularidade; um sistema de *delivery* fora do ar após divulgar uma promoção temporária em rede nacional (LUCAS, 2019). O usuário requer no mínimo uma aplicação estável para realizar suas atividades rotineiras e demais ações, já o dono do produto ou serviço de software desenvolvido espera que sua aplicação *web* atenda aos requisitos dos usuários e tenham um desempenho garantido, satisfazendo assim seu público alvo. É recomendado que os sistemas sejam testados, analisados e corrigidos, antes de sua utilização. Desta forma, a avaliação de performance de um sistema antes de seu uso é de extrema importância (COUTO, 2015).

Conforme o que foi discorrido sobre a relevância que a performance tem em *softwares* para satisfazer e atender as necessidades de seus usuários independente da quantidade de acessos, este trabalho visa realizar uma análise comparativa de performance de requisições em aplicações *web* servidoras, será realizado testes de performance onde possa abordar a simulação de cenários reais de uso por parte dos usuários, que simulem uma determinada ação onde será avaliada as métricas de performance, dado a quantidade de usuários realizando requisições simultaneamente, visando também analisar qual das abordagens melhor atende aos requisitos de desempenho em grandes cargas de acessos em cada um dos cenários simulados, visto que os testes de performance podem oferecer insumos para a tomada de decisão sobre diversos elementos da aplicação e do negócio.

Para realização dos testes de performance serão implementadas aplicações *web* servidoras em REST e GraphQL com operações básicas de CRUD (*CREATE, READ, UPDATE e DELETE*) em NodeJS, um ambiente em tempo de execução de código aberto e multiplataforma que permite aos desenvolvedores criarem todo tipo de aplicativos e ferramentas *server-side* na linguagem de programação JavaScript, utilizando o *framework*: ExpressJS. Com os resultados obtidos será apresentado em tabela o comparativo das métricas de performance das requisições em APIs REST e APIs GraphQL avaliadas a partir da ferramenta em cada um dos cenários

utilizados.

1.1 Objetivos

Nesta Seção, serão descritos o objetivo geral e objetivos específicos do presente trabalho.

1.1.1 Objetivo geral

Realizar uma análise comparativa de performance em requisições a APIs REST e APIs GraphQL, visando simular diferentes cenários reais de uso e analisar qual das tecnologias de desenvolvimento de API supracitadas se mostra a mais performática em cada cenário.

1.1.2 Objetivos Específicos

a) Definir *framework* para desenvolvimento das APIs.
b) Construir API em REST e GraphQL com um *framework* JavaScript de *back-end* para realização dos testes de performance.
c) Comparar cargas de requisições que simulam cenários reais de uso por parte dos usuários para avaliar métricas de performance em APIs REST e APIs GraphQL.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados os conceitos abordados pelo estudo que compõem o embasamento teórico e possibilitam o entendimento deste trabalho.

2.1 API *(Application Programming Interface)*

Interface de Programação de Aplicações ou Interface de Programação de Aplicação, cuja sigla API provém do Inglês *Application Programming Interface* são "tradutores" com a função de conectar sistemas, softwares e aplicativos (COSTA, 2022). As APIs permitem que o usuário final utilize um aplicativo, software ou até uma simples planilha, consultando, alterando e armazenando dados de diversos sistemas, sem que o usuário precise acessá-los diretamente (COSTA, 2022).

No que se diz a respeito ao presente trabalho a definição de API se limita ao contexto de um conjunto de funcionalidades *back-end* que consulta dados específicos sobre um recurso de armazenamento persistente através de uma ferramenta que simule um serviço acessando dados da API. Para a criação e design de API é utilizado duas tecnologias para seu desenvolvimento: REST e GraphQL, os tipos mais comumente utilizados pelas empresas.

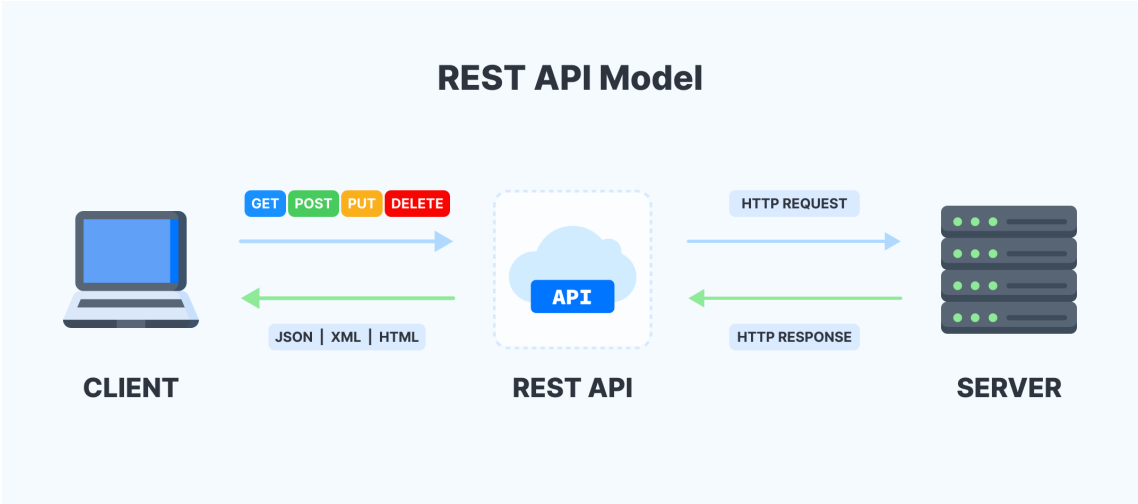
2.1.1 REST

O REST surgiu como uma forma alternativa ao SOAP (*Simple Object Access Protocol*), em que um cliente envia uma requisição ao servidor, o servidor então processa essa requisição e retorna uma resposta (RIBEIRO, 2021). Essas requisições e respostas estão envolvidas diretamente na transferência de representações de dados e recursos, em uma de suas especificações REST define-se como um modelo arquitetural de software que determina um conjunto de restrições a serem usadas para criar serviços *web* e demais aplicações de software.

O termo REST se trata de mostrar uma visão de como um aplicativo da *web* bem projetado se comporta: é uma rede de recursos (máquina de estados virtuais) onde o usuário avança pela aplicação selecionando identificadores de recursos (SEABRA, 2019). Serviços *web* implementados no modelo arquitetural REST representado conforme a Figura 1, permite que os clientes representados por um usuário realizando acesso por notebook, tablet ou smartphone (ou até mesmo outros sistemas realizando requisições) acessem e manipulem representações textuais e outros dados de recursos da *web* usando um conjunto uniforme e predefinido de operações

sem estado. O cliente realiza uma solicitação por meio das operações do REST (GET, POST, PUT e DELETE), a API recebe a informação do que foi solicitado e então processa no servidor apontando o responsável por prover as informações da resposta, em seguida a API recebe a resposta e a envia para o cliente por meio de um formato selecionado, seja JSON, XML, HTML ou outro.

Figura 1 – Serviço *web* em um modelo arquitetural REST



Fonte: (APPMASTER, 2022)

O REST nasceu no ano de 2000, na tese de doutorado de Roy Fielding, na Universidade da Califórnia (BANKS, 2018). Em sua tese, ele descreve o conceito como uma arquitetura orientada a recursos, nos quais os usuários têm a possibilidade de acessar tais recursos através de operações pré-definidas, sendo elas, GET, PUT, POST e DELETE, O presente trabalho se propõe a desenvolver APIs que necessitam transitar dados utilizando as operações disponíveis no REST. Cada uma dessas operações com sua função (SOARES, 2020), são descritas abaixo:

1. **GET**: recupera um recurso.
2. **PUT**: atualiza um recurso existente ou cria um recurso.
3. **POST**: cria um novo recurso.
4. **DELETE**: remove um recurso.

