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

Iara Siria Morais Lopes[1], Lucas Henrique de Faria Silva[1], Lesandro Ponciano[1]

[1]Bacharelado em Engenharia de Software
Instituto de Ciências Exatas e Informática – PUC Minas
Ed. Fernanda. Rua Cláudio Manoel, 1.162, Funcionários, Belo Horizonte – MG – Brasil

{iarasiria,lhfsilva}@sga.pucminas.br, {lesandrop}@pucminas.br

Resumo. *GraphQL é uma linguagem de consulta criada como alternativa a Representational State Transfer (REST) e Simple Object Access Protocol (SOAP) para implementar web services. Apesar de serem abordagens amplamente utilizadas, tem-se uma ausência da análise de custo-benefício ao escolhê-las para implementação de web services no contexto de aplicações para dispositivos móveis. Neste estudo, compara-se as três abordagens em termos de seus custos e benefícios em tempo de design e em tempo de implementação, na perspectiva de desenvolvedores de aplicações para dispositivos móveis, e tempo de execução, através de uma análise de desempenho. Para isso, realiza-se uma análise de cada abordagem utilizando as heurísticas de Nielsen para tratar da usabilidade de Application Programming Interfaces (API) e o modelo de qualidade ISO 25010. Os resultados demonstram que REST e GraphQL são equivalentes no tempo de execução. No tempo de design, REST apresentou vantagens distintas de GraphQL quanto à usabilidade de API. Quanto ao tempo de implementação, GraphQL tem vantagem na validação de dados recebidos e REST sobre a criação automática dos diretórios necessários para a codificação. A abordagem SOAP apresenta mais desvantagens do que as outras duas alternativas em todos os tempos.*

Palavras-chave: Dispositivos móveis, GraphQL, REST, SOAP, web services

1. Introdução

O desenvolvimento de aplicações distribuídas muitas vezes é baseado em *web services*, que são uma maneira comum de trocar dados e informações através da Internet [Kumari and Rath 2015]. Existem diversas abordagens para acessos à dados através de *web services*, como: *Representational State Transfer* (REST), *Simple Object Access Protocol* (SOAP) e *Graph Query Language* (GraphQL). SOAP é a abordagem mais antiga entre as três [Serrano, Hernantes and Gallardo 2014], enquanto, ao longo dos anos, REST se tornou a abordagem mais utilizada para a implementação de *web services* [Vogel, Weber and Zirpins 2018]. GraphQL, lançado em 2016, é uma nova alternativa, na qual o consumidor do *web service* solicita somente os dados necessários para sua solução, gerando benefícios de desempenho [Vogel, Weber and Zirpins 2018]. Apesar de serem abordagens conhecidas, não se tem conhecimento do custo-benefício da adoção dessas abordagens, na visão de programadores de aplicações para

dispositivos móveis. O contexto dessas aplicações apresenta maior heterogeneidade e dinamicidade de tipo e qualidade de conexão, tornando a otimização da troca de dados com servidores relevante.

Neste contexto, o problema tratado neste trabalho é a **ausência de análises de custo-benefício do uso das abordagens GraphQL, SOAP e REST, para a integração de *web services* com aplicações para dispositivos móveis**. Para realizar a análise de custo-benefício das abordagens de desenvolvimento de *web services*, pode-se considerar diversas características e métricas. Neste trabalho, a ênfase é dada a 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.

Desde o seu lançamento, GraphQL foi adotado por empresas como GitHub, Coursera, Pinterest e Twitter [Hartig and Pérez 2018]. Diversos benefícios já foram relatados em sua implementação, como a realização de consultas eficientes que retornam apenas o que foi requisitado [Seda et al. 2018]. No entanto, ainda há uma carência de estudos que quantifiquem o custo-benefício do uso de GraphQL em comparação às abordagens SOAP e REST no contexto de aplicações para dispositivos móveis. Desta forma, o objetivo principal deste estudo é **investigar o custo-benefício do uso de GraphQL como alternativa às abordagens de implementação de *web services* SOAP e REST, no contexto de aplicações para dispositivos móveis**. Para alcançar esse objetivo, são estabelecidos os seguintes objetivos específicos: i) realizar a avaliação das três abordagens considerando diferentes tempos do ciclo de vida de uma aplicação; ii) colocar em perspectiva o custo-benefício da adoção de GraphQL em comparação a SOAP e REST.

Pode-se citar duas principais contribuições. Primeira, a análise de características da usabilidade de *Application Programming Interfaces* (APIs) e adequação funcional, em conjunto com métricas de desempenho para cada abordagem de implementação de *web services*, para o contexto de dispositivos móveis. Segunda, uma análise das vantagens e desvantagens das abordagens REST, GraphQL e SOAP no *design*, implementação e execução de aplicações para dispositivos móveis.

O restante deste trabalho está dividido em seis seções. A seção 2 apresenta mais detalhes sobre as abordagens de implementações de *web services* GraphQL, SOAP e REST, bem como o seu uso em aplicações reais. A seção 3 descreve os principais trabalhos relacionados. A seção 4 detalha os materiais e métodos. A seção 5 apresenta os resultados do trabalho. Por fim, na seção 6 são apresentados as conclusões e trabalhos futuros.

2. Referencial Teórico

O referencial teórico apresentado nesta seção contém os principais conceitos abordados pelo estudo. O primeiro é referente a aplicações para dispositivos móveis, contexto em que o trabalho é aplicado. O segundo apresenta as abordagens para implementação de *web services* escolhidas para a comparação no estudo. Por fim, apresenta-se o conceito

de usabilidade de APIs e o modelo de qualidade ISO 25010, com o objetivo de contextualizar os fundamentos da metodologia aplicada.

2.1. Aplicações para Dispositivos Móveis

Aplicações para dispositivos móveis são sistemas usados em vários campos e para diversos tipos de serviços. Sendo altamente interativas, encorajam usuários, por exemplo, a participarem de redes sociais, facilitando a troca de mensagens, a interação através de comentários e o compartilhamento de diversos tipos de mídias [Alaa and Hussein 2009]. Isto posto, essas aplicações, a partir do momento em que utilizam a Internet para prover conexões entre dispositivos, podem ser caracterizadas pela interatividade com uma API de um servidor remoto, o que pode ser quantificado em termos de transferência de mensagens e dados, havendo também ênfase no tempo de resposta [Arazy et al. 2010].

Além disso, ao se considerar o número de requisições simultâneas realizadas em aplicações como Instagram, Facebook e Youtube, o tempo total para processar todas estas requisições é diretamente impactado, determinando a taxa de vazão de dados do servidor para os dispositivos móveis [Badidi 2011]. Os serviços e funcionalidades oferecidos devem ser adequados com o dispositivo móvel e suas especificações, junto às expectativas do usuário. Isso determina a qualidade geral das aplicações para dispositivos móveis [Rajan, Malini and Sundarakantham 2014]. Dessa forma, tem-se como desafio diante de aplicações móveis a garantia de acesso eficiente, atendendo critérios relacionados ao tempo de resposta e taxa de vazão [Chalmers and Sloman 2009].

2.2. Abordagens para Implementação de *Web Services*

A web é geralmente utilizada por pessoas para acessar hipertexto e outros tipos de mídias através de navegadores. Porém, cada vez mais, a web é utilizada para a comunicação entre dispositivos. A comunicação entre máquinas através da Internet é realizada através da utilização de *web services* [Shelby 2010]. Um *web service* é identificado através de *Uniform Resource Identifier* (URI). De acordo com a abordagem de implementação do *web service* (REST ou SOAP, por exemplo), o serviço pode ter um ou vários *endpoints*. Um *endpoint* é uma URI para cada recurso ou uma única URI representando o *web service* [AlShahwan and Moessner 2010].

A forma mais comum de realizar a comunicação entre aplicações para dispositivos móveis na Internet é através de *web services*. Para isso, as abordagens mais utilizadas para implementação de *web services* são o protocolo SOAP, o estilo arquitetural REST e a linguagem de consulta GraphQL [Seda et al. 2018]. A escolha da abordagem para a implementação de um *web service* afeta o *design* de sua arquitetura no geral, o que impacta os tempos de *design*, implementação e execução.

2.2.1. REST

REST é um estilo arquitetural baseado em uma arquitetura cliente-servidor. Ele é utilizado em conjunto com o *Hypertext Transfer Protocol* (HTTP) [Belqasmi et al. 2012]. *Web services* que atendem as restrições impostas pelo estilo REST são chamadas de *RESTful* APIs [Kumari and Rath 2015]. Em um sistema desenvolvido usando REST,

cada recurso é unicamente direcionado através de sua URI. Um recurso é qualquer forma de informação que pode ser nomeada e é importante o suficiente para ser referenciada [Belqasmi et al. 2012]. Em *RESTful* APIs, o servidor de aplicação implementa uma lista de *endpoints* (cada *endpoint* representa um recurso diferente do *web service*) que podem ser chamadas pelos clientes (aplicações para dispositivos móveis, por exemplo).

Quatro operações principais definem a interface REST: criar, ler, atualizar e deletar, implementadas usando, respectivamente, *POST*, *GET*, *PUT* e *DELETE*. *POST* é utilizado para a criação de novos dados. *GET* é utilizado para a recuperação de dados. *PUT* é utilizado para a atualização de dados e *DELETE* é utilizado para deletar dados. Cada requisição REST contém toda a informação necessária para que o servidor realize o seu processamento, não havendo nenhuma dependência entre requisições diferentes [Fielding and Taylor 2002].

2.2.2. SOAP

SOAP é um protocolo criado para a troca de mensagens baseado no formato *Extensible Markup Language* (XML). Cada requisição feita utilizando o protocolo SOAP contém um arquivo XML com todos os dados necessários para o processamento dela [Martin et al. 2007]. *Web services* baseados em SOAP possuem uma única linguagem de descrição, denominada como *Web Services Description Language* (WSDL) [Kumari and Rath 2015]. A WSDL provê informação sobre como utilizar o *web service*, incluindo a descrição e o modo de utilizar o mesmo [Belqasmi et al. 2012].

Uma arquitetura de *web services* baseada em SOAP é composta de três componentes: 1) o provedor de serviços, responsável pela publicação do *web service*; 2) o registro do servidor, componente que documenta todas as operações disponibilizadas; e 3) o consumidor, que faz a conexão com o serviço e o utiliza para realizar requisições [Belqasmi et al. 2012]. Uma mensagem SOAP consiste de um envelope, baseado em XML. Esse envelope tem um cabeçalho opcional e um corpo obrigatório. O cabeçalho contém os metadados da mensagem, e o corpo define os dados enviados pela mensagem [Sungkur and Daiboo 2015].

2.2.3. GraphQL

Essencialmente, GraphQL permite que clientes realizem consultas em banco de dados através de um esquema [Brito et al. 2019], que representa o modelo de dados existente no sistema. Através desse esquema, é possível fazer requisições a um único *endpoint*, onde, no corpo da requisição, são enviados os dados esperados pelo consumidor do serviço [Hartig and Pérez 2017].

GraphQL, ao invés de ter uma lista fixa de operações, expõe uma interface de um banco de dados, que pode ser consultada pelos clientes [Brito et al. 2019]. Para ilustrar o uso desta linguagem, pode-se utilizar um sistema simples sobre um catálogo de curso. Como mostrado no exemplo apresentado na Figura 1(a), o banco de dados possui dois objetos: *Curso*, que possui quatro atributos (*id*, *nome*, *duração* e *coordenador*) e *Coordenador*, que possui três atributos (*id*, *nome*, *email*). Um esquema GraphQL, geralmente, também inclui um objeto do tipo *Query*. Esse objeto é responsável por expor os tipos de objetos que podem ser consultados por clientes. Por

exemplo, na linha 15, *curso* é uma consulta que aceita um *id* como argumento e retorna o objeto *Curso* que tem o *id* igual ao parâmetro enviado.

GraphQL também define uma linguagem de consulta, que é utilizada pelos clientes. A Figura 1(b) mostra três exemplos destas consultas. Na primeira consulta (*CursoPorNome*), o cliente requisita pelo objeto *Curso* com o atributo *id* igual a 100, que solicita, explicitamente, apenas o nome do *Curso*. A segunda consulta (*CursoPorDuracaoENome*) é similar, porém agora o cliente solicita dois atributos: nome e duração. Na última consulta (*CursoPorCoordenadorENome*), o cliente solicita os atributos *nome* e *coordenador* do *Curso*. Já que o atributo *coordenador* é outro objeto, é necessário também especificar quais atributos são necessários do objeto *Coordenador*. Como apresentado na Figura 1(c), o resultado é um objeto JSON contendo apenas os dados necessários.

(a) Esquema de um serviço GraphQL

(b) Consultas de um serviço GraphQL, definido pelo cliente-

(c) Retorno JSON da consulta CursoPorCoordenadorENome

Figura 1. Esquema e consultas de um serviço implementado com GraphQL

2.3. Usabilidade de APIs

APIs são interfaces fundamentais para as interações entre programadores e computadores, sendo um componente essencial no desenvolvimento de software moderno [Piccioni, Furia and Meyer 2013]. A usabilidade de uma API define o seu nível de documentação necessário para o seu uso, sua simplicidade e otimização para reusabilidade. APIs difíceis de utilizar, com baixa usabilidade, reduzem a produtividade do programador e impactam diretamente na qualidade do produto final [Henning 2009].

Ao se tratar de APIs para *web services*, existem três *stakeholders*: o desenvolvedor, o consumidor e o usuário final [Yamamoto et al. 2018]. O desenvolvedor é responsável por implementar a API e os seus serviços. O consumidor é o desenvolvedor da aplicação que utiliza a API em questão. O usuário final é a pessoa

que utiliza a aplicação desenvolvida. O uso de APIs pode ser difícil, resultando em erros e ineficiência. Isto posto, projetar APIs com os seus consumidores em mente pode resultar em menos erros, junto com maior eficiência, efetividade e segurança.