2.1.2 GraphQL

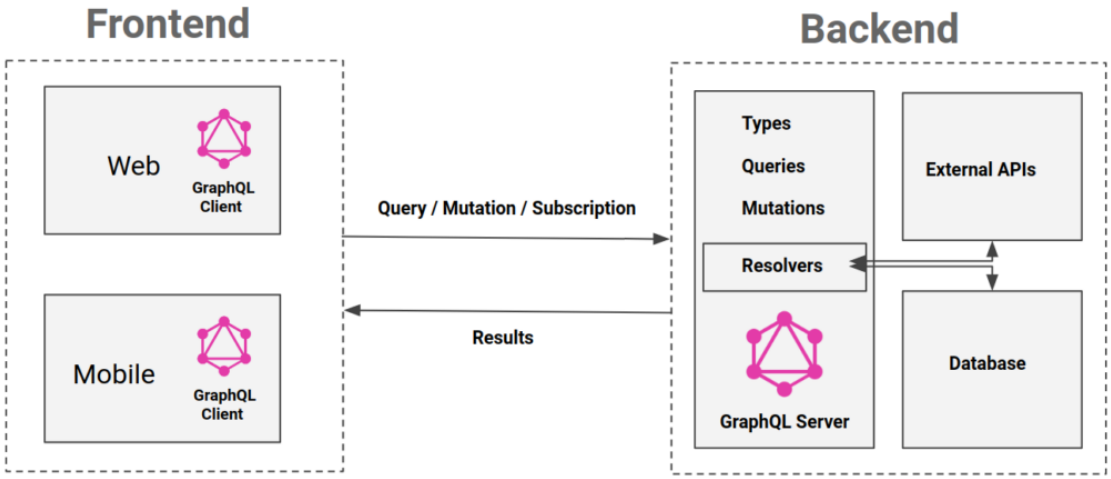
GraphQL [1] é uma linguagem de consulta para APIs e um tempo de execução para atender a essas consultas com seus dados existentes. O GraphQL fornece uma descrição completa e compreensível dos dados em sua API, oferece aos clientes o poder de solicitar exatamente o

[1] https://graphql.org/

que eles precisam e nada mais, facilita a evolução de APIs temporalmente e permite ferramentas de desenvolvedor poderosas.

Na literatura é encontrada também uma definição que complementa a anterior com a seguinte frase: “GraphQL atende a consultas com seus dados em tempo de execução. O serviço GraphQL é independente do meio de transporte, mas em geral é servido sobre o protocoloco de comunicação HTTP (*Hypertext Transfer Protocol*)” (BANKS, 2018). Na Figura 2 é possível observar uma representação da arquitetura de uma API GraphQL na realização de requisições.

Figura 2 – Arquitetura de uma API GraphQL



Fonte: (QUIQUE, 2022)

É possível observar na imagem como as requisições de comportam com o GraphQL, diferenciando-se do REST tanto em termos de estrutura como em suas funções e operações, assim como serão implementadas APIs REST, para ser realizado a análise comparativa de performance proposta pelo presente trabalho, o mesmo será desenvolvido utilizando GraphQL. Para um entendimento mais claro do fluxo em requisições a API GraphQL como representado na Figura 2, é descrito alguns conceitos técnicos e básicos de GraphQL:

1. ***Query***: termo usado de para definir qualquer consulta GraphQL que busque *(pull)* dados no servidor.
2. ***Mutation***: termo usado para definir qualquer consulta GraphQL que envie *(push)* dados para o servidor.
3. ***Schema***: é a definição de todas as *queries*, *mutations*, campos e tipos da API GraphQL.
4. ***Subscriptions***: são uma forma de criar e manter uma conexão em tempo real com o servidor.
5. ***Resolvers***: é uma função responsável por preencher os dados de um único campo em seu

esquema.

2.2 Testes de Performance

O teste de performance visa principalmente demonstrar se numa carga qualquer de informação o desempenho de uma aplicação atende às metas desejadas (MOLINARI, 2009). Com o teste de performance é possível avaliar a eficiência das aplicações e buscar possíveis falhas no sistema, importantes para o processo de construção do mesmo e disponibiliza informações necessárias para que a empresa possa tomar ações corretivas em relação às inconformidades relacionadas ao desempenho dos seus sistemas (COUTO, 2015). Existem hoje na literatura diversos tipos de testes de performance, porém o presente trabalho focará no teste de carga.

Como subtipo do teste de performance o teste de carga se dá pela quantidade de acessos e número máximo de requisições que uma API consegue suportar, se concentrando na capacidade de um sistema em lidar com níveis crescentes de cargas reais, antecipadamente, resultantes de solicitações de transação geradas por quantidades de usuários ou processos controlados, ou concorrentes (LUCAS, 2019). O presente trabalho irá realizar testes de carga com uma ferramenta de *benchmarking* que se trata de técnicas desenvolvida e muito utilizada por indústrias para monitorar a concorrência, comparar o desempenho de resultados e melhorar a eficiência de processos. Em cada uma das APIs desenvolvidas será avaliada métricas de performance e desempenho na realização de requisições.

2.3 Javascript

JavaScript é uma linguagem de *script* usada em milhões de páginas *web* em todo o mundo para validar formulários, detectar objetos e adicionar uma série de outras funcionalidades interativas (SILVA, 2010). Criada pela Netscape [2] em parceria com a Sun Microsystems [3], o JavaScript além de adicionar a interatividade nas páginas *web* em *client-side* (lado do cliente), por exemplo, ela se expandiu e se tornou uma linguagem que agora pode ser executada em *server-side* (lado do servidor) com auxílio do NodeJS que será descrito na seção seguinte.

Além de uma linguagem de programação popular da *web*, a ampla maioria dos sites modernos usa JavaScript e todos os navegadores modernos — em computadores de mesa, consoles de jogos, *tablets* e *smartphones* — incluem interpretadores JavaScript, tornando-a

[2] https://isp.netscape.com/
[3] https://www.oracle.com/br/corporate/

a linguagem de programação mais onipresente da história (FLANAGAN, 2004). Com essa popularidade ao longo dos anos foram sendo criados diversos *frameworks* visando facilitar o desenvolvimento de aplicações.

2.4 NodeJS

O NodeJS também chamado de Node, é um ambiente JavaScript para desenvolvimento de aplicações em *server-side* (TILKOV; VINOSKI, 2010). Foi baseado na implementação do *runtime* da Google, chamado motor "V8", utilizado para interpretação da linguagem JavaScript no *browser* do Google Chrome. O NodeJS e o V8 são implementados nas linguagens de programação C e C++, focando em performance e baixo consumo de memória (RIBEIRO, 2021).

Com a utilização do V8, foi possível para desenvolvedores utilizarem a linguagem de programação JavaScript, inicialmente criada para desenvolvimento do *front-end* em *client-side*, ser utilizada também no desenvolvimento de aplicações no *back-end* em *server-side* (RIBEIRO, 2021). A criação de um ambiente em NodeJS para realizar implantação de uma aplicação, é uma tarefa que não usa tanto recursos computacionais em comparação com outras tecnologias tradicionais se utilizado em conjunto com outras ferramentas que otimizem mais as tarefas manuais do programador, faz com que o mesmo se preocupe mais nas regras de negócio da aplicação que está desenvolvendo, do que mesmo em configurações complexas para ser possível colocar o sistema no ar funcionando e disponível para os usuários, a leveza e flexilidade também é um ponto a se destacar no NodeJS.

2.5 Frameworks

O reúso na engenharia de software é bastante utilizado, com a reutilização de software a qualidade do desenvolvimento aumenta, pois, bibliotecas e *frameworks*, por exemplo, foram construídos com base nisso, em tarefas repetitivas identificadas no desenvolvimento de aplicações, sendo observado a necessidade de otimizar essas tarefas, unindo todo aquele código com suas funcionalidades que eram utilizadas várias vezes em um *framework*, facilitando na implementação de novas aplicações destacando vantagens como redução no esforço de trabalho e o tempo de codificação.

2.5.1 ExpressJS

O primeiro framework utilizado para o desenvolvimento das APIs do presente trabalho foi o ExpressJS [4], *framework* para aplicativo da *web* do NodeJS mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos *web* e móvel, com uma infinidade de métodos utilitários HTTP e *middleware* à sua disposição, criando uma API robusta e rápido facilmente.

Escrito com a linguagem JavaScript, é um *framework* muito popular tanto em grandes empresas quanto nas comunidades de desenvolvimento, o ExpressJS auxilia na criação de aplicações utilizando o NodeJS em conjunto com o JavaScript, tornando este ecossistema e *stack* ainda mais poderosa. Com o ExpressJS o uso do TypeScript é opcional, assim como a estrutura de pastas é de total escolha do desenvolvedor em criar uma do zero, sem módulo ou arquitetura já definida.

[4] https://expressjs.com/

3 TRABALHOS RELACIONADOS

Neste capítulo, são descritos os trabalhos relacionados que possuem relação com a proposta deste trabalho, que abordam análises comparativas entre REST e GraphQL

3.1 GraphQL: Uma alternativa aos *webservices*

No trabalho de (RIBEIRO, 2021) é apresentado o GraphQL como alternativas aos webservices REST e SOAP elucidando pontos que essas tecnologias necessitam de um tempo maior de desenvolvimento caso seja necessário um controle maior na resposta das requisições. É citado alguns exemplos como o de uma de requisição a uma URL (Uniform Resource Locator — Localizador de Recursos Uniforme) e que essa mesma retornasse uma imensa quantidade de dados, dos quais não seriam usados nem metade.

Com isso, ele introduz sobre o GraphQL levantando como hipótese que pode ser uma tecnologia melhor do que as existentes e que pode ser utilizada a qualquer momento, que embora seja nova, por corporações de grande porte como uma alternativa a webservices que utilizem tecnologias REST ou SOAP, salientado também que com o GraphQL nas requisições é capaz 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.

O estudo pretende, por um experimento, validar com métricas quantitativas que GraphQL pode ser uma tecnologia melhor do que as existentes como uma alternativa a webservices que utilizem tecnologias REST ou SOAP em termos de performance e tempo de desenvolvimento. Com isso é definido métricas e realizado o experimento analisando as requisições individualmente, ele pega, por exemplo, a requisição de POST, que está relacionado a criar um registro no banco e com isso ele pega a mesma para as três tecnologias citadas e analisa o resultado apresentado pelas métricas, o autor desenvolve APIs para realizar o experimento.

A relação do presente trabalho com o aqui citado se dá pelo fato que ambos objetivam a realização de uma análise de performance, com diferentes tecnologias de implementação de API, implementando APIs com cada uma delas para a realização dos testes e experimento, no entanto, o presente trabalho não aborda a tecnologia SOAP e os testes realizado são baseados em cenários reais, da utilização por parte de usuários realizando diversas requisições buscando finalizar sua ação e chegar ao final do seu objetivo de uso, com isso não é realizado testes ou

experimento em requisições individualmente, mas sim em um grupo delas presentes na execução de algum cenário real.

3.2 REST or GraphQL? A Performance Comparative Study

Em (SEABRA, 2019) o trabalho visa explorar o comportamento, em termos de desempenho, entre APIS REST e GraphQL, considerando características da arquitetura implementada nas aplicações e o design de cada modelo. Para medir as métricas de performance, a ferramenta AB (APACHE BENCHMARKING) foi utilizada. Esta ferramenta realiza testes de carga ao enviar um número arbitrário de requisições simultâneas para servidores. Devido a não identificação de nenhum sistema de software que tivesse duas versões da mesma aplicação desenvolvida utilizando REST e GraphQL, em repositórios de projetos de software como *GitHub* [1] e *Bitbucket* [2], eles decidiram implementar três aplicações usando ambas tecnologias, as aplicações foram escritas em Node.js utilizando o *framework web* Express.

Os resultados (SEABRA, 2019) foram de grande relevância para a comunidade de desenvolvimento, como principal descoberta, foi observado que serviços *web* migrados para GraphQL apresentaram um aumento no desempenho em dois terços das aplicações testadas, no que se refere ao número médio de requisições por segundo e taxa de transferência de documento por segundo.

Assim como a pesquisa realizada por Seabra, Nazário e Pinto (2019), o presente trabalho analisará a capacidade e comportamento, em termos de performance, diferentes aplicações desenvolvidas com REST e GraphQL, utilizando uma ferramenta de *benchmarking*, no entanto, o trabalho citado é utilizado apenas um *framework* JavaScript de *back-end*, pois o foco é mais entre os modelos arquiteturais e seu *design* para desenvolvimento de aplicações, já no presente trabalho o foco será também nos modelos, porém com diferentes *frameworks* JavaScript, de modo a analisar qual apresenta melhor desempenho em conjunto com as tecnologias de implementação de API REST e GraphQL, servindo até mesmo de extensão do trabalho relacionado.

[1] https://github.com/
[2] https://bitbucket.org/

3.3 Análise do Custo-Benefício de GraphQL em Comparação a REST e SOAP em Aplicações para Dispositivos Móveis

No estudo de (LOPES, 2019) foi realizado uma análise de custo-benefício da utilização do GraphQL como alternativa ao REST e SOAP na integração de *web services* com foco em aplicativos móveis, a ênfase é dada as características associadas ao tempo de design, quando a aplicação está sendo projetada e ao tempo de implementação, quando a aplicação está sendo desenvolvida, junto a métricas associadas ao tempo de execução, quando a aplicação está sendo executada. Dessa forma, é analisado o custo-benefício das três abordagens nesses três contextos, colocando em perspectiva GraphQL, a abordagem mais recente.

A partir dos resultados obtidos, os autores concluíram que a escolha da abordagem depende das prioridades de cada equipe de desenvolvimento, ao se tratar de REST e GraphQL. SOAP apresenta-se como a escolha mais inapropriada em todas as etapas de desenvolvimento. REST apresentou melhor padronização e consistência no tempo de design, além de não precisar da criação manual de arquivos na implementação. GraphQL apresentou flexibilidade na definição dos dados e geração automática de documentação, além de validação automática na implementação.

A relação do trabalho aqui citado com este se dá pelo fato de que ambos realizam uma comparação de desempenho da aplicação desenvolvida com as diferentes tecnologias de construção de API REST e GraphQL, avaliando tempo de resposta de requisições e uso de dados em cada uma das requisições, o que diferencia é que o trabalho citado utilizou um aplicativo móvel para realizar os testes, já o presente trabalho tem como ênfase APIs para aplicações *web* desenvolvidas com tecnologias do JavaScript, realizando os testes de performance com uma ferramenta de *benchmarking*.

3.4 Quadro comparativo

A tabela apresenta uma comparação entre os trabalhos relacionados e o presente trabalho.

Tabela 1 – Tabela comparativa entre o presente trabalho com trabalhos relacionados.

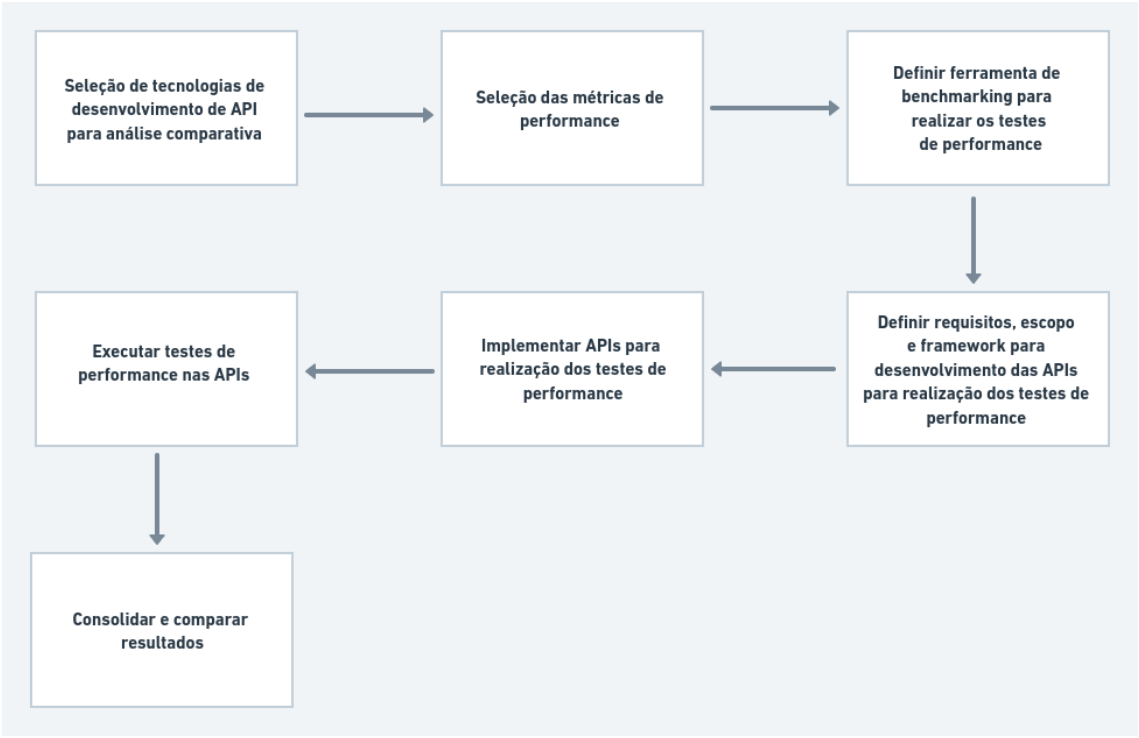
Trabalhos Relacionados	**Presente Trabalho**	**(Ribeiro 2021)**	**(Seabra; Nazário; Pinto et al., 2019)**	**(Lopes; Silva; Ponciano et al., 2019**
Utilização de ferramenta de benchmarking para realização dos testes	Sim	Não	Sim	Sim
Testes de carga em requisições as APIs REST e APIs GraphQL	Sim	Sim	Sim	Sim
Implementação de APIS para realização dos testes	Sim	Sim	Sim	Não
Comparar a performance de requisições que simulem um cenário real de uso	Sim	Não	Não	Não

Fonte: Elaborado pelo autor.