Uma API pode ser avaliada e mensurada de modo a melhorar a sua usabilidade. Existem diversos métodos, centrados na interação humano-computador, de como melhorar a usabilidade de APIs [Myers and Stylos 2016]. Uma possível abordagem para esta avaliação são as heurísticas de Nielsen, que descrevem dez propriedades que um desenvolvedor pode utilizar para checar qualquer *design* de maneira igual, sejam APIs ou interfaces de usuário. Dessa forma, é possível avaliar a usabilidade de APIs, o que inclui o quão simples é aprender o seu uso, a produtividade dos programadores utilizando-as, sua consistência e sua semelhança ao modelo mental do usuário. Com a usabilidade sendo considerada pelos desenvolvedores de APIs, o resultado são APIs mais fáceis de serem aprendidas e utilizadas corretamente.

2.4. Modelo de Qualidade ISO/IEC 25010

O modelo ISO/IEC 25010 é uma evolução do modelo ISO 9126, dividido em oito características, apresentadas na Figura 2 [Miguel, Mauricio and Rodríguez 2014]. Seu principal objetivo é fornecer um modelo de avaliação de qualidade para produtos de software, utilizando três perspectivas diferentes: usuários primários, secundários e indiretos, discutindo as características na visão de cada um.

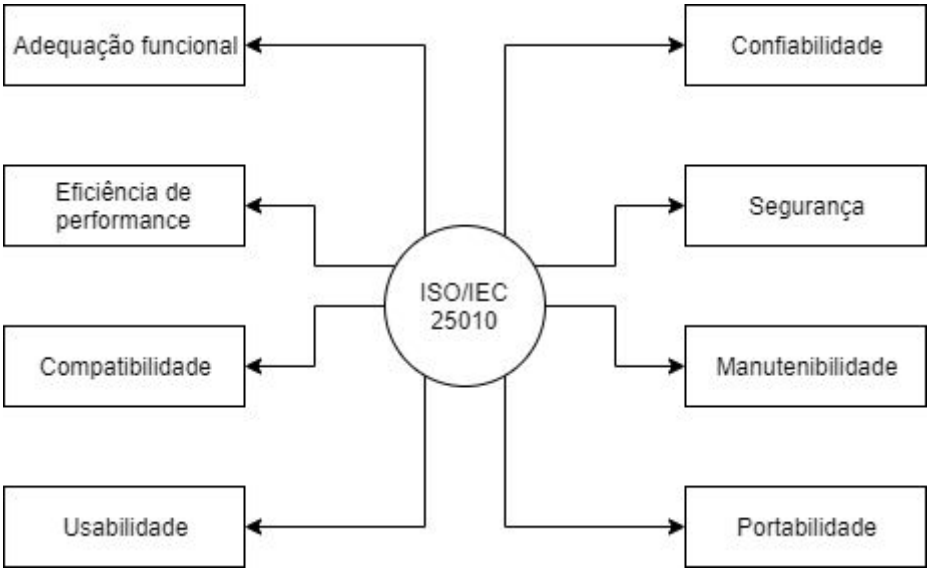

Figura 2. Características definidas pelo modelo ISO/IEC 25010

As características definidas pela norma são relevantes para todos os produtos de software e sistema computacionais, fornecendo uma terminologia para especificar, medir e avaliar a qualidade de sistemas de maneira consistente. Apesar do seu escopo ser projetado para produtos de software, muitas de suas características também são relevantes para sistemas maiores e serviços [ISO 25010, 2011]. Por exemplo, a eficiência de performance pode ser utilizada para tratar do desempenho de *web services* e a adequação funcional para verificar se uma biblioteca apresenta todos os requisitos

necessários para o seu uso.

3. Trabalhos Relacionados

Os trabalhos relacionados discutidos nesta seção envolvem a comparação das abordagens REST e SOAP e a investigação dos benefícios e experiências no uso de GraphQL. Além disso, também são apresentados trabalhos que estudam a usabilidade para a avaliação de APIs, para nortear a metodologia proposta.

Belqasmi et al. (2012) realizam uma comparação de desempenho entre REST e SOAP para aplicações do tipo multimídia. Para isso, utilizou-se a mesma arquitetura para ambos. Métricas de desempenho, como tempo de resposta em ambientes distribuídos, tempo de resposta na mesma máquina e tamanho de pacotes enviados e recebidos, foram analisadas. A aplicação baseada em REST, pelos resultados, gera menor tempo de resposta e menor *overhead* em termos de carga na rede.

Kumari e Rath (2015) também tem como objetivo avaliar os estilos SOAP e REST em termos de desempenho, tendo como base a integração com uma aplicação corporativa. Isto posto, foram implementadas APIs nas duas abordagens. São utilizadas duas métricas para avaliação, sendo elas, a taxa de vazão e tempo de resposta. Junto a isso, tem-se a simulação de usuários simultâneos com a ajuda do conceito *multithreading*, no qual cada usuário é equivalente a uma *thread*. Ao analisar as métricas de desempenho definidas, obtiveram como resultado que usar REST é mais vantajoso que usar SOAP.

Ao abordar GraphQL para a implementação de *web services*, o estudo de Vogel et al. (2018) relata experiências juntamente com uma discussão sobre a migração para GraphQL. Um requisito considerado para a migração foi a coexistência de camadas arquiteturais originais e interfaces REST juntamente com uma interface GraphQL adicional. Como resultado, identificaram que com a migração, o tempo de resposta do servidor GraphQL requer 46% menos tempo para consultas que envolvem múltiplas entidades distintas. Porém, *trade-offs* tiveram que ser feitos para a migração, diante de desafios que são causados por diferenças conceituais a nível de arquitetura e serviço.

Ainda tratando sobre migrações para GraphQL, o estudo de Brito et al. (2019) investiga os benefícios do uso da abordagem para implementar *web services*. Para isso, clientes da API REST do Github foram migrados para usar a mesma API em GraphQL, tendo como objetivo de analisar se a migração reduz o tamanho de documentos e o número de campos retornados. Como principais conclusões, tem-se que a tecnologia reduz o tamanho de documentos JSON, porém, não leva a uma redução do número de consultas realizadas por clientes da API.

O estudo de Seabra et al. (2019) trabalha a comparação de desempenho entre REST e GraphQL. Para isso, três aplicações são reescritas utilizando as duas abordagens. É observado que, para cargas de trabalhos menores (menos de 3.000 requisições), GraphQL apresenta um desempenho superior quanto a taxa de requisições e transferência de dados por segundo. REST, por outro lado, apresenta um desempenho superior para cargas de trabalho maiores. Para cargas de trabalho triviais (menos de 100 requisições), as abordagens apresentam desempenho similar.

Ao se tratar da avaliação da usabilidade de APIs, o estudo de Myers e Stylos

(2016) propõe um conjunto de diretrizes, baseadas nas heurísticas de Nielsen, para realizar esse tipo de avaliação, mostrando exemplos de como cada uma pode ser aplicada ao se considerar consumidores e usuários finais. O trabalho foi produto de mais de uma década de estudo sobre o assunto, com a contribuição de mais de 30 pesquisadores de uma universidade. Junto a isso, o estudo demonstra apoio em prol da inclusão da usabilidade como métrica de qualidade chave a ser utilizada por todas as APIs, de forma que a liberação de APIs que não forem avaliadas quanto à usabilidade seja inaceitável.

Dentro do contexto de avaliação da usabilidade de APIs, o estudo de Yamamoto et al. (2018) identificou a ausência de um modelo de qualidade de APIs Web ao levar em consideração o fato de que elas são desenvolvidas, mantidas e executadas remotamente e independentes dos seus consumidores. Para resolver isto, foi proposto um modelo de qualidade de APIs baseado na ISO 25010, tendo como foco os consumidores e levando em consideração duas características: facilidade de aprendizado e estabilidade. A primeira lida com a usabilidade da API pelo desenvolvedor da aplicação, e a estabilidade é o grau com que a interface da API sofre alterações que requer que os consumidores alterem o seu código.

4. Materiais e Métodos

Ao realizar a integração de um aplicativo para dispositivos móveis com um *web service*, os desenvolvedores da aplicação possuem a responsabilidade de tomar decisões arquiteturais, realizar a integração por meio da codificação, manter o funcionamento adequado da aplicação e também a experiência do usuário. Esses três momentos podem ser quantificados em três tempos, sendo eles, tempo de *design*, tempo de implementação e tempo de execução, respectivamente. Nesta seção, são apresentados os critérios e métodos da avaliação de cada abordagem de implementação para *web services*, em cada um dos três tempos.

4.1 Ambiente Experimental

A avaliação das abordagens em tempo de implementação e execução envolve o desenvolvimento de uma aplicação para Android. Realiza-se o experimento em um ambiente controlado com infraestrutura minimalista. Um único servidor de *backend* fornece os *web services* implementados em REST, SOAP e GraphQL, e a aplicação é executada em um *smartphone*.

A regra de negócio implementada na aplicação é um serviço de lista de professores e os detalhes de cada professor. Todos os dados são armazenados em memória. Esta lógica é exposta através de três abordagens: uma SOAP/XML, outra REST/JSON e a última GraphQL/JSON. Para o REST e o GraphQL, utiliza-se o NodeJS como servidor, e para o *web service* em SOAP, utiliza-se o ASP.NET Core 2.2. Para o servidor REST, utiliza-se o framework Express. Para o servidor GraphQL, utiliza-se o framework graphql-yoga. Para a criação do servidor SOAP, utiliza-se o ASP.NET Core 2.2. O servidor utilizado nas três abordagens (REST, SOAP, GraphQL) possui as seguintes configurações:

- Ambiente: máquina virtual na DigitalOcean
- Localização: New York - USA

- Sistema Operacional: Ubuntu 18.04 x64
- Processador: *single core*
- Memória RAM: 1 GB
- Disco: SSD 25GB

Para o consumo dos serviços no aplicativo Android, utiliza-se bibliotecas distintas. Para o REST, utiliza-se o Retrofit, a biblioteca mais utilizada para o consumo de serviços REST no contexto de aplicações Android [Belkhir et al. 2019]. Na abordagem SOAP, utiliza-se a biblioteca KSoap 2, a única biblioteca encontrada para suporte ao SOAP no Android que continua com o desenvolvimento ativo. Para o GraphQL, utiliza-se a biblioteca Apollo Client, a única biblioteca para o consumo de serviços em GraphQL no Android. O *smartphone* Android utilizado para os testes possui as seguintes especificações:

- Modelo: Xiaomi Mi A2
- Sistema Operacional: Android 9, patch de 05/08/2019
- Memória RAM: 4 GB
- Processador: Qualcomm Snapdragon 660, octa-core (4x2.2GHz + 4x1.8GHz)

Durante os testes em tempo de execução, utiliza-se três tipos de conexão no celular: 3G, 4G e Wi-Fi (*link* de 50mbps).

4.2. Procedimentos de Avaliação em Tempo de *Design*

Nesta seção, são apresentados os critérios e a metodologia para a avaliação do tempo de *design*, tendo como base as heurísticas de Nielsen.

4.2.1. Critérios de Avaliação

O tempo de *design* ocorre no momento de definição da abordagem de implementação de *web service* a ser utilizada. Os desenvolvedores de aplicações para dispositivos móveis são os consumidores da API implementada pelo *web service* escolhido. Dessa forma, neste momento, deve-se avaliar a usabilidade da API ao considerar cada tipo de abordagem de implementação de *web service*, de forma que os desenvolvedores de aplicações para dispositivos móveis consigam realizar e manter a integração do *web service* com o aplicativo de forma mais eficiente, efetiva e segura. Para realizar a avaliação, utiliza-se como base as heurísticas de Nielsen, uma das formas de avaliar usabilidade de APIs [Myers and Stylos 2016]. Estas diretrizes levam em consideração a usabilidade percebida pelos consumidores de APIs e são apresentadas abaixo:

- Visibilidade do Status do Sistema: a verificação do estado da API e a incompatibilidade diante de uma operação devem fornecer um feedback apropriado. Nesta heurística, é avaliado se a abordagem, no retorno da requisição, utiliza corretamente os códigos de erros HTTP.
- Compatibilidade entre o sistema e o mundo real: o idioma, os padrões e as convenções da API devem corresponder no máximo possível ao mundo do consumidor.
- Controle e liberdade para o usuário: usuários de API devem abortar ou redefinir operações e serem capazes de retornar facilmente ao estado normal da API.

- Consistência de padronização: toda decisão de *design* de API tomada estabelece uma regra em que os consumidores aprenderão.
- Prevenção de erros: a API deve orientar o consumidor a usá-la de forma correta.
- Reconhecimento ao invés de memorização: os nomes de parâmetros de serviços providos pela API devem ser claros e compreensíveis pelos consumidores, evitando confusões e erros.
- Eficiência e flexibilidade de uso: os consumidores devem ser capazes de realizar suas tarefas com a API de forma eficiente e flexível.
- Projeto estético e minimalista: APIs organizadas e menos complexas são mais utilizáveis.
- Ajuda os usuários a reconhecerem, diagnosticarem e recuperarem-se de erros: uma API deve incluir informações de erro humanizadas, oferecendo texto legível e conselhos para solucionar problemas.
- Ajuda e documentação: a documentação da API é interface principal pela qual o consumidor interage ao utilizá-la, dessa forma, devem ser realizadas adequadamente.

4.2.2. Metodologia da Avaliação

Para a avaliação de cada abordagem em tempo de *design*, utiliza-se as heurísticas de Nielsen para verificar se a abordagem em questão fornece características nativas que podem se adequar a cada heurística. Isto posto, define-se uma característica para cada heurística, cujo propósito é demonstrar se a abordagem a viola ou não, justificando o motivo em caso positivo. As características são apresentadas na Tabela 1.

Além disso, ao avaliar a usabilidade, considera-se o quanto a API é comum à realidade do usuário. REST é a abordagem mais utilizada no contexto de dispositivos móveis, portanto reflete a realidade da maioria dos desenvolvedores de aplicações para dispositivos móveis [Belkhir et al. 2019].

A aplicação da metodologia é realizada pelos autores. Ambos são estudantes do oitavo período do curso de Engenharia de Software da PUC Minas, e cursaram a disciplina de Interação Humano-Computador, que apresenta as heurísticas de Nielsen e outras avaliações de usabilidade. Além disso, um dos autores atua em uma empresa de Belo Horizonte como Desenvolvedor Android.