4 PROCEDIMENTOS METODOLÓGICOS

Nesta seção serão apresentadas as etapas necessárias para a realização do presente trabalho. Independente das APIs testadas, as fases para planejamento e aplicação dos testes devem ser a mesma, no entanto, todas as requisições e respostas precisam ser analisadas, a partir disso o presente trabalho procurou basea-se em passos estabelecidos na literatura como "Definição do processo, planejamento de testes e cenários, construção de cenários, executando cenários e analisando cenários" (BANKS, 2018) para o planejamento e execução dos testes. Começando com a definição de tecnologias de implementação de API para o projeto de pesquisa e da ferramenta de *benchmarking* para realizar a análise de performance, em seguida a Seleção de métricas da ferramenta escolhida para análise comparativa de performance, definição de escopo, requisitos e *frameworks* para o desenvolvimento das APIs para a realização dos testes de performance, implementação das APIs para os testes de performance, realização dos testes e consolidar e analisar os resultados. Os procedimentos metodológicos são representados na Figura 1 e descritos a seguir.

Figura 3 – Fluxo de execução dos procedimentos metodológicos



Fonte: Elaborado pelo autor

4.1 Seleção de tecnologias de desenvolvimento de API para análise comparativa de performance

Será realizado uma pesquisa para escolha de tecnologias de desenvolvimento de API para serem analisadas e comparadas. No presente trabalho foi estabelecido que seriam utilizados duas tecnologias de desenvolvimento de API alvos deste estudo. A escolha das tecnologias se deu por alguns fatores listados a seguir:

1. Comunidade ativa.
2. Quantidade de aparições em tópicos de pesquisa e artigos publicados.
3. Popularidade.

4.2 Seleção das métricas de performance

Neste passo será definido as métricas de performance a serem avaliadas nas requisições as APIs. Com auxílio da ferramenta de *benchmarking* para realização dos testes que será definida na seção seguinte, será possível verificar e avaliar as métricas de desempenho e performance que a ferramenta possa prover, assim como os recursos para a aplicação dos testes.

4.3 Definir ferramenta de *benchmarking* para realizar os testes de performance

Após a escolha das tecnologias de implementação de APIs e as métricas de performance alvos do presente estudo, se conduz necessário a definição de uma ferramenta de *benchmarking* que será utilizada para a realização dos testes de performance. A escolha foi realizada com base em pesquisa de trabalhos relacionados e literatura disponibilizadas. Para realização dos testes de carga alvo do presente trabalho foi considerado ferramentas que apresentem os seguintes recursos descritos:

1. **Quantidade de usuários realizando acessos simultâneos:** tempo suficiente para que o usuário pare, pense e execute novamente suas interações com o sistema. Conduz os usuários virtuais se comportarem como se fossem usuários reais. Por meio deste tempo, são injetados atrasos e paradas variáveis nos testes para simular cargas de utilização mais realistas.
2. ***Cache* de navegador:** permite que requisições estáticas como imagens, folhas de estilos, *scripts* entre outros sejam realizadas apenas uma vez por usuário, sabendo que o comportamento padrão dos navegadores de mercado faz *cache* de arquivos que não são alterados

com frequência.

3. **Concorrência:** significa vários usuários utilizando a aplicação e executando diversas funcionalidades no mesmo período, simultaneamente.
4. **Requisições derivadas:** requisições feitas a partir de uma requisição HTTP principal, são elas as requisições para imagens, folhas de estilo e arquivos de *script* JavaScript.
5. **Número de iterações:** recomenda-se que haja mais de uma iteração para cada usuário virtual, preferencialmente que o teste seja feito por um período que permita a todos os usuários estar utilizando o sistema concorrentemente. A repetição dos testes por várias iterações permite ao sistema se adaptar ao número de usuários concorrentes, tornando os tempos de resposta mais próximos de um cenário real.
6. **Temporizador de Sincronização:** o temporizador de sincronização deve ser utilizado em casos que se deseje garantir que uma requisição seja executada por todos os usuários virtuais em simultâneo. Normalmente a requisição a ser sincronizada encontra-se entre outras requisições que podem tirar a sincronização dos usuários virtuais, como, por exemplo, o *login* no sistema, sendo que neste normalmente a funcionalidade que se deseja testar esta após a entrada dos usuários no sistema.

4.4 Definir requisitos, escopo e *framework* para desenvolvimento das APIs para realização dos testes de performance

Para realizar este trabalho serão implementadas duas APIs com o intuito de focar a análise de performance apenas nas tecnologias selecionadas após o procedimento da seção 4.1, sem que tecnologias de outros contextos influenciem na análise da performance. O mesmo acontece com o escopo e requisitos das APIs que serão implementadas, a ideia é reduzir ao máximo interferências de regras de negócio que possam exigir grandes complexidades em determinadas funcionalidades implementadas em uma API e na outra não.

4.4.1 Definir Requisitos das APIs

Os requisitos das APIs que serão os mesmos para todas as APIs, com isso uma decisão tomada para a criação de alguma funcionalidade em uma API refletirá nas outra, procurando evitar que as métricas analisadas tenham valores discrepantes em cada API apenas por mudanças em requisitos.

4.4.2 Definir escopo das APIs

Este passo consistirá na definição do escopo das APIs a serem implementadas para realização dos testes de performance, a ideia inicial é utilizar um mesmo contexto de negócio, onde as regras e decisões relacionadas ao negócio, fosse a mesma tomada para todas as APIs implementadas.

4.4.3 Definir framework

Para a realização dos testes de performance é necessário APIs desenvolvidas com framework da linguagem de programação Javascript, linguagem foco deste presente trabalho, a seleção do *framework* foi baseado em alguns critérios, em relação à quantidade de *frameworks* foi estabelecido a escolha de um. Os critérios de escolha são listados a seguir.

1. Popularidade entre os desenvolvedores *back-end*.
2. Utilização no mercado atual.
3. Comunidade ativa.

4.5 Implementar APIs para realização dos testes de performance

Em uma consulta inicial em repositório de projetos de software como *GitHub* e *Bitbucket*, não foi identificado pelo autor nenhuma API que tivesse duas versões de uma mesma aplicação desenvolvida em uma mesma linguagem ou framework, que utilizasse as tecnologias de implementação de API selecionadas na seção 4.1, dada as opções disponibilizadas. Em alguns casos ainda foi identificado algumas APIs, porém incompletas e sem informações detalhadas de como executar o projeto para realização dos testes. Com isso foi definido no presente trabalho que as APIs seriam implementadas do zero pelo autor.

4.6 Executar testes de performance em APIs

Após o desenvolvimento das APIs serão realizados os testes de carga em cada API, analisando métricas de desempenho selecionadas após a conclusão do passo da seção 4.2. Inicialmente para cada API será criado um plano de teste, onde dentro de cada plano estará todas as requisições listadas que pertence a um determinado cenário real de uma ação realizada pelo usuário, em seguida será definido cargas de acesso, ou seja, uma determinada quantidade

de usuários definida realizando requisições simultaneamente, o mesmo se repetirá para todas as APIs. Após a execução dos testes com as diferentes cargas será analisado as métricas de performance.

4.7 Consolidar e comparar resultados.

Após a realização dos testes de performance será registrado em tabelas seus respectivos resultados para poder ser visualizado e analisado para se tirar considerações dos valores de cada métrica dado cada cenário real de uso simulado.