Tabela 1. Características qualitativas para avaliação do tempo de *design*

Heurística	**Característica**
Visibilidade do Status do Sistema	A abordagem utiliza os códigos HTTP nas respostas das requisições?
Compatibilidade entre o sistema e o mundo real	A abordagem segue os padrões da abordagem mais utilizada (REST) pelos desenvolvedores de aplicações para dispositivos móveis?
Controle e liberdade para o usuário	A abordagem permite realizar a limitação do número de consultas à API quando necessário?
Consistência de padronização	A abordagem permite o consumidor da API definir as regras de negócio?

Prevenção de erros	A abordagem possui algum mecanismo para evitar erros providos por dados incorretos nas requisições?
Reconhecimento ao invés de memorização	A abordagem permite o consumidor da API definir os nomes dos dados dos serviços?
Eficiência e flexibilidade de uso	A abordagem permite o consumidor da API definir os serviços e os dados dos serviços?
Estética de *design* minimalista	A abordagem utiliza o JSON para a troca de dados?
Ajuda os usuários a reconhecerem, diagnosticarem e recuperarem-se de erros	A abordagem possui um padrão para a definição de erros?
Ajuda e documentação	A abordagem possui um mecanismo de geração de documentação automática?

4.3. Procedimentos de Avaliação em Tempo de Implementação

Nesta seção, são apresentados os critérios e a metodologia para a avaliação do tempo de implementação, com base nas características do modelo de qualidade ISO 25010.

4.3.1. Critérios de Avaliação

O tempo de implementação é realizado na etapa de codificação para a integração do aplicativo com o *web service*. Diante disso, tem-se a preparação do projeto para realizar as integrações, para requisições e recebimento de respostas, conforme regras de negócio do projeto. Para a avaliação do impacto de escolha de cada abordagem na integração do aplicativo com o *web service*, utiliza-se as seguintes características do modelo de qualidade ISO 25010:

- Adequação funcional: representa o grau em que um produto ou sistema fornece funções que atendem as necessidades do projeto. Nesse caso, é avaliada a biblioteca utilizada para a integração do aplicativo com o *web service*.
- Manutenibilidade: representa o grau de eficácia e eficiência com que um produto ou sistema pode ser modificado para melhorá-lo, corrigi-lo ou adaptá-lo às mudanças no ambiente e nos requisitos. Nesse caso, tem-se como foco a possibilidade de modificabilidade no código do aplicativo ao invés do código do *web service* quando se tem mudanças no ambiente e nos requisitos.

As outras características definidas pela ISO 25010 não podem ser facilmente adaptadas para o contexto de *web services*, pois a norma é usada primariamente para a avaliação de produtos finais de software. Por isso, são utilizadas apenas as duas características acima.

4.3.2. Metodologia da Avaliação

Para a realização da avaliação, desenvolve-se um aplicativo Android com o objetivo de realizar requisições e receber respostas de cada um dos *web services*, de acordo com as especificações apresentadas no Ambiente Experimental. Para isso, são geradas versões do aplicativo integrado com cada um dos *web services*, tendo como apoio uma biblioteca, de acordo com a abordagem.

O modelo de qualidade ISO 25010 fornece as características para a avaliação da metodologia, enquanto a criação das características de implementação é de responsabilidade do avaliador. A partir do código fonte e do desenvolvimento da aplicação, as características de implementação apresentadas na Tabela 2, criadas a partir das características da ISO 25010, são respondidas utilizando respostas de "Sim" ou "Não". A aplicação da metodologia neste caso também é realizada pelos autores. Ambos cursaram a disciplina de Qualidade de Software, que trata das normas de qualidade como a ISO 25010.

Tabela 2. Características qualitativas para a avaliação do tempo de implementação

Característica	**Característica de implementação**	**Avaliação**
Adequação funcional	A abordagem possui uma biblioteca com documentação necessária para integrá-la e utilizá-la no projeto?	Realizada através da análise da documentação oficial disponíveis nas páginas web oficiais e no repositório do GitHub de cada biblioteca em comparação com a documentação necessária para sua integração e utilização.
	A abordagem possui uma biblioteca que assegura que o mapeamento dos dados do *web service* estão corretos?	Realizada durante a integração da aplicação com o *web service*, analisando a capacidade da biblioteca de criar classes de dados e mapear esses dados automaticamente, garantindo a integridade dos dados.
	A abordagem possui uma biblioteca que não necessita da criação manual de arquivos e diretórios para a sua inicialização?	Realizada durante a integração da aplicação com a biblioteca, analisando a necessidade de criar diretórios e arquivos para que ela possa funcionar adequadamente.
	A abordagem possui uma biblioteca atualizada?	Realizada através do repositório do GitHub de cada biblioteca, onde é analisado: frequência de commits, tempo desde o último commit, quantidade de stars e taxa de issues fechadas.
Manutenibilidade	A abordagem permite o consumidor do serviço definir os dados que serão retornados?	Realizada através da análise de cada *web service* e sua capacidade de definir quais serviços e dados serão utilizados deste *web service* no código do aplicativo.

4.4. Procedimentos de Avaliação em Tempo de Execução

Nesta seção, são apresentados os critérios e a metodologia para a avaliação do tempo de

execução, tendo como base características do modelo de qualidade ISO 25010.

4.4.1. Critérios de Avaliação

O tempo de execução tem como foco o aplicativo em produção, dessa forma, é realizada uma análise de desempenho do aplicativo integrado com o *web service*. Por isso, observa-se as variações em diferentes tipos de conexões, como 3G, 4G e Wi-Fi, contextos comuns quando se trata de dispositivos móveis. Para avaliação do impacto de cada abordagem na integração do aplicativo com o *web service*, utiliza-se a característica eficiência de desempenho do modelo de qualidade ISO 25010.

A eficiência de desempenho representa a capacidade do sistema ao atender requisitos não funcionais em relação aos recursos disponíveis. Nesse caso, tem-se como foco a execução do aplicativo após a integração com o *web service*, analisando métricas de **tempo de resposta, uso de dados e uso de bateria**.

4.4.2. Metodologia da Avaliação

As métricas para eficiência de desempenho são medidas de forma quantitativa, utilizando como base as avaliações propostas em um *benchmark* criado para comparar a performance entre *web services* SOAP e REST, identificados a partir de uma revisão da literatura [Sungkur and Daiboo 2015]. Para a avaliação de cada abordagem em tempo de execução, utilizam-se os mesmos aplicativos do tempo de implementação. A Tabela 3 resume o experimento de tempo de execução.

Tabela 3. Detalhamento do experimento de tempo de execução

Fatores a serem testados	**Descrição**
Aplicações (requisições a serem realizadas). Cada uma é implementada nas abordagens apresentadas.	*GET /teachers*: retorna a lista de professores; *GET /teachers/ID*: retorna um professor específico, com mais detalhes
Número de execuções	Cada requisição é realizada cinco vezes, e para o cálculo da métrica, será utilizada a média entre esses cinco valores
Tipo de conexão	São testados os três tipos mais comuns de conexão: 3G, 4G e Wi-Fi (velocidade de *50mbps*).
Abordagens de implementação	REST, SOAP e GraphQL
Métricas avaliadas	Tempo de resposta; uso de dados; uso de bateria

Para realizar as medições do tempo de resposta e do tamanho de pacote, utiliza-se o *Network Profiler* da *integrated development environment* (IDE) Android Studio. O *Network Profiler* exibe atividade da rede em tempo real em uma linha do tempo, mostrando os dados enviados e recebidos, bem como o número atual de conexões. Assim, é possível examinar como e quando o aplicativo transfere dados e otimizar o código subjacente em função disso [Google LLC 2019].

Para realizar as medições sobre o uso de bateria, utiliza-se a ferramenta nativa de uso de bateria presente no Android. Em cada uma das versões dos aplicativos integrados com cada *web service*, ocorre a execução por 30 minutos de requisições contínuas que solicitam uma lista de professores. Ao finalizar o tempo, é consultada a ferramenta dentro das configurações do celular, onde é registrada a porcentagem de bateria gasta pelo aplicativo durante o tempo de processamento das requisições.

5. Resultados

Nesta seção, primeiro, apresenta-se os resultados obtidos em tempo de *design*, contendo as respostas positivas ou negativas para cada característica relacionada às heurísticas de Nielsen. Após isso, tem-se os resultados obtidos em tempo de implementação, contendo as respostas positivas ou negativas para cada característica de implementação relacionada à adequação, uma característica presente na ISO 25010. Por fim, tem-se os resultados obtidos em tempo de execução, contendo gráficos para cada métrica relacionada a desempenho, uma característica também presente na ISO 25010.

5.1 Tempo de *Design*

A Tabela 4 apresenta o atendimento das características das heurísticas em cada uma das abordagens. Ao levar em conta as heurísticas, SOAP mostra-se como a pior escolha, enquanto REST e GraphQL empatam no número de heurísticas atendidas, porém, para heurísticas diferentes.

Tabela 4. Avaliação das características de tempo de *design*

Heurística	**Atende a heurística?**		
	REST	**GraphQL**	**SOAP**
Visibilidade do Status do Sistema	Sim	Não	Não
Compatibilidade entre o sistema e o mundo real	Sim	Não	Não
Controle e liberdade para o usuário	Sim	Não	Não
Consistência de padronização	Não	Sim	Não
Prevenção de erros	Não	Sim	Não
Reconhecimento ao invés de memorização	Não	Não	Não
Eficiência e flexibilidade de uso	Não	Sim	Não
Estética de *design* minimalista	Sim	Sim	Não
Ajude os usuários a reconhecerem, diagnosticarem e recuperarem-se de erros	Sim	Não	Não
Ajuda e documentação	Não	Sim	Não
TOTAL DE HEURÍSTICAS ATENDIDAS	**5/10**	**5/10**	**0/10**

5.2 Tempo de Implementação

Para a análise do tempo de implementação, é realizada a integração de cada *web service* com o aplicativo para Android. A partir disso, é analisada a seleção das bibliotecas escolhidas para a integração e a etapa de codificação, com o propósito de apresentar as características relevantes em termos de adequação funcional e manutenibilidade. A Tabela 5 apresenta as respostas de cada característica para a sua respectiva abordagem.

Tabela 5. Avaliação das características de tempo de implementação

Característica de implementação	**Abordagem**		
	REST	**GraphQL**	**SOAP**
A abordagem possui uma biblioteca com documentação necessária para integrá-la e utilizá-la no projeto?	Sim	Sim	Não
A abordagem possui uma biblioteca que assegura que o mapeamento dos dados do *web service* estão corretos?	Não	Sim	Não
A abordagem possui uma biblioteca que não necessita da criação manual de arquivos e diretórios para a sua inicialização?	Sim	Não	Sim
A abordagem possui uma biblioteca atualizada?	Sim	Sim	Sim
TOTAL DE CARACTERÍSTICAS ATENDIDAS	**3/4**	**3/4**	**2/4**

A partir do resultado na Tabela 5 é possível perceber que todas as abordagens possuem bibliotecas atualizadas para o desenvolvimento. Porém, REST e SOAP não possuem um sistema de validação dos dados recebidos dos serviços, e GraphQL necessita da criação manual de arquivos e diretórios. Além disso, a biblioteca KSOAP2 não possui documentação necessária para integrá-la e utilizá-la no projeto.

5.3. Tempo de Execução

Para a análise do tempo de execução, é executado um *benchmark* com o objetivo de medir a eficiência de desempenho de cada abordagem, utilizando métricas de tempo de resposta, uso de bateria e uso de dados. Os resultados para essas métricas são apresentados a seguir.

5.3.1. Tempo de Resposta

As Figuras 3 e 4 apresentam a métrica de tempo de resposta, o que mostra o desempenho de cada uma das abordagens nos dois serviços utilizados e com cada tipo de conexão. É possível perceber que REST fica em primeiro lugar neste quesito, com GraphQL se aproximando em segundo. SOAP, em contrapartida, apresenta um tempo de resposta médio consideravelmente maior. Todas as abordagens sofrem uma perda de performance em conexões mais lentas (3G e 4G), e apresentam maior variação no 3G.