5 RESULTADOS

Nesta Seção, são apresentados os resultados do presente trabalho.

5.1 Seleção de tecnologias de desenvolvimento de API para análise comparativa de performance do projeto de pesquisa

A escolha das tecnologias se deu por critérios estabelecidos na seção 4.1, foi utilizado algumas plataformas e ferramenta de pesquisa para identificar tecnologias de desenvolvimento de API que atendessem os requisitos para realização dos testes de performance em requisições, dessa forma foram escolhidas as duas mais utilizadas: REST e GraphQL, nas subseções seguintes é descrito de forma mais detalhada cada um dos critérios utilizado para essa escolha.

5.1.1 Comunidade ativa

Para analisar comunidades ativas das tecnologias e contribuições foi selecionado o *Stack Overflow* [1] uma plataforma gratuita de perguntas e respostas para programadores e fonte de pesquisa paras pessoas que codificam em todo o mundo, e o resultados extraídos a partir dela está representado na tabela 2.

Tabela 2 – Análise comparativa de ocorrências no *Stack Overflow*.

Tecnologias de implementação de API	Ocorrências no Stack Overflow
REST	90,448
GraphQL	509
SOAP	460

Fonte: Elaborada pelo autor

As ocorrências no *Stack Overflow* se diz respeito a procura por soluções de problemas encontrados pelos desenvolvedores relacionados a alguma tecnologia específica e até mesmo dúvidas sobre alguma implementação ou configuração de alguma ferramenta. Comunidade ativa se diz respeito aquela que se apresenta em grande número tanto de perguntas quanto de respostas na plataforma. Podemos visualizar na tabela 2 que o REST se apresenta bem a frente em quantidade de ocorrências das demais tecnologias analisadas, o GraphQL como já citado em seções anteriores é uma tecnologia relativamente nova, lançada em 2015 e vindo em ascensão, por conta disso possui menos ocorrências em relação ao REST, que por ser mais consolidado

[1] https://stackoverflow.com/

e está a mais tempo no mercado se mostra com mais buscas na plataforma, quanto ao SOAP tecnologia também antiga, mas que vem sendo gradualmente descontinuada, no entanto, por ainda está presente em muitos projetos de software, ocorrem muitas buscas por soluções de problemas.

5.1.2 Quantidade de aparições em tópicos de pesquisa e artigos publicados.

Para análise de conteúdos já publicados e a quantidade de seus respectivos autores, foi utilizado o *Medium* [2] como fonte de pesquisa, o *Medium* é uma plataforma híbrida online para publicação de jornalismo social, fundada em agosto de 2012 pelo co-fundador do Twitter Evan Williams. A plataforma é um exemplo de jornalismo social, tendo uma coleção híbrida de pessoas e publicações amadoras e profissionais, ou blogs, ou editores exclusivos no *Medium*, e é regularmente considerada um host de blogs. Até hoje foram mais de 7,5 milhões de artigos publicados e mais de 60 milhões de leitores (SPENCER, 2020).

Tabela 3 – Análise de artigos publicados no *Medium*.

Tecnologias de implementação de API	Artigos	Autores
REST	6,1k	4,8k
GraphQL	8,1k	4.95k
SOAP	1.2k	948

Fonte: Elaborada pelo autor

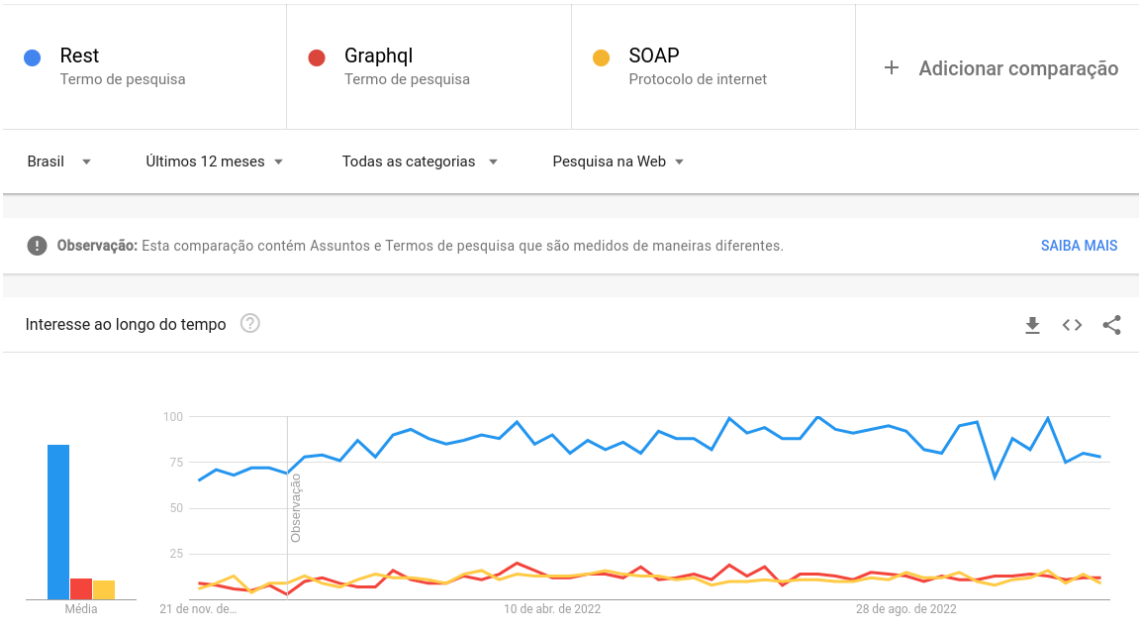
Podemos visualizar na tabela 4 que o GraphQL diferente do resultado visualizado na tabela 3, se apresenta a frente do REST com um número expressivo de 8100 artigos publicados com 4950 autores, logo em seguida o REST com 6100 artigos publicados e 4800 autores, isso significa que cada vez mais os desenvolvedores estão adotando essas tecnologias e criando conteúdo, tutoriais e até mesmo cursos na plataforma *Medium*, então quanto mais pesquisado sobre o GraphQL e o REST, por exemplo, mais artigos publicados irão parecer como resultado da pesquisa, isso mostra a grande adoção por elas e em relação ao SOAP podemos analisar o quanto ele se apresenta distante das outras duas tecnologias comparadas, justamente por desenvolvedores estarem descontinuando essa tecnologia em projetos de softwares modernos e até mesmo os mais antigos em migração para as tecnologias mais atuais.

[2] https://medium.com/

5.1.3 Popularidade

Para uma análise de popularidade entre as tecnologias nas comunidades de desenvolvimento foi selecionado o *Google Trends* [3], uma ferramenta gratuita que permite a descoberta das principais tendências relacionadas a uma palavra-chave específica. Esse buscador do Google indica o que tem sido mais pesquisado conforme a localização e o período determinado pelo usuário (CASTRO, 2021).

Figura 4 – Resultados da pesquisa realizada na ferramenta Google Trends



Fonte: Elaborado pelo autor

Na Figura 4 podemos visualizar que a popularidade do REST na comunidade de desenvolvimento é muito superior entre as tecnologias analisadas deste presente trabalho, se mostra a tecnologia mais buscada nos últimos 12 meses conforme o *Google Trends* com uma média (88), o SOAP mais uma vez se apresenta como último nessa análise comparativa, obtendo a pior média (6) e o GraphQL ainda que distante do REST, mas se mostra bem a frente do SOAP, ficando ali entre as duas tecnologias (17), como a segunda mais popular.

5.2 Seleção de métricas de performance

Para realização dos testes de cargas nas APIs implementadas, foi considerado as métricas descritas na seção 2.2, dessa forma foram selecionadas as seguintes métricas de performance em APIs alvo do presente trabalho:

[3] https://trends.google.com.br/trends/?geo=BR

1. Menor tempo de resposta das requisições.
2. Maior tempo de resposta das requisições.
3. Média dos tempos de resposta por requisição.
4. Quantidade máxima de requisições enviadas.
5. Quantidade de *Kilobytes* recebidas.
6. Quantidade de *Kilobytes* enviadas.

5.3 Definição da ferramenta de *benchmarking* para realização da análise de performance

Foi definido a utilização da ferramenta *JMeter* [4]. Foi considerado ferramentas utilizadas em trabalhos relacionados e disponíveis na literatura, assim como considerando que a ferramenta disponibilizasse dos recursos descritos na seção 4.3.

5.4 Definição do escopo, requisitos e *frameworks* para desenvolvimento das APIs para realização dos testes de performance

Nesta seção será apresentado escopo, requisitos e *frameworks* definidos para a aplicação dos testes de performance em requisições as APIs proposta pelo presente trabalho.

5.4.1 Definição dos Requisitos

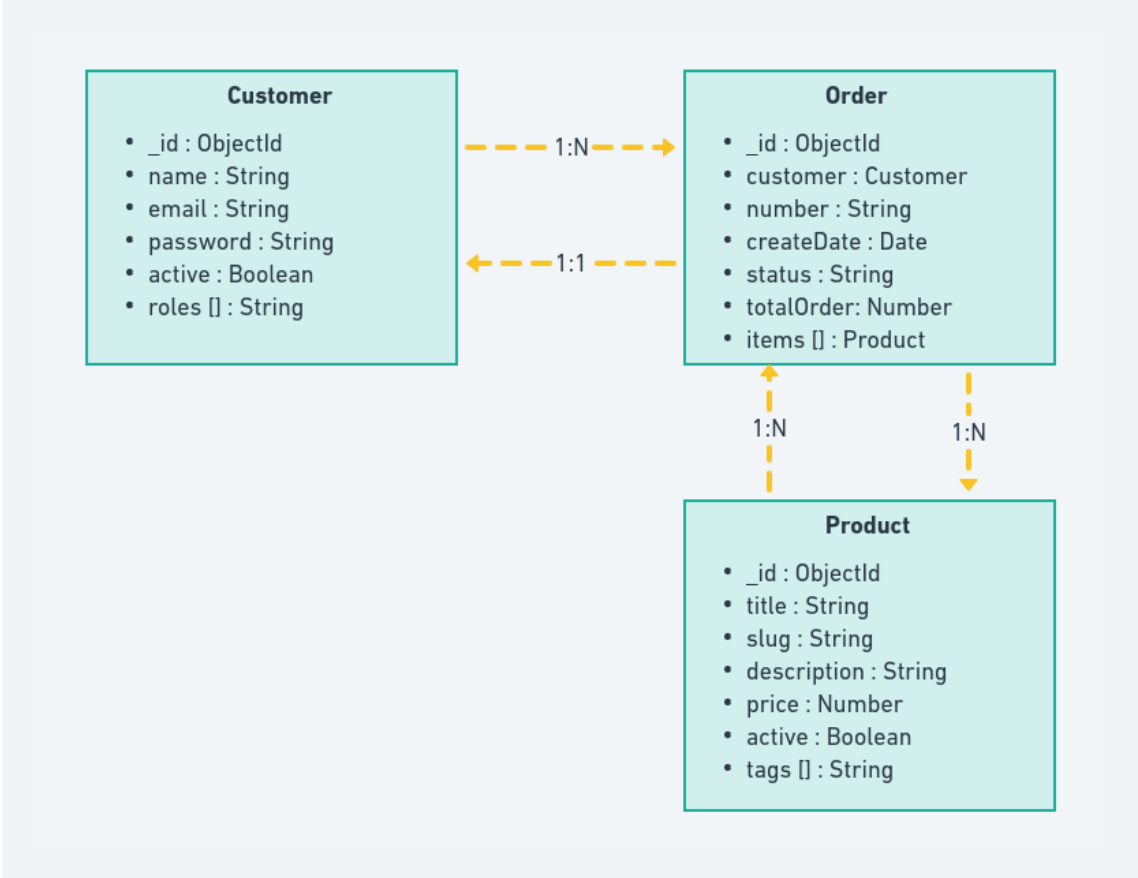
Foi especificado que todas as entidades devem possuir os métodos CRUD (*Create, Read (List e Detail), Update, Delete*), dado que esses métodos utilizados do CRUD conseguem abordar diversos contextos e cenários de requisições, sendo possível criar diversas variações de funcionalidades. Os métodos de listagem devem possuir filtros pelas propriedades definidas.

5.4.2 Definição do escopo

Após a realização das pesquisas foi definido que o escopo das APIs fossem voltadas para um sistema de gerenciamento de um *e-commerce* simples. No sistema são gerenciados clientes, produtos e pedidos. A API deve armazenar e devolver os dados quando solicitados. As entidades, relacionamentos e suas propriedades estão representadas na modelagem de banco de dados na Figura 5.

[4] https://jmeter.apache.org/

Figura 5 – Modelagem do banco de dados das APIs



Fonte: Elaborado pelo autor

5.4.3 Definição dos frameworks

Foi definido a utilização da linguagem de programação *JavaScript* para implementação das APIs, utilizando um dos *frameworks* do NodeJS. Baseado nos critérios estabelecidos na subseção 4.4.3, para atender os critérios de popularidade entre os desenvolvedores *back-end* e utilização no mercado atual foi buscado no site StateOfJs [5] o ranking dos frameworks *back-end* mais utilizados em 2021 ao qual o *Express* se mostra em larga vantagem com 81% de utilização como podemos visualizar na tabela 4, seguido pelo segundo o *Next* com apenas 45%, quase que a metade em relação ao *Express*. Já para atender o critério de comunidade ativa foi utilizado a plataforma de hospedagem de código *GitHub* onde estão disponibilizados os repositórios dos *frameworks* pesquisados (como também os diversos projetos desenvolvidos com o mesmo), a análise dos repositórios partiu dos resultados da pesquisa anterior das mais utilizadas, sendo limitada a uma quantidade de 6, foi então possível observar mais uma vez na tabela 4 a vantagem do Express em relação ao restante, se apresentando com 17 milhões de usuários e 294 contribui-

[5] https://stateofjs.com/en-us/

dores, representando assim uma comunidade bem ativa com base nos critérios utilizados e os dados apresentando na Tabela 4 o *Framework* selecionado foi o *Express*, para a implementação das APIs REST e GraphQL.

Tabela 4 – Ranking dos frameworks mais utilizados em 2021 - StateOfJs e informações dos respectivos repositórios no *GitHub Stack Overflow*.

Frameworks	Porcentagem de uso no StateOfJs (2021)	Informações do Repositório no GitHub
Express	81%	5,740 commits, 10k forks 58.9k estrelas, 294 contribuidores e mais de 17 milhões de usuários
Next	45%	13,236 commits, 21.2k forks 96.6k estrelas, 2,393 contribuidores e mais de 1 milhão de usuário
Gatsby	27%	20,858 commits, 10.5k forks 53.8k estrelas, 3,929 contribuidores e mais de 454 mil de usuários
Nuxt	20%	5,590 commits, 3.3k forks 41.6k estrelas, 325 contribuidores e mais de 143 mil de usuário
Nest	19%	10,847 commits, 5.5k forks 48.1k estrelas, 314 contribuidores e mais de 162 mil de usuários
Strapi	13%	25,214 commits, 6k forks 49.9k estrelas, 819 contribuidores e mais de 25.4 mil de usuários

Fonte: Elaborado pelo autor

O intuito também é utilizar o mesmo banco de dados para as aplicações, dessa forma a análise das métricas de performance vão se concentrar mais nas tecnologias de desenvolvimento de API: REST e GraphQL. O repositório das APIs desenvolvidas com o *framework* (escolhido pelo autor no presente trabalho) para realização dos testes de performance pode ser acessado através: <github.com/FabricioNPinheiro/APIs-server-tcc>

5.5 Execução dos Testes de Performance

Está seção apresenta os resultados e descreve como foi realizado a execução dos testes de performance das tecnologias de desenvolvimento de API: REST e Graphql escolhidos na seção 5.1. Os casos de testes foram realizados em uma máquina com a seguinte configuração:

1. **Modelo**: Acer Aspire 5 A515-54G-77RU.
2. **Processador**: Intel Core i7 10510U (10ª geração) – 8 MB de cache 4 núcleos e 8 threads – de 1.80 GHz até 4.90 GHz.
3. **Memória(RAM)**: 12 GB DDR4 2666 MHz.

4. **Placa de Vídeo**: NVIDIA GeForce MX250 2 GB GDDR5 GPU do tipo dedicada.

5.5.1 Plano de teste

Como é possível observar na figura 6 que representa o plano de testes que será executado, foi definido pelo autor três cenários para serem executado os testes: realização de um pedido, gerenciamento de clientes e produtos e filtros/consultas, assim como o tipo de usuário referente a cada cenário que simula um caso de uso real.

Cenário 1 A realização de um pedido é uma das rotinas mais executadas de forma mais frequente em um *ecommerce* pelos clientes que desejam efetuar uma determinada compra, dessa forma a ideia é verificar como se comportam as APIs em termos de performance com uma grande carga de requisição. No que se diz respeito as requisições selecionadas para esse cenário, foi partindo de um determinado momento que o usuário estaria na página principal do site, a partir desse momento irá fazer pesquisas por produtos utilizando diferentes filtros, em seguida ao selecionar um produto ele será redirecionado para realizar seu cadastro e posteriormente login na plataforma, então será possível completar sua compra e poderá fica analisando o *status* do pedido assim como os detalhes do mesmo.

Cenário 2 O gerenciamento de clientes e produtos por parte do administrador do *ecommerce* se trata de uma rotina bastante utilizada pelos gestores da empresa que detém o produto, por exemplo, de modo a adicionar, atualizar, listar e remover novos produtos e clientes.

Cenário 3 O último cenário foi definido pelo autor após uma análise de que os filtros/consultas são requisições que lidam com maior quantidade de dados, dado que é enviado a requisição para o servidor, em seguida buscada no banco de dados e retornada para o usuário, passando por diversas etapas até sua finalização completa, com isso exige mais do servidor e um teste de performance com todos os filtros/consultas seria ideal para analisar o comportamento performático das APIs.

5.5.2 Execução dos testes

Durante a realização dos testes foi utilizado diferentes tempos de carga em cada um dos cenários, que se diz respeito ao tempo em segundos que aquela carga de requisição fica sendo executada no servidor, dado as configurações da máquina utilizada pelo autor em questão de suportar todo o processo foi definido os seguintes parâmetros:

1. Duração das cargas de testes em segundos: **120sec**.

2. Tempo de limpeza do cache de uma carga de requisições para outra em segundos: **20sec**.
3. Número de *Threads* (usuários): **1600**.

As análises foram feitas através da ferramenta JMeter definida na seção 5.3 e os dados interpretados e inseridos em tabela para melhor uma melhor visualização dos resultados.

5.5.2.1 Cenário de realização de um pedido

Para o cenário de realização de um pedido foi selecionado requisições que tivesse no contexto de efetuar uma compra em um *ecommerce*, como, por exemplo, buscar pedidos que se refere a busca ali por um determinado produto utilizando algum filtro específico, se cadastrar, realizar um pedido, buscar por pedidos dentre outros, todas as requisições selecionadas para esse cenário estão representados na Tabela 5.

Tabela 5 – Casos de testes para o cenário de realização de um pedido.

Caso de Teste	Requisição	Descrição
CT1	find-all-products	Listar produtos
CT2	find-product-by-slug	Listar produto por slug
CT3	find-product-by-tag	Listar produto por tag
CT4	find-product-by-id	Listar produto por id
CT5	create-customer	Cadastro de usuário
CT6	create-order	Realizar pedido
CT7	find-order-by-status	Listar pedido por status
CT8	find-order-by-id	Listar pedido por id
CT9	find-order-by-number	Listar pedido por número
CT10	update-order-field-by-id	Atualizar pedido por id
CT11	delete-order-by-id	Deletar pedido por id

Fonte: Elaborado pelo autor

Observando a tabela 6 é possível notar que o GraphQL suportou um número maior de requisições (1.205,810) comparado ao REST (1.198,468), além de uma quantidade maior de requisições enviadas o GraphQL mostrou vantagem em métricas como menor tempo de resposta das requisições (0.0113), maior tempo de resposta das requisições (0.0378) e média do tempo de resposta das requisições (0.0301), obtendo assim resultados melhores, enquanto o REST apresentou (0.0156), (0.1946) e (0.0395), respectivamente, o que chama atenção é o valor da métrica de maior tempo de resposta do REST, com um valor bem superior a qualquer outra

Tabela 6 – Resultados dos testes para simulação do cenário de realização de pedido.

Métricas	REST	GraphQL
Menor tempo de resposta das requisições	0.0156ms	0.0113ms
Maior tempo de resposta das requisições	0.1946ms	0.0378ms
Média do tempo de resposta das requisições	0.0395ms	0.0301ms
Quantidade máxima de requisições enviadas	1.198,468	1.205,810
Quantidade de kilobytes enviadas (KB/s)	14.861	13.279
Quantidade de kilobytes recebidas (KB/s)	29.246,674	21.179,605

Fonte: Elaborado pelo autor

requisição de maior tempo de resposta do GraphQL, vale destacar também que a quantidade de *kilobytes* recebidas por segundo no GraphQL (21.179,605) é bem menor que a quantidade recebida pelo REST (29.246,674), neste resultado pode estar relacionado ao armazenamento de cachê de cada uma das tecnologias, ressurtindo que o GraphQL pode ter um armazenamento maior.

5.5.2.2 Cenário de gerenciamento de clientes e produtos

Tabela 7 – Casos de testes para o cenário de gerenciamento de produtos e clientes.

Caso de Teste	Requisição	Descrição
CT1	find-all-customers	Listar clientes
CT2	find-customer-by-id	Listar cliente por id
CT3	find-customer-by-email	Listar cliente por email
CT4	find-all-customers-active	Listar clientes ativos
CT5	find-all-customers-by-roles	Listar clientes por permissão
CT6	update-customer-field-by-id	Atualizar cliente por id
CT7	delete-customer-by-id	Deletar cliente por id
CT8	find-all-products	Listar produtos
CT9	find-product-by-slug	Listar produtos por slug
CT10	find-product-by-tags	Listar produtos por tag
CT11	find-product-by-id	Listar produto por id
CT12	update-product-field-by-id	Atualizar pedido por id
CT13	delete-product-by-id	Deletar produto por id

Fonte: Elaborado pelo autor

A tabela 8 apresenta os resultados referentes ao cenário de gerenciamento de produtos e clientes, podemos analisar que o GraphQL possui melhores resultados referentes a métricas

Tabela 8 – Resultados dos testes para simulação do cenário de gerenciamento de produtos e clientes.

Métricas	REST	GraphQL
Menor tempo de resposta das requisições	0,0181ms	0,0162ms
Maior tempo de resposta das requisições	0,1829ms	0,0471ms
Média do tempo de resposta das requisições	0,0312 ms	0,0323ms
Quantidade máxima de requisições enviadas	1.228.166	1.193.803
Quantidade de kilobytes enviadas (KB/s)	7.792	6.294
Quantidade de kilobytes recebidas (KB/s)	31.345.678	24.032.864

Fonte: Elaborado pelo autor

como menor tempo de resposta das requisições (0.0162), maior tempo de resposta das requisições (0.0471) e média do tempo de resposta das requisições (0.0329), enquanto REST apresenta resultados inferiores dessas métricas (0.0181), (0.1829) e (0.0307), respectivamente. No entanto, no que se diz respeito a quantidade máxima de requisições enviadas, o REST conseguiu suportar uma quantidade maior (1.228,166) comparado ao GraphQL (1.193,803), em quantidade total de *kilobytes* enviadas o GraphQL recebe uma quantidade menor (24.032,864) de dados, enquanto o REST apesar de executar mais requisições que o GraphQL, mas ainda assim é uma quantidade bem superior (31.345,678).

5.5.2.3 Cenário de filtros/consulta

Como podemos observar na tabela 10, o REST obteve um melhor resultado no que se diz respeito a métrica de quantidade máxima de requisições executadas pelo sistema (1.292,532) dado o número de 1600 *threads* executadas por um período de 120 segundos, o sistema suportou uma carga maior de requisições enviadas por segundo (10427/s), enquanto o GraphQL obteve um resultado inferior na quantidade máxima de requisições executadas (1.126,027) obtendo consequentemente um valor inferior de requisições enviadas por segundo (7.986).

Dessa forma é possível inferir que em um cenário de filtros onde existem muitas consultas ao sistema para se buscar dados, entre elas consultas de maior complexidade e outras que trazem apenas simples dados sem passar algum tipo de filtro, o REST nesse contexto se apresenta como uma alternativa melhor a ser utilizada. Nessas cargas de requisições a ferramenta também nos traz métricas mais específicas mostrando que em todas as requisições executadas qual delas obteve menor e maior tempo de resposta das requisições, como também a média desses tempos de resposta, com o REST obtendo os resultados em milissegundos de (0.171) em

Tabela 9 – Casos de testes para o cenário de filtros/consultas.

Caso de Teste	Requisição	Descrição
CT1	find-all-customers	Listar clientes
CT2	find-customer-by-id	Listar cliente por id
CT3	find-customer-by-email	Listar cliente por email
CT4	find-all-customers-active	Listar clientes ativos
CT5	find-all-customers-by-roles	Listar clientes por permissão
CT6	find-all-products	Listar produtos
CT7	find-product-by-slug	Listar produtos por slug
CT8	find-product-by-tags	Listar produtos por tag
CT9	find-product-by-id	Listar produto por id
CT10	find-all-orders	Listar pedidos
CT11	find-order-by-status	Listar pedidos por status
CT12	find-order-by-id	Listar pedidos por id
CT13	find-order-by-number	Listar pedido por

Fonte: Elaborado pelo autor

Tabela 10 – Resultados dos testes para simulação do cenário de filtros/consultas.

Métricas	REST	GraphQL
Menor tempo de resposta das requisições	0.0171ms	0.0186ms
Maior tempo de resposta das requisições	0.0398ms	0.0413ms
Média do tempo de resposta das requisições	0.0293ms	0.0392ms
Quantidade máxima de requisições enviadas	1.292,532	1.126,027
Quantidade de kilobytes enviadas (KB/s)	1.494	1.378
Quantidade de kilobytes recebidas (KB/s)	32.473,541	25.057,177

Fonte: Elaborado pelo autor

menor tempo de requisições por segundo, (0.0398) de maior tempo de resposta de requisição e uma média dos tempos de resposta de (0.0293), já o GraphQL apresentou (0.0186), (0.0413) e (0.0302), respectivamente.

Em relação à quantidade de Kilobytes recebidas por segundo, os resultados chama atenção pela diferença entre elas, enquanto o REST apresenta um valor expressivo de 32.473,541 de KB/s, o GraphQL já obtém um resultado menor de 25.057,17 KB/s, mostrando que o GraphQL armazena um cachê maior de informações retornadas, dado que esse valor é calculado dos dados que vem do servidor conectado a um banco de dados.

6 CONSIDERAÇÕES FINAIS

Através deste trabalho foi possível observar que apesar das diversas tecnologias disponibilizadas no mercado para o desenvolvimento de software, tecnologias recém lançadas que vem com o intuito de resolver determinado problema, não significa que resolverá todos, o mesmo se aplica para tecnologias mais antigas e consolidadas, na escolha de qual tecnologia é a mais adequada para um determinado projeto se torna imprescindível analisar todos os contextos ao, qual o produto que será desenvolvido está incluído, como regras de negócios, requisitos e tecnologias utilizadas, verificar também se os *frameworks* atendem bem as necessidades de projeto de *software* que visa ter qualidade e performance.

Desta forma, um dos resultados gerais do presente trabalho foi uma comparação de performance entre tecnologias de desenvolvimento de API, analisando os resultados apresentados por métricas de performance, através de uma ferramenta de *benchmarking* visando ajudar desenvolvedores que buscam desenvolver software performáticos e não precisa se preocupar depois se o sistema vai fica fora do ar por conta de uma grande quantidade de acessos.

Além disso, foi realizado testes de carga em requisições das APIs desenvolvidas, buscando criar simulações de cenários reais de uso, as requisições foram agrupadas de forma que atendesse algum tipo de necessidade dos usuários ao acessar um sistema de *e-commerce*, dessa forma foram definidos cenários conforme as regras de negócio e requisitos das APIs desenvolvidas, a ideia era usar o mesmo conjunto de decisões de desenvolvimento para todas, focando a análise de performance no GraphQL e REST, tecnologias de desenvolvimento de API alvo deste presente trabalho.

Os resultados obtidos foram relevantes para analisar além do cenário os tipos de requisições que influenciam uma tecnologia ou outra, ao obter resultados melhores em alguns casos e piores em outros. O GraphQL, por exemplo, ao ter mais requisições de criar registro em banco de dados, como o caso do POST nos métodos do REST e a *mutation* no GraphQL, responsáveis em criar um registro no sistema, ele se mostra um pouco superior ao REST, mas a partir do momento em que o cenário apresenta mais casos de filtros/consultas, como o caso do GET em REST e *query* em GraphQL que busca dados no servidor, nesse cenário o REST se deu melhor, obtendo resultados mais satisfatórios se comparado ao GraphQL.

Contextualizando em uma análise mais geral os filtros/consultas são uma problemática para o GraphQL principalmente em grandes cargas, apenas no cenário de realizar pedido o GraphQL se deu melhor por conta de poucas requisições desse contexto, já requisições de criação

se mostraram um problema para o REST em grandes cargas testadas, mas saiu em vantagem nos cenários de gerenciamento de clientes e produtos e no cenário de filtros/consultas onde possuem mais requisições de buscar dados na API. Dessa forma, é reforçado que o entendimento claro do contexto do que será desenvolvido é super importante para que assim se tome decisões mais coerentes de qual tecnologia escolher para implementação e desenvolvimento de um projeto de software.

REFERÊNCIAS

APPMASTER. **O que é a API REST e como ela difere de outros tipos?** 2022. Disponível em: https://appmaster.io/pt/blog/o-que-e-a-api-rest-e-como-ela-difere-de-outros-tipos/. Acesso em: 10 jun. 2022.

BANKS, E. P. e A. **Introdução ao GraphQL**: Busca de dados com abordagem declarativa para aplicações web modernas. [*S. l.*]: Novatec Editora, 2018.

CASTRO, I. N. de. **Entenda o que é Google Trends e como ele pode ajudar a sua empresa**. 2021. Disponível em: https://rockcontent.com/br/blog/google-trends/. Acesso em: 15 de set 2022.

COSTA, J. **APIs**: Tudo sobre o que é api, exemplos e importância! 2022. Disponível em: https://www.zenvia.com/blog/apis-entenda-o-que-sao-e-como-funcionam/. Acesso em: 15 jun. 2022.

COUTO, T. de Almeida Correia e T. **Teste de performance em aplicacoes web**: Garantindo o desempenho de uma aplicacao. [S.l: s.n], p. 23–30, 2015.

FLANAGAN, D. **JavaScript**: o guia definitivo. [*S. l.*]: Bookman Editora, 2004.

LAWI, B. L. E. P. e. T. Y. A. Evaluating graphql and rest api services performance in a massive and intensive accessible information system. **Computers**, MDPI, 2021.

LOPES, L. H. d. F. S. e. L. P. I. S. M. **Análise do Custo-Benefício de GraphQL em Comparação a REST e SOAP**: em aplicações para dispositivos móveis. [S.l: s.n], 2019.

LUCAS, S. **Primeiros passos com testes de performance e JMeter**. 2019. Disponível em: https://medium.com/@samuellucas/primeiros-passos-com-testes-de-performance-e-jmeter-a96b4db360ab. Acesso em: 14 jun. 2022.

MOLINARI, L. **Testes de Performance**. [*S. l.*]: Visual Books, 2009.

QUIQUE. **What is GraphQL? Learn it in 5 minutes**. 2022. Disponível em: https://pragmaticreviews.com/what-is-graphql-learn-it-in-5-minutes/. Acesso em: 11 jun. 2022.

RIBEIRO, L. Graphql: An alternative to webservices. **Brazilian Journal of Development**, Brazilian Journals Publicações de Periódicos e Editora Ltda, v. 7, n. 12, p. 1–45, 2021.

SEABRA, M. F. N. e. G. P. M. Rest or graphql? a performance comparative study. XIII Brazilian Symposium, 2019.

SILVA, M. S. **JavaScript - Guia do Programador**: Guia completo das funcionalidades de linguagem javascript. [*S. l.*]: Novatec Editora, 2010.

SOARES, N. A. **Graphql uma abordagem alternativa ao padrão rest**. 41 p. Monografia (TCC) – Universidade do Sul de Santa Catarina, Florianópolis, 2020.

SPENCER, G. **Medium.com – o melhor site de Blogs:**: o que é, para que serve e como funciona. 2020. Disponível em: https://www.tecnoveste.com.br/medium-com-o-melhor-site-de-blogs-o-que-e-para-que-serve-e-como-funciona/. Acesso em: 20 set. 2022.

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