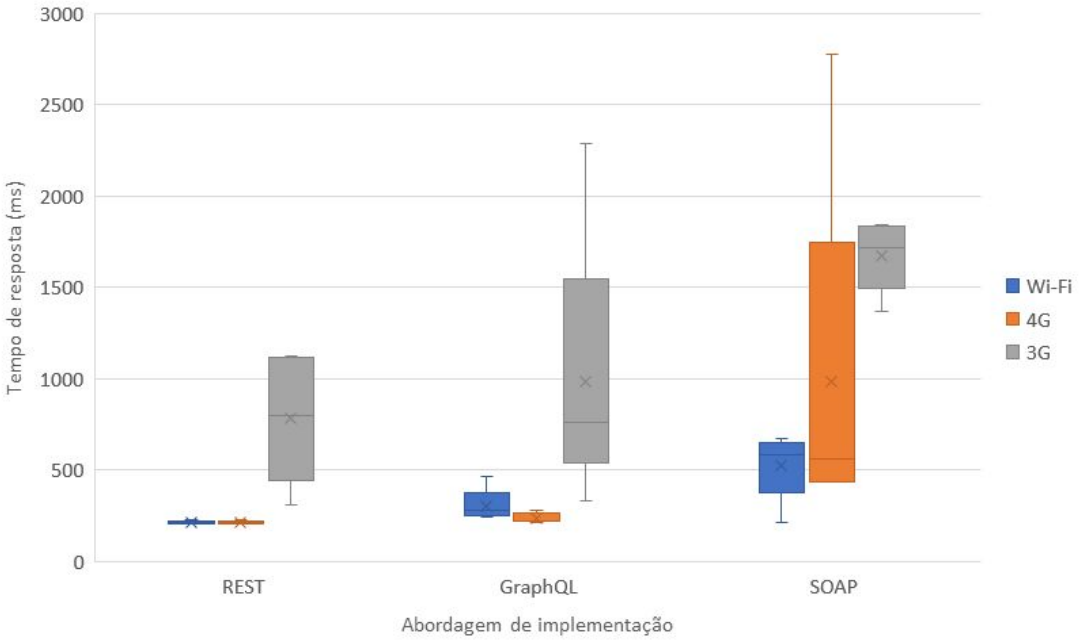

Figura 3. Tempo de resposta da aplicação *GET /teachers*

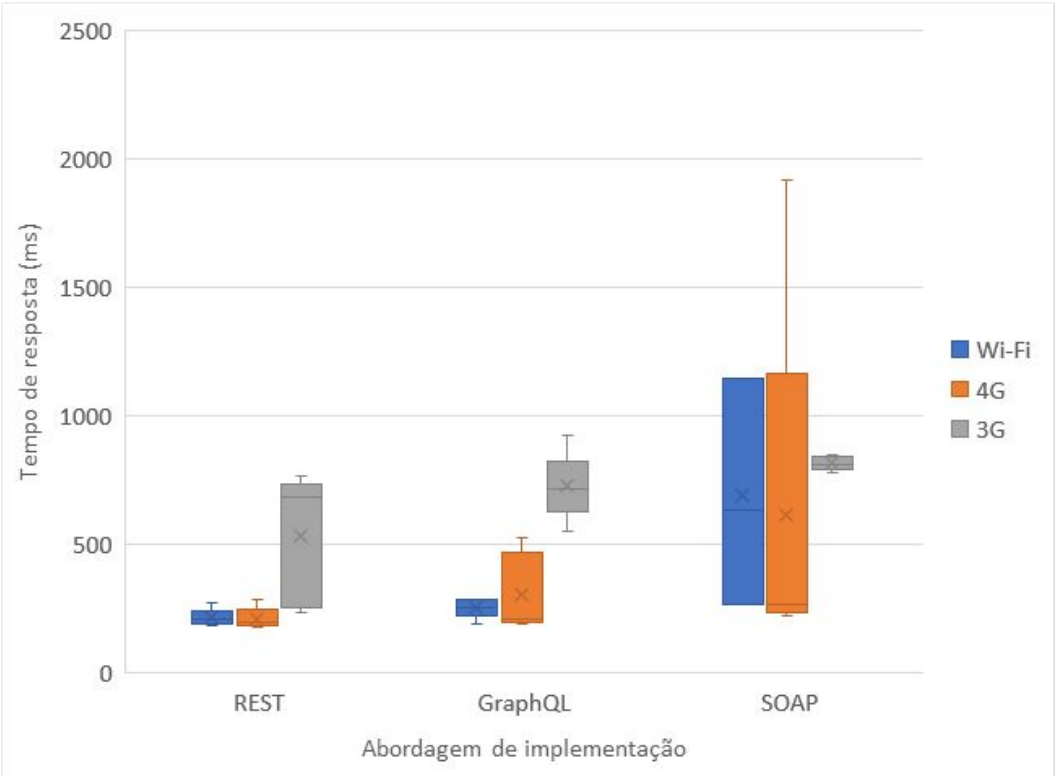

Figura 4. Tempo de resposta da aplicação *GET /teachers/ID*

5.3.2. Uso de Dados

A Figura 5 apresenta a métrica de uso de dados, mostrando o tamanho da pacote recebido pela aplicação. Quanto maior este tamanho, maior é o uso de dados pelo

aparelho, o que pode resultar em tarifas de dados móveis maiores, quando falamos de conexões 3G ou 4G. É possível perceber que a abordagem SOAP é a pior abordagem levando em consideração o uso de dados, enquanto REST e GraphQL apresentam pouca diferença quanto ao uso de dados.

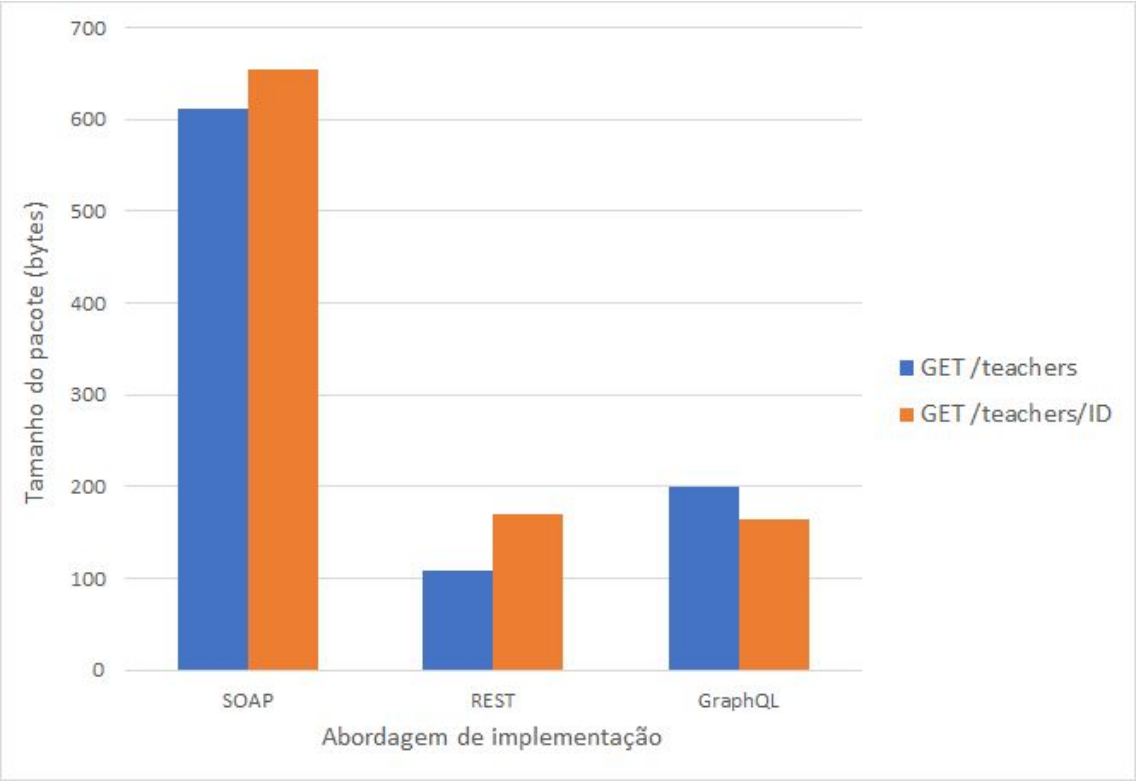

Figura 5. Uso de dados de cada requisição

5.3.3. Uso de Bateria

A Tabela 6 apresenta a métrica de uso de bateria, mostrando o desempenho de cada uma das abordagens nos dois serviços utilizados e com cada tipo de conexão, considerando uso de CPU e de Internet. Os resultados do experimento demonstram que, quanto ao uso de bateria, GraphQL é a melhor escolha, seguido pelo REST e então SOAP, ao se tratar da execução de requisições continuamente.

Tabela 6. Uso de bateria para cada abordagem

Abordagem	**Uso estimado de bateria do dispositivo**
SOAP	7% em 30 minutos
REST	6% em 30 minutos
GraphQL	4% em 30 minutos

6. Conclusões e Trabalhos Futuros

Este estudo se propôs a investigar o custo-benefício do uso de GraphQL como alternativa às abordagens de implementação de *web services* SOAP e REST, no contexto de aplicações para dispositivos móveis. Para comparar as diferentes abordagens, foram realizadas três avaliações, em tempo de *design*, implementação e execução, de forma a identificar as vantagens e desvantagens de cada abordagem. Para a avaliação em tempo de *design*, foram utilizadas as heurísticas de Nielsen, com o objetivo de avaliar a usabilidade de APIs, de cada abordagem. Para o tempo de implementação e execução, foi utilizado o modelo de qualidade ISO 25010, onde são criadas características de implementação para avaliação. Um aplicativo para Android foi desenvolvido para a avaliação em tempo de implementação e execução.

A partir dos resultados obtidos, é possível concluir 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. Em tempo de execução, REST e GraphQL se comportam de maneira similar, onde REST mostrou uma pequena vantagem no tempo de resposta, mas GraphQL possui a vantagem quanto ao uso de bateria.

Ao comparar os resultados com os trabalhos relacionados, quanto ao tempo de execução, são mostradas diversas similaridades. SOAP continua sendo a opção inadequada ao lidar com desempenho se comparada com REST. GraphQL apresenta o desempenho similar ao comparado com REST. Ao lidar com mensagens de retorno maiores em requisições, GraphQL apresenta uma vantagem significativa quanto ao uso de dados. Quanto aos tempos de *design* e implementação, não é possível realizar nenhuma comparação, visto que não existe uma comparação das abordagens neste momento.

Em relação aos trabalhos futuros, partindo dos resultados da avaliação das abordagens de *web service* tratadas neste estudo, sugere-se realizar uma pesquisa qualitativa para avaliar o custo-benefício do GraphQL em comparação a REST e SOAP no contexto de aplicações *web*. Dentro desse contexto, tem-se outros parâmetros para o tempo de implementação, devido ao fato de se utilizar tecnologias diferentes para a implementação. Além disso, os acessos são na maior parte das vezes realizados através de computadores, tendo como opções a conexão cabeada e o Wi-Fi. Desse modo, as mesmas abordagens podem ser comparadas, porém, em um contexto significativamente diferente.

Referências Bibliográficas

Alaa, G. and Hussein, D. (2009). *A Roadmap For Integrating Web 2.0 Tools Into IS Teacing & Research In A Developing Country Contect: The Egyptian Perspective*.

MCIS 2009 Proceedings.

AlShahwan F. and Moessner K., *Providing SOAP Web Services and RESTful Web Services from Mobile Hosts*. 2010 Fifth International Conference on Internet and Web Applications and Services, Barcelona, 2010, pp. 174-179.

Arazy, O., Kumar, N. and Shapira, B. (2010). *A Theory-Driven Design Framework for Social Recommender Systems*. Journal of the Association for Information Systems, v. 11, n. 9, p. 455–490.

Badidi, E. (2011). *A framework for brokered Service Level agreements in SOA environments*. 2011 7th International Conference on Next Generation Web Services Practices, Salamanca, Spain. IEEE.

Belkhir, A., Abdellatif, M., Tighilt, R., et al. (2019). *An Observational Study on the State of REST API Uses in Android Mobile Applications*. 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft).

Belqasmi, F., Singh, J., Bani Melhem, S. Y. and Glitho, R. H. (2012). *SOAP-based vs. RESTful web services: A case study for multimedia conferencing*. IEEE Internet Computing, v. 16, n. 4, p. 54–63.

Brito, G., Mombach, T. and Valente, M. T. (2019). *Migrating to GraphQL: A Practical Assessment*. IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), Hangzhou, China, p. 140–150.

Chalmers, D. and Sloman, M. (2009). *A survey of quality of service in mobile computing environments*. IEEE Communications Surveys & Tutorials, v. 2, n. 2, p. 2–10.

Fielding, R. T. and Taylor, R. N. (2002). *Principled design of the modern Web architecture*. ACM Transactions on Internet Technology, v. 2, n. 2, p.115–150.

Google LLC (2019). Android Studio User Guide. https://developer.android.com/studio/profile/network-profiler. Acesso em 9/05/2019.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H., Karmarkar, A. and Lafon, Y. . (2007). *SOAP Version 1.2 Part 1: Messaging Framework (2nd Ed) [Online]*. http://www.w3.org/TR/soap12-part1/. Acesso em 27/05/2019.

Hartig, O. and Pérez, J. (2017). *An initial analysis of facebook's GraphQL language*. 11th Alberto Mendelzon International Workshop on Foundation of Databases and the Web (AMW), Montevideo, Uruguay, p. 1–10.

Hartig, O. and Pérez, J. (2018). *Semantics and Complexity of GraphQL*. WWW '18 Proceedings of the 2018 World Wide Web Conference, Lyon, France, p. 1155–1164.

M. Henning, "API design matters," Commun. ACM, vol. 52, no. 5, pp. 46–56, May 2009

ISO/IEC 25010:2011, Systems and Software Engineering —Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality

Models, 2011. Disponível em: https://www.iso.org/standard/35733.html

Kumari, S. and Rath, S. K. (2015). *Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration*. 2015 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2015, p. 1656–1660.

Miguel, J., Mauricio, D. e Rodríguez, G. (2014). *A Review of Software Quality Models for the Evaluation of Software Products*. International Journal of Software Engineering & Applications, v. 5, n. 6, p. 31–53.

Myers, B. A. and Stylos, J. (2016). *Improving API usability*. Communications of the ACM, v. 59, n. 6, p. 62–69.

Piccioni, M., Furia, C. A. e Meyer, B. (2013). *An Empirical Study of API Usability*. In 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, MD, USA, pp. 5–14.

Rajan, S., Malini, A. and Sundarakantham, K. (2014). *Performance evaluation of online mobile application using Test My App*. 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, Ramanathapuram, India, n. 978, p. 1148–1152.

Seabra, M., Nazário, M. F. and Pinto, G. (2019). *REST or GraphQL? A Performance Comparative Study*. In Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse - SBCARS '19, Salvador, Brazil, p. 123-132.

Seda, P., Masek, P., Sedova, J., Seda, M., Krejci J. and Hosek, J. (2018). *Efficient Architecture Design for Software as a Service in Cloud Environments*. 2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Moscow, Russia, p. 317–322.

Serrano, N., Hernantes, J., and Gallardo, G. (2014). *Service-Oriented Architecture and Legacy Systems*. IEEE Software, vol. 31, no. 5, pp. 15-19.

Shelby, Z. (2010). *Embedded web services*. (2010). IEEE Wireless Communications, vol. 17, n. 6, p. 52-57.

Sungkur, R. K. and Daiboo, S. (2015). *SOREST, A Novel Framework Combining SOAP and REST for Implementing Web Services.* The Second International Conference on Data Mining, Internet Computing, and Big Data, Reduit, Mauritius, p. 22–34.

Vogel, M., Weber, S. and Zirpins, C. (2018). *Experiences on Migrating RESTful Web Services to GraphQL*. 15th International Conference on Service-Oriented Computing, ICSOC 2017, held in Málaga, Spain, v. 10797, p. 283–295.

Yamamoto, R., Ohashi, K., Fukuyori, M., et al. (2018). *A Quality Model and Its Quantitative Evaluation Method for Web APIs*. 2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE.