UNIVERSIDADE DO SUL DE SANTA CATARINA

NILTON AUGUSTO SOARES NETO

GRAPHQL UMA ABORDAGEM ALTERNATIVA AO PADRÃO REST

Florianópolis

2020

NILTON AUGUSTO SOARES NETO

GRAPHQL UMA ABORDAGEM ALTERNATIVA AO PADRÃO REST

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação da Universidade do Sul de Santa Catarina como requisito parcial à obtenção do título de bacharel em Sistemas de Informação.

Orientador: Prof. Flávio Ceci, Dr.

Florianópolis

2020

NILTON AUGUSTO SOARES NETO

GRAPHQL UMA ABORDAGEM ALTERNATIVA AO PADRÃO REST

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de bacharel em Sistemas de Informação e aprovado em sua forma final pelo Curso de Sistemas de informação da Universidade do Sul de Santa Catarina.

Florianópolis, 10 de dezembro de 2020.

_____

Professor e orientador Flávio Ceci, Dr.
Universidade do Sul de Santa Catarina

_____

Prof. Richard Henrique de Souza, Ms.
Universidade...

_____

Prof. Alexandre Vitoreti de Oliveira, Ms.
Universidade do Sul de Santa Catarina

AGRADECIMENTOS

Agradeço aos meus pais por acreditarem no meu potencial e proporcionar que eu pudesse ter feito uma boa faculdade. Sei do esforço que foi feito para que eu pudesse continuar minha graduação e agradeço demais por tudo, sem vocês, nada disso seria possível. Acredito que o esforço valeu a pena.

Agradeço também a minha irmã por sempre me incentivar a continuar e por ser minha inspiração em muitas coisas, sei que sem ela eu não teria conseguido chegar até aqui e devo muito a ela pela preocupação que ela teve comigo durante minha graduação.

Gostaria também de agradecer a uma pessoa que já não está mais entre nós. Meu tio avô Acácio Carlos Vieira, uma das pessoas que mais importantes da minha vida e que me inspirou a ser a pessoa e o profissional que me tornei, com ele aprendi valores que vou levar para a vida toda. Sem ele eu provavelmente não teria iniciado minha graduação e não estaria onde estou. Devo muito a ele, tenho saudades.

Gostaria de agradecer também aos meus avós pelos ensinamentos e valores que aprendi, sempre me incentivaram e acreditaram no meu potencial. Com certeza tiveram grande influência positiva na minha vida.

Agradeço também aos excelentes professores que tive na minha trajetória acadêmica, em especial ao Prof. Dr. e meu orientador Flavio Ceci, que teve muita paciência e empatia para me orientar nesse trabalho que me desafiou por meses.

RESUMO

Criar um sistema de grande complexidade tem sido cada vez mais desafiador. Diversos problemas relacionados a gargalos de recursos e demora no processamento de dados constantemente assombram a vida de desenvolvedores. O Facebook também passava por esse tipo de problema com seu aplicativo mobile. Foi quando em 2012, a empresa decidiu criar o GraphQL, com a ideia de diminuir o tamanho e tempo de resposta das requisições. Com isso criaram uma nova tecnologia com potencial de mudar todo um mercado de desenvolvimento. Em 2016, liberaram publicamente uma versão estável para ambientes de produção. Para a apresentação desta monografia foi criado duas estruturas de APIs, uma REST e uma GraphQL. Ambas conectadas em um único banco de dados não relacional MongoDB e integradas a uma única SPA que será a interface visual do projeto.

Palavras-chave: GraphQL. REST. MongoDB.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O mundo do desenvolvimento de software vem evoluindo com muita velocidade e em grande escala, trazendo diversos novos conceitos, maneiras de se pensar, de implementar, novas tecnologias, padrões e entre outros.

Os desafios para se escalar um produto ou projeto vem ganhando complexidade com o passar do tempo, criando necessidades de mercado cada vez mais inteligentes para resolver os problemas que aparecem com o crescimento do software.

Uma das características de um software escalável é mantê-lo disponível mesmo com um aumento de tráfego de acessos, assim confirmado por Bondi (2017, p. 1, tradução nossa) "Escalabilidade é a capacidade de um sistema acomodar um número crescente de trabalho ou ser suscetível a ampliação."

Outra característica muito importante relacionada à escalabilidade de um software é a manutenção. Entende-se que um software onde a manutenção não gera grande impacto, de tempo e custo, é considerado um software escalável. Assim confirmado por Durães (2020) em seu artigo sobre escalabilidade.

> A manutenção do software impacta diretamente nos custos operacionais da empresa, seja em recursos técnicos ou humanos. Portanto, ele deve ser desenhado para receber atualizações necessárias de forma fácil e linear, adaptando-se a novas regras e processos.

As soluções precisam, de modo geral, atender à necessidade e resolver o problema apresentado, devendo ser, de certa forma, simples para quem for desenvolver ou fazer manutenção posteriormente, para que possa ter uma fácil leitura do que está acontecendo, dando ao programador agilidade e eficiência na resolução de problemas.

Este trabalho vai trazer especificamente uma tecnologia que vem crescendo cada vez mais no mercado de desenvolvimento e que está apresentando excelentes resultados de performance e satisfação por quem está utilizando, o GraphQL.

Hanashiro (2018, p. 9) define GraphQL da seguinte forma "Podemos dizer que o GraphQL é uma linguagem de consulta e manipulação, assim como o SQL. Enquanto o SQL[1] é um padrão para fazer consultas em bancos de dados, o GraphQL nos dá a possibilidade de indicar a uma API[2] os dados que queremos".

---

[1] SQL é uma linguagem padrão para trabalhar com bancos de dados relacionais

[2] API é um conjunto de rotinas e padrões de programação para acesso a um software web

Isso dá uma grande flexibilidade na hora de fazer consultas a uma API, garantindo que não vamos ter um overfetch [3]de informações, que segundo Porcello (2018, p. 13) esse fenômeno é o excesso de dados retornados por uma API sem necessidade, como acontece normalmente nas APIs REST[4] comuns.

1.1 PROBLEMÁTICA

Conforme Porcello (2018, p. 10) o GraphQL começou a ser desenvolvido em 2012 pela empresa Facebook, com o intuito de resolver uma das suas necessidades, a performance de seus aplicativos mobile, Android e IOS, que sofriam com o overfetch de informação e apresentavam falhas com muita frequência.

Por exemplo, na Figura 1, há um overfetch no resultado de uma requisição em uma API REST. Sendo que no exemplo apresentado na Figura 1, as únicas informações necessárias são o nome, altura e peso. Em uma API REST, quando solicitamos as informações do personagem, retorna todo esse conjunto enorme de informações, sendo a maior parte delas indesejadas.

Figura 1 – Resposta de requisição de uma API REST


```
"name": "Luke Skywalker",
"height": "172",
"mass": "77",
"hair_color": "blond",
"skin_color": "fair",
"eye_color": "blue",
"birth_year": "19BBY",
"gender": "male",
"homeworld": "https://swapi.co/api/planets/1/",
"films": [
  "https://swapi.co/api/films/2/",
  "https://swapi.co/api/films/6/",
  "https://swapi.co/api/films/3/",
  "https://swapi.co/api/films/1/",
  "https://swapi.co/api/films/7/"
],
"species": [
  "https://swapi.co/api/species/1/"
],
"vehicles": [
  "https://swapi.co/api/vehicles/14/",
  "https://swapi.co/api/vehicles/30/"
],
"starships": [
  "https://swapi.co/api/starships/12/",
  "https://swapi.co/api/starships/22/"
],
```


Fonte: Eve Porcello (2018).

[3] Overfetch segundo Porcello (2018, p.13) é o excesso de dados retornado por uma API REST

[4] A sigla REST significa Transferência Representacional de Estado, em português

Com o GraphQL, ele permite solicitar, de forma fácil, apenas as três informações que são necessárias, sem desperdiçar tempo e recurso, trazendo informações irrelevantes para quem está fazendo a consulta, otimizando assim o custo operacional de uma chamada. Segue um exemplo desse comportamento na Figura 2.

Figura 2 – Consulta e resposta GraphQL

Fonte: Eve Porcello (2018).

Com o crescimento dos projetos e a necessidade de expandir a quantidade de usuários utilizando sistemas que usam API REST, foi visto que este recurso estava se tendo problemas com a demanda de utilização, assim confirmado por Porcello (2018, p. 10).

> Os aplicativos para IOS e Android da empresa eram somente wrappers finos em torno das views do site móvel. O Facebook tinha um servidor RESTful e tabelas de dados FQL (versão de SQL do Facebook). Havia problemas de desempenho e os aplicativos, com frequência, apresentavam falhas.

O lançamento oficial do GraphQL para ambiente de produção foi feito em 2016, confirmado por Porcello (2018, p. 10) “Em setembro de 2016, o GraphQL estava oficialmente pronto para o ambiente de produção”.

A partir desse momento, diversas empresas começaram a adotar o conceito do GraphQL como forma de substituir os projetos implementados em RESTful.

Dentre as empresas que utilizam GraphQL em toda busca de dados estão IBM, Intuit, Airbnb e entre outras.

1.2 OBJETIVO GERAL

Apresentar o conceito e aplicação do GraphQL como uma alternativa ao padrão RESTful.

1.3 OBJETIVO ESPECÍFICO

Os objetivos específicos desta monografia podem ser encontrados abaixo.

- Identificar as principais características de soluções baseadas em RESTful e GraphQL;
- Apresentar um experimento utilizando ambas arquiteturas;
- Avaliar a partir de um questionário o uso do GraphQL;
- Formular as conclusões sobre o seu uso.

1.4 JUSTIFICATIVA

As API's REST estão começando a demonstrar gargalos em determinados cenários que necessitavam de maior uso de recurso computacional, assim confirmado por Porcello (2018, p. 11), "[...] à medida que a web evoluiu, o REST mostrou sinais de esgotamento em certas condições. O GraphQL foi criado para reduzir esses sinais".

Alguns dos exemplos dessas condições de esgotamento citados por Porcello (2018) são o overfetching e o underFetching de informações.

Onde o overfetching é a busca de dados em excesso e o underfetching é a necessidade de fazer diversas requisições para a API e obter um conjunto de informações.

Com isso, será apresentado como o GraphQL resolve esse tipo de problema e torna mais simples buscar e manipular dados de uma API.

1.5 ESTRUTURA DA MONOGRAFIA

Este trabalho tem os seguintes capítulos que buscam atingir os objetivos propostos anteriormente:

- **Capítulo 1 – Introdução**. Este capítulo apresenta o tema proposto, trazendo informações que contextualizam a pesquisa, a problemática, os objetivos, diretos e indiretos, a justificativa do trabalho e a estrutura da monografia.

- **Capítulo 2 – Revisão Bibliográfic**a. Este capítulo apresenta as principais teorias da literatura à pesquisa, com o objetivo de apresentar a base necessária para o desenvolvimento deste trabalho acadêmico.
- **Capítulo 3 – Metodologia**. Este capítulo descreve a metodologia escolhida para o desenvolvimento deste trabalho acadêmico.
- **Capítulo 4 – Modelagem**. Este capítulo apresenta os requisitos funcionais e não funcionais do protótipo, os diagramas de componente, diagrama de entidades e por fim, os protótipos de tela.
- **Capítulo 5 – Desenvolvimento**. Este capítulo descreve os itens relacionados ao desenvolvimento do protótipo, cita as tecnologias e ferramentas utilizadas, fala sobre o histórico do desenvolvimento, apresenta a versão final das telas do sistema e apresenta um questionário em relação ao GraphQL.
- **Capítulo 6 – Conclusão**. Este capítulo traz as conclusões feitas sobre o GraphQL e os trabalhos futuros gerados através dessa monografia.

2 REVISÃO BIBLIOGRÁFICIA

Nesta revisão da literatura, visando aprofundar o conhecimento sobre o assunto na introdução desse trabalho, serão abordados a arquitetura de aplicações web, a arquitetura REST, o conceito de GraphQL e o ambiente de execução Javascript NodeJs.

2.1 ARQUITETURA DE UMA APLICAÇÃO WEB

A arquitetura de uma aplicação web é um conjunto de itens necessários para que um usuário tenha acesso a um conteúdo que esteja online na internet. Esse conjunto inclui um navegador, conexão de internet e um servidor web. Esse conjunto trabalha da seguinte forma: "Navegadores fazem requisição de uma página web para um servidor. Cada página é um conjunto de instruções expressadas em HTML. Algumas páginas incluem scripts que rodam do lado do cliente e são interpretados pelo navegador" (Conallen,1999, p.4, tradução nossa).

Os scripts citados por Conallen são codificados em javascript, a linguagem universal dos maiores navegadores de mercado. Esses scripts tem a função de tornar uma página web dinâmica conforme o usuário interage com a página através de botões, caixas de pesquisa, links que encaminham para outras páginas, entre outros tipos de interações.

Hoje, essa arquitetura está mais moderna, com outras preocupações além de apenas transmitir uma página web de um servidor para um cliente. Essa afirmação é confirmada pela citação de Conallen (1999, p. 4, tradução nossa).

> Os servidores da Web de hoje melhoraram esse design básico. Os servidores da Web atuais têm muito mais consciência da segurança e incluem recursos como gerenciamento do estado do cliente no servidor, integração de processamento de transações, administração remota e pool de recursos, para citar apenas alguns.

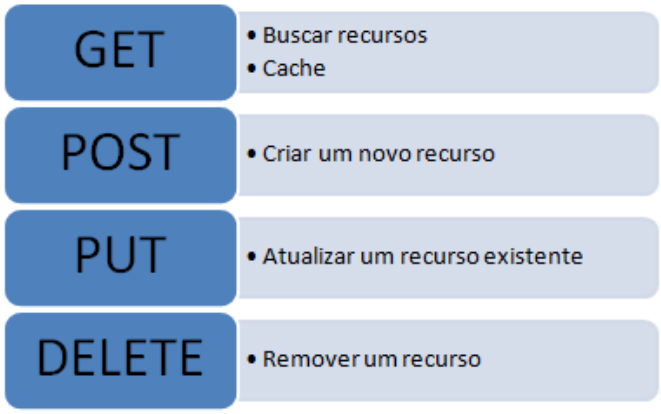
2.2 REST

O segundo tema a ser abordado nesta revisão bibliográfica é o REST, *Representational State Transfer,* um tema muito atual no mundo do desenvolvimento de aplicações que necessitam transitar dados entre elas.

Porcello (2018, p. 12) conta como o REST nasceu no ano de 2000, na tese de doutorado de Roy Fielding, na Universidade da Califórnia. Em sua tese, ele descreve o conceito como uma arquitetura orientada a recursos, nos quais os usuários tem a possibilidade de acessar tais

recursos através de operações pré-definidas, sendo elas, GET, PUT, POST e DELETE, cada uma dessas operações com sua função, de acordo com a Figura 3.

Figura 3 – Verbos HTTP e suas utilizações



Fonte: Rafael Godinho (2009).

De acordo com Schulthess (2017, apud Junior, 2019, p. 10) "Até o ano de 2000, não existiam padrões em como design de API deveria ser. A integração necessária, uso de protocolos, como SOAP, que era notoriamente conhecido como algo complexo a ser construído, manusear e debugar".

Apenas para contextualizar sobre o SOAP, antecessor ao REST, ele surgiu no fim dos anos 90 e foi criado pela Microsoft com o objetivo de codificar uma mensagem em XML e utilizar HTTP como transporte. Porcello (2018, p.12) também conta um pouco sobre o SOAP em seu livro.

> No final dos anos 1990, o SOAP (Simple Object Access Protocol, ou Protocolo Simples de Acesso a Objetos) surgiu na Microsoft. O SOAP usava XML para codificar uma mensagem e HTTP como transporte. Também usava um sistema de tipos e introduziu o conceito de chamadas orientadas a recursos para dados. O SOAP oferecia resultados razoavelmente previsíveis, mas causava frustação porque suas implementações eram muito complicadas.

2.2.1 Cacheamento

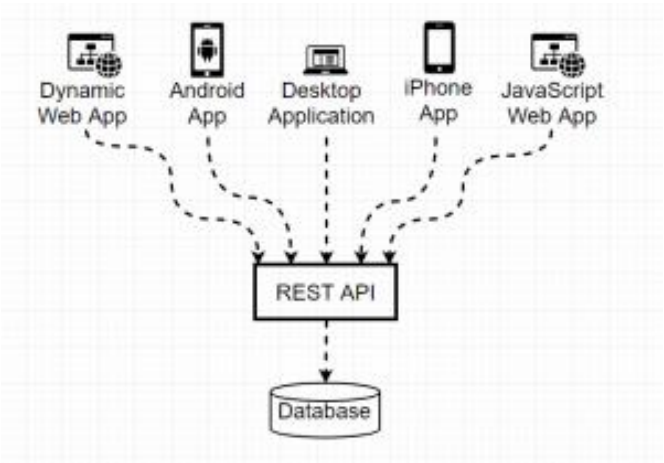
Como na Internet, os clientes podem armazenar as respostas em cache. As respostas devem, portanto, implícita ou explicitamente, definir-se como armazenáveis em cache, ou não, para impedir que os clientes reutilizem dados obsoletos ou inadequados em resposta a solicitações adicionais. O cache bem gerenciado elimina parcial ou completamente algumas interações cliente-servidor, melhorando ainda mais a escalabilidade e o desempenho (Fredrich, 2018, tradução nossa).

2.2.2 Client-Server REST

Esse tópico trata-se de umas das constraints da arquitetura REST. Uma constraint nada mais é que um conjunto de regras de estilos a serem seguidos e aplicados para homologar uma arquitetura como REST.

Fielding (2000, apud Sousa, p.7) conta que quando separamos a interface do usuário do armazenamento de dados da aplicação, melhoramos a portabilidade da interface, o que ajuda a tornar o sistema escalável, além de tornar o projeto mais simples para manutenções e melhorias de forma independente.

Figura 4– Diagrama REST



Fonte: Vissarion Yfantis (2019).

A Figura 4 mostra plataformas de interface distintas que podem interagir com uma mesma API RESTful para solicitar e enviar dados. Com isso, segundo Sousa (2017, p8), dessa forma é possível evoluir funcionalidades de maneira independente, permitindo que o servidor de armazenamento e processamento de dados apenas utilize seus recursos para processamento de informação e troca de informação, sem se preocupar com a interface.

2.3 GRAPHQL

Neste tópico será apresentado a revisão teórica do principal tema desse trabalho de conclusão de curso, o GraphQL.

Para Seabra (2019, p. 4) a definição de GraphQL é a seguinte “GraphQL é uma linguagem de consulta para APIs. A linguagem fornece uma descrição do modelo de dados, oferecendo aos clientes a capacidade de solicitar exatamente os dados que precisam.”.

A literatura de Porcello (2018, p.8) complementa a definição anterior com a seguinte frase. “É também um runtime para atender a consultas com seus dados. O serviço GraphQL é independente do meio de transporte, mas em geral, é servido sobre HTTP”.

2.3.1 Consulta GraphQL

Uma consulta GraphQL é um pouco diferente para quem está acostumado com REST, ela se torna um pouco mais verbosa, porém mais objetiva.

A consulta é realizada através de um schema, que para Hanashiro (2018, p.15) tem a seguinte definição: “Um esquema (schema) é uma representação de um objeto. Como usamos o GraphQL para fazer consultas de dados, ele precisa conhecer a estrutura desses dados para saber quais campos estão disponíveis. Por isso, precisamos de um modo de declarar esses esquemas”.

Hanashiro (2018, p.15) também complementa a ideia de o GraphQL ser agnóstico a linguagem.

> Como uma das ideias bases do GraphQL é funcionar com qualquer linguagem de programação e framework, foi criada uma linguagem própria de declaração de esquemas, a Schema Definition Language (Linguagem de Definição de Esquema) ou, simplesmente, SDL. Assim podemos declarar nossos esquemas de forma única para qualquer linguagem.

Na Figura 5, temos um exemplo no lado esquerdo de uma como é montada uma estrutura de consulta com GraphQL, e a direita, a resposta fornecida pela API GraphQL para a consulta construída.

Figura 5– Consulta e resposta GraphQL

Fonte: Eve Porcello (2018).

2.3.2 Manipulando dados em GraphQL

As modificações, ou mutações, de dados com GraphQL são semelhantes a fazer uma consulta, porém precisam de alguns parâmetros diferentes para que seja entendido como uma modificação.

Hanashiro (2018, p. 22) complementa “As mutations nos permitem indicar ao back-end que queremos fazer alguma modificação nos dados presentes em nosso banco de dados ou criar algo novo”.

Figura 6 – Mutação de criação de dado em GraphQL

```
mutation{
    createAluno(nomeCompleto: "Akira Hanashiro", idade: 25){
        id
    }
}
```

Fonte: Akira Hanashiro (2018).

2.3.3 Subscriptions com GraphQL

O termo subscription adotado no GraphQL é equivalente ao termo WebSocket ou Polling em APIs REST, sendo é a terceira operação adicionada pelo Facebook ao GraphQL.

Esse tipo de operação é utilizado quando precisamos de dados atualizados em tempo real, sem precisar fazer uma atualização programada na tela do usuário.

Porcello (2018, p.49) conta um pouco sobre o porquê a subscription precisou ser implementada. “As subscriptions no GraphQL surgiram em virtude de um caso de uso real do Facebook. A equipe queria ter uma forma de mostrar informações em tempo real sobre o número de curtidas (Live Likes) que uma postagem estava obtendo sem ter que atualizar a página”.

2.4 NODE JS

Neto (2017) entende a definição de Node JS da seguinte forma “A definição mais apropriada seria: um ambiente de runtime para Javascript que roda em cima de uma engine conhecida como Google v8”.

Neto (2017), também conta que o nascimento do NODE JS se deu pela necessidade de Ryan Dahl em resolver um problema relacionado a acompanhar o progresso de upload de arquivos, sem que ele precisasse criar um polling no sistema.

Na afirmação de Huffine (2020, tradução nossa)

> Polling é uma técnica em que verificamos novos dados em um determinado intervalo, fazendo periodicamente solicitações de API para um servidor. Por exemplo, podemos usar uma Poll se houver dados que sejam alterados com frequência ou precisarmos aguardar a transição do servidor para um determinado estado. O Polling é uma alternativa simples para soquetes da web ou eventos do servidor.

2.4.1 Google V8

O Google V8 é o motor utilizado no Chrome, criado pela Google, sendo nesse motor que o NodeJS trabalha.

Segundo Neto (2017) "O V8 é uma engine (motor) criada pela Google para ser usada no browser chrome. Em 2008 a Google tornou o V8 open source e passou a chamá-lo de Chromium project."

Essa mudança para torná-lo open source foi importante para a comunidade, pois possibilitou que desenvolvedores entendessem o como funcionava por baixo dos panos, permitindo compreender como o javascript era interpretado e compilado dentro do V8.

Segundo NETO (2017) "O V8 compila o código para linguagem de máquina, além de otimizar drasticamente a execução usando heurísticas, permitindo que a execução seja feita em cima do código compilado e não interpretado". Isso acaba fazendo com que o javascript que está sendo executado no navegador tenha uma performance excelente.

2.5 MONGODB

MongoDB é um banco de dados não relacional orientado a documentos, criado pela empresa 10gen em 2007.

Medeiros (2014) define esse tipo de banco da seguinte forma

> Banco de Dados Orientados a Documentos tem como característica conter todas as informações importantes em um único documento, ser livre de esquemas, possuir identificadores únicos universais (UUID), possibilitar a consulta de documentos através de métodos avançados de agrupamento e filtragem (MapReduce) e também permitir redundância e inconsistência.

Medeiros (2014) também fala das características do MongoDB, por se tratar de um software de código aberto, GNU AGPL versão 3.0, alta performance, não possuir esquemas fixos, ser desenvolvido na linguagem C++ e ser multiplataforma. Detalhe que Medeiros (2014) também conta é que, apesar do projeto ter início em 2007, ele foi se tornar um projeto de código aberto apenas em 2009.

Duarte (2017) conta que o MongoDB é utilizado em organizações de todos os tamanhos para criar aplicações que necessitem de baixa latência, alta vazão de informação e alta disponibilidade. Ele também comenta que o armazenamento das informações é feito em documentos em vez de tabelas e colunas.

Duarte (2017) afirma que aproximadamente 20% dos desenvolvedores que utilizam banco não relacional também usam o MongoDB. Este é um dado que mostra uma popularidade destacada dentre os bancos não relacionais e que indica ter uma comunidade maior para dar suporte.

3 METODOLOGIA

Este capítulo trata sobre a caracterização da pesquisa, a abordagem do problema apresentado, a caracterização dos objetivos, assim como as etapas metodológicas e as delimitações da monografia.

3.1 CARACTERIZAÇÃO DO TIPO DE PESQUISA

A pesquisa tratada na monografia é do tipo pesquisa aplicada. Segundo afirmação de Silva (2005, p.20) “Pesquisa Aplicada: objetiva gerar conhecimentos para aplicação prática e dirigidos à solução de problemas específicos. Envolve verdades e interesses locais”.

A abordagem do problema será feita através de uma abordagem qualitativa. Será aplicado um questionário com objetivo de avaliar a etapa de desenvolvimento de ambas estruturas e avaliar a execução da aplicação.

Em seu livro, Silva (2005, p.20) apresenta o que é uma pesquisa qualitativa.

> Pesquisa Qualitativa: considera que há uma relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito que não pode ser traduzido em números. A interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa. Não requer o uso de métodos e técnicas estatísticas.

Quanto aos objetivos desta monografia, será utilizado o método de pesquisa exploratória. Confirme Gil (1991, Apud Silva, 2005, p.21), a pesquisa exploratória “visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses.”

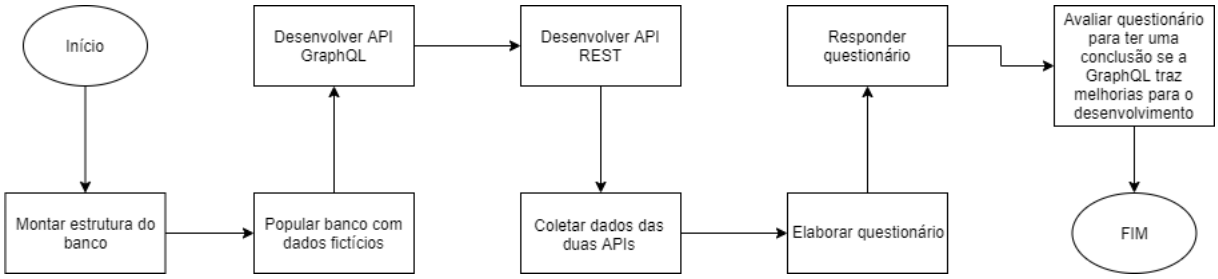
3.2 ATIVIDADES METODOLÓGICAS

Nesta monografia, serão construídas duas APIs de demonstração para análise dos dados que ajudara na resposta do questionário, uma delas será uma API REST convencional, e a outra, uma API utilizando a teoria do GraphQL.

Ambas APIs serão desenvolvidas utilizando o Node.Js e vão compartilhar um único banco de dados MongoDB.

O objetivo é coletar os dados das chamadas HTTP das duas APIs e posteriormente utilizar essa amostra de dados para gerar informação que possa ser usada para responder o questionário de avaliação das estruturas desenvolvidas. O código construído também será ser analisado e avaliado no questionário.

Figura 7 – Atividades para desenvolvimento



Fonte: Autoria própria.

Após a finalização das atividades propostas na figura 7, é feita uma avaliação para avaliar se o GraphQL traz, ou não, benefícios para o desenvolvimento de sistemas web.

3.3 DELIMITAÇÕES

Esta monografia não se compromete a trabalhar os seguintes itens:

- Segurança de acesso as APIs;
- Implementar otimizações no banco de dados;
- Disponibilizar um link público de acesso as APIs;
- Qualquer tipo de teste automatizado nas APIs;
- Utilizar design pattern ou arquitetura de software específica;
- Acesso mobile.

4 MODELAGEM

Esse capítulo tem como objetivo, apresentar a modelagem do sistema de demonstração. Segundo Sommerville (2011, p.82) “Modelagem de sistema é o processo de desenvolvimento de modelos abstratos de um sistema, em que cada modelo apresenta uma visão ou perspectiva, diferente do sistema’.

As seguintes seções do capítulo são, requisitos funcionais, requisitos não funcionais, diagrama de componente e por último, o diagrama de entidades.

4.1 REQUISITOS FUNCIONAIS

Pelo entendimento de Sommerville (2011, p.59) “Os requisitos funcionais de um sistema descrevem o que ele deve fazer. Eles dependem do tipo de software a ser desenvolvido, de quem são seus possíveis usuários e da abordagem geral adotada pela organização[...]”.

Figura 8 – Requisitos funcionais do sistema

Requisitos funcionais	
Número	Requisito
RF001	O sistema deve permitir o cadastro de uma pessoa.
RF002	O sistema deve permitir cadastro e alteração do endereço de uma pessoa apenas no cadastro/edição da pessoa.

Fonte: Autoria própria.

4.2 REQUISITOS NÃO FUNCIONAIS

Segundo Sommerville (2011, p.59) os requisitos não funcionais não estão diretamente relacionados as funcionalidades do sistema, mas sim a outras propriedades do sistema como, confiabilidade, tempo de resposta e infraestrutura.

Figura 9 – Requisitos não funcionais do sistema

Requisitos NÃO funcionais	
Número	Requisito
RNF001	O sistema deve estar disponível através de duas APIs, sendo uma delas GraphQL e outra REST.
RNF002	Ambas APIs devem acessar o mesmo banco de dados para consultar, criar e atualizar.
RNF003	O sistema deve estar disponível na última versão dos navegadores Chrome, Firefox e Edge.

Fonte: Autoria própria.

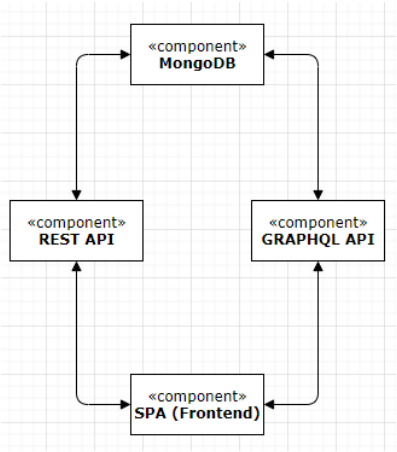
4.3 DIAGRAMA DE COMPONENTES

BOOCH (2012, p. 313) "[...]o diagrama de componentes mostra a estrutura e os caminhos potenciais das mensagens do componente. O diagrama de componentes em si não mostra a sequência das mensagens por meio dos componentes."

Ele vai permitir uma melhor visualização de como o sistema de demonstração irá utilizar duas APIs totalmente diferentes funcionando em conjunto em uma única aplicação.

No exemplo da Figura 10, temos a visualização da comunicação entre os componentes. Onde as APIs GraphQL e REST, estão se comunicando com o mesmo banco de dados e a SPA está se comunicando com as duas APIs para obter e enviar dados.

Figura 10 – Diagrama de componentes

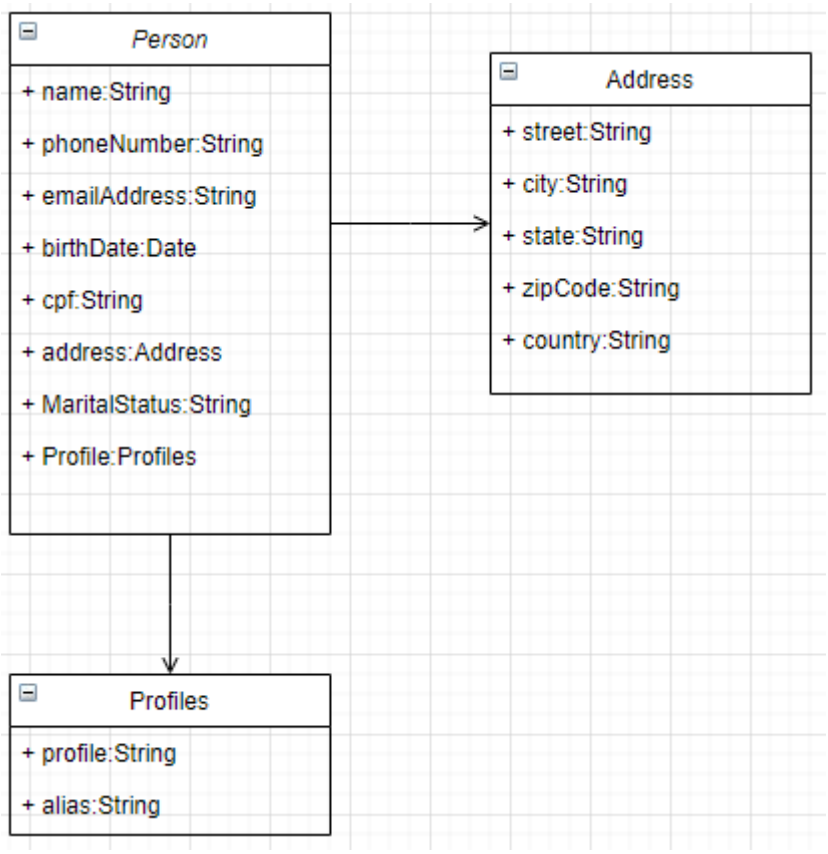


Fonte: Autoria própria.

4.4 DIAGRAMA DE ENTIDADES

BOOCH (2012, p. 313) entende que o diagrama de entidades tem o seu foco apenas nos dados dos objetos, diferente do digrama de classes que vai um pouco além e permite a modelagem dos comportamentos de cada entidade.

Figura 11 – Diagrama de entidades



Fonte: Autoria própria.

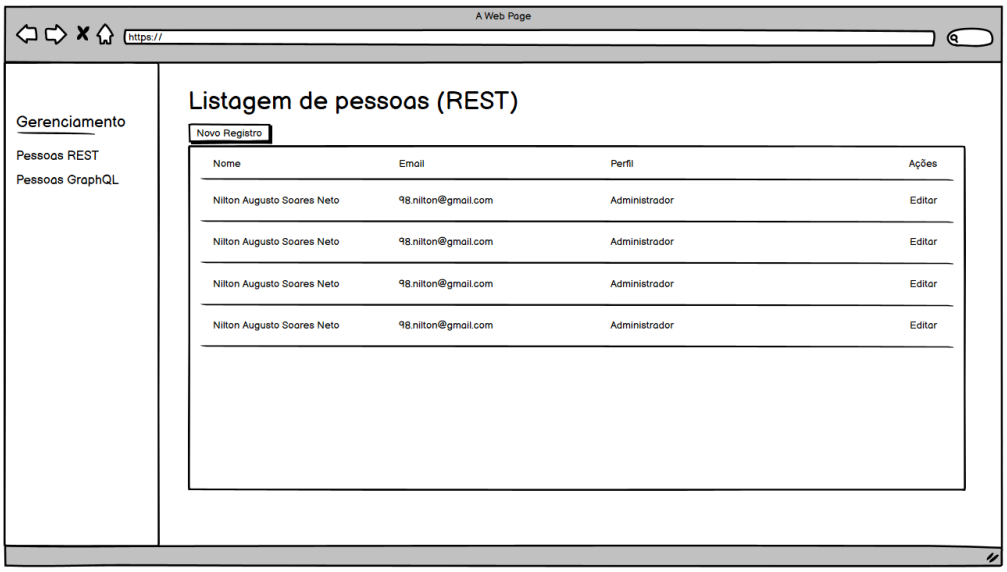
As entidades modeladas na figura 11, permitem uma melhor visualização dos efeitos negativos do overfetching na aplicação, como em uma listagem, onde não são necessários todos esses dados para representar um usuário único.

4.5 PROTÓTIPOS DE TELA

Nessa seção, será apresentado os protótipos das principais telas do sistema, formando o layout básico do sistema.

Para Sommerville (2011, p. 30), “Um protótipo é uma versão inicial de um sistema de software, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções”.

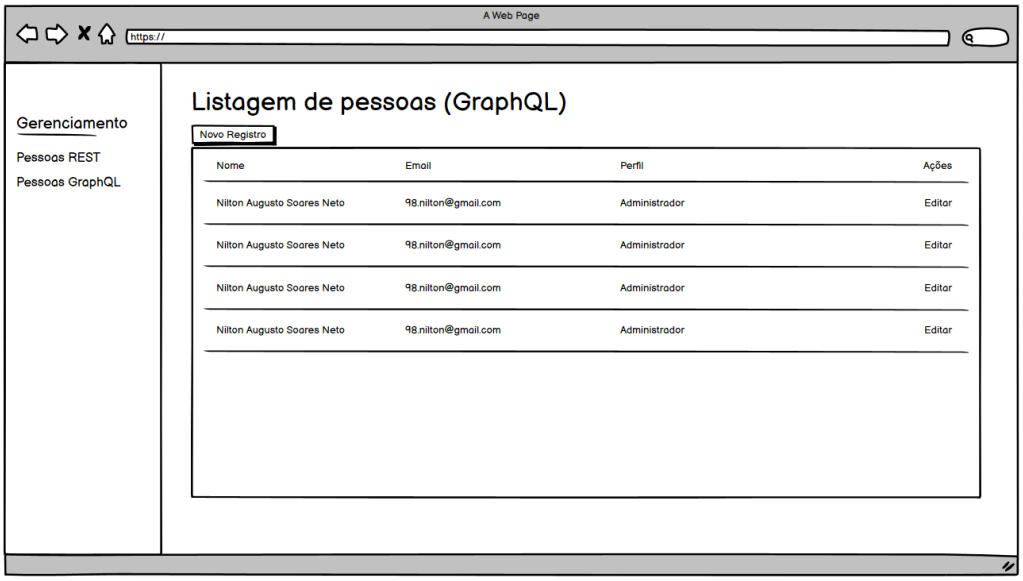
Figura 12 – Listagem de pessoas (REST)



Fonte: Autoria própria.

Na tela apresentada na figura 12, será listado os resultados da busca de pessoas através da API REST, e ao clicar em editar, o usuário será levado a uma tela de gerenciamento para aquela pessoa selecionada, e ao clicar em novo registro, o usuário será levado a uma tela de cadastro em branco para preencher com as informações de uma nova pessoa.

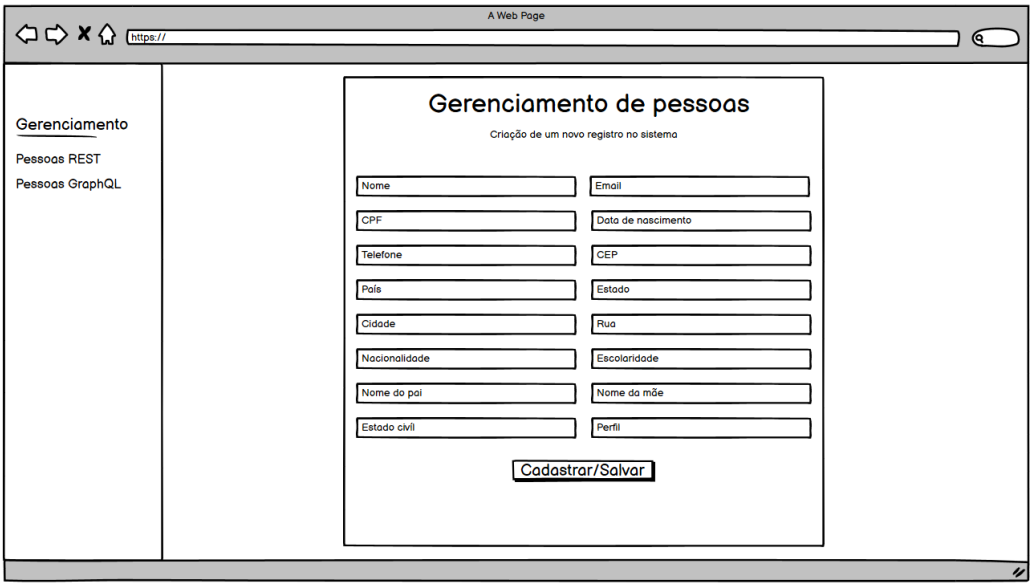
Figura 13 – Listagem de pessoas (GraphQL)



Fonte: Autoria própria.

Na tela apresentada na figura 13, será listado os resultados da busca de pessoas através da API GraphQL, e ao clicar em editar, o usuário será levado a uma tela de gerenciamento para aquela pessoa selecionada, e ao clicar em novo registro, o usuário será levado a uma tela de cadastro em branco para preencher com as informações de uma nova pessoa.

Figura 14 – Gerenciamento de pessoas



Fonte: Autoria própria.

Nessa tela apresentada na figura 14, o usuário pode chegar nessa tela por dois caminhos. Ao clicar em criar um novo registro com informações novas, ou editar uma pessoa que ele selecionou na listagem.

5 DESENVOLVIMENTO

Nesse capítulo são abordados as tecnologias, ferramentas e o questionário respondido da pesquisa. Assim é possível ter um melhor entendimento do que foi utilizado para construir os protótipos funcionais e os resultados da pesquisa irão auxiliar a embasar a conclusão dessa monografia.

5.1 TECNOLOGIAS E FERRAMENTAS

Nessa seção será apresentado as tecnologias e ferramentas escolhidas e utilizadas pelo autor para o desenvolvimento dos protótipos da pesquisa.

5.1.1 Express.js

Express é um framework para aplicativo web feito em Node.JS. A escolha da ferramenta para fazer parte da API REST, é por ser utilizado em larga escala pelo mercado e a disponibilidade de conteúdo para desenvolver.

5.1.2 Apollo-server/cliente

Atualmente é a biblioteca que melhor desempenha o papel de implementar a estrutura GraphQL e que permite desenvolver um sistema web completo utilizando javascript.

5.1.3 Vue.js

Vue.JS é um framework frontend javascript que permite criar componentes visuais reutilizáveis, visando uma manutenção mais fácil e com ganhos de performance por ser um framework SPA. A escolha pela ferramenta foi pela familiaridade do autor com a estrutura e construção de sistemas com essa tecnologia.

5.1.4 WebStorm

WebStorm é um IDE de desenvolvimento voltada ao conteúdo frontend. Criada pela empresa JetBrains, essa ferramenta é amplamente conhecida e utilizada no mercado web. Foi

escolhida por ter um suporte mais focado no desenvolvimento frontend, o que facilitou no desenvolvimento da SPA.

5.1.5 VsCode

VsCode é uma IDE de desenvolvimento de softwares que possuí suporte para dezenas de linguagens diferentes, com suporte para plugins diversos que auxiliam no desenvolvimento de um software. A escolha pela IDE foi pela facilidade e simplicidade da ferramenta para desenvolver o backend.

5.1.6 Git

Git é uma ferramenta de versionamento de código. Foi escolhida para poder auxiliar no desenvolvimento do projeto, pois caso necessitasse, poderia voltar em versões anteriores dos projetos.

5.2 HISTÓRICO DO DESENVOLVIMENTO

Durante o desenvolvimento dos protótipos, houve apenas um impeditivo em desenvolver as duas APIs e a SPA.

Um único problema foi encontrado durante o desenvolvimento da primeira API, relacionado à configuração do banco de dados MongoDB. Após tentar usar o banco em ambiente local utilizando Docker, um gerenciador de containers virtual,’ a API não conseguia estabelecer uma conexão, porém, quando utilizado uma interface gerenciadora de banco não relacional, a conexão funcionava como esperado.

A solução para o problema foi não utilizar o Docker para fazer o host do banco e sim utilizar um banco de dados em nuvem oferecido gratuitamente pelo site Mongo Atlas.

Sobre o tempo que foi necessário para construir cada uma das APIs, o tempo foi de um dia para cada uma das APIs e mais um dia construindo a SPA com a integração completa.

5.3 APRESENTAÇÃO DO SISTEMA

Nesta seção, será apresentado a implementação visual do sistema, em acordo com a estrutura proposta no capítulo 4.5.

Figura 15 – Implementação visual da listagem de pessoa na API REST

Fonte: Autoria própria.

Na figura 15, trata-se da tela de listagem de pessoas, onde as informações foram trazidas através de uma API REST. As opções disponíveis são: Editar um usuário já cadastrado, criar um novo usuário através da estrutura REST ou utilizar uma das opções do menu lateral esquerdo.

Figura 16 – Implementação visual da listagem de pessoa na API GraphQL

Fonte: Autoria própria.

Na figura 16, trata-se da mesma tela de listagem de pessoas, porém as informações foram trazidas através de uma API GraphQL. As opções disponíveis são: Editar um usuário já cadastrado, criar um novo usuário através da estrutura GraphQL ou utilizar uma das opções do menu lateral esquerdo.

Figura 17 – Implementação visual da tela de criação de uma pessoa

Fonte: Autoria própria.

Na figura 17, a tela de criação de um novo registro para uma pessoa é compartilhada visualmente com ambas APIs. A única diferença é qual API faz a chamada de criação. Após o cadastro completo ser preenchido, o usuário pode utilizar o botão salvar para persistir o dado no banco de dados da aplicação através da API que está sendo utilizada ou utilizar o menu lateral esquerdo para não persistir o dado alterado.

Figura 18 – Implementação visual da tela de edição de uma pessoa

Fonte: Autoria própria.

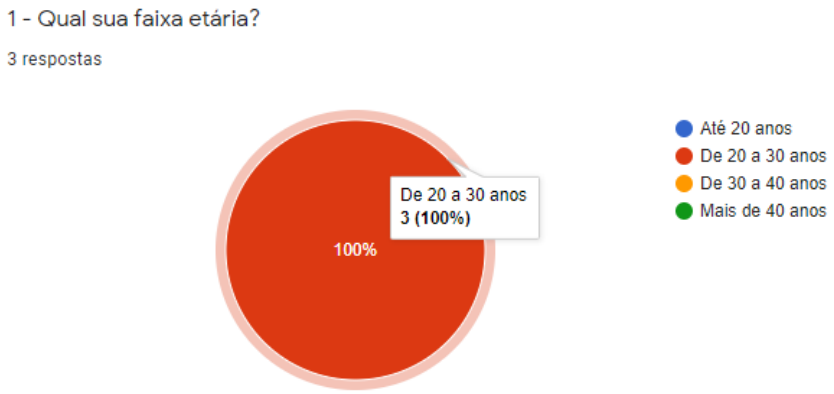
Na figura 18, a tela de edição de um registro já existente de uma pessoa é compartilhada visualmente com ambas APIs. A única diferença é qual API faz a chamada de edição do dado. Após o cadastro ser alterado, o usuário pode utilizar o botão salvar para persistir o dado no banco de dados da aplicação através da API que está sendo utilizada ou utilizar o menu lateral esquerdo para não persistir o dado alterado.

5.4 AVALIAÇÃO DA PROPOSTA

Para avaliar o conteúdo levantado nesta monografia e comprovar ou não os pontos citados anteriormente, foi desenvolvido um questionário de nove perguntas que buscam conhecer o público que respondeu ao questionário, e também ter a opinião técnica de cada um buscando sempre ligar algumas dessas perguntas aos objetivos específicos da pesquisa e a problemática.

As respostas foram submetidas por três desenvolvedores que possuem faixa etária próxima e níveis de conhecimento distintos.

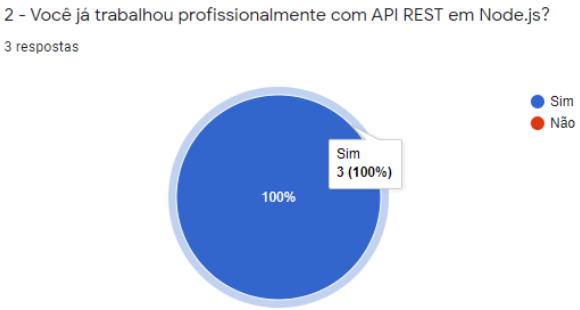
Figura 19 – Pergunta 1 do questionário



Fonte: Autoria própria.

A pergunta 1 do questionário, é sobre a faixa etária dos contribuintes da pesquisa. Nesse caso, os três possuem entre 20 e 30 anos.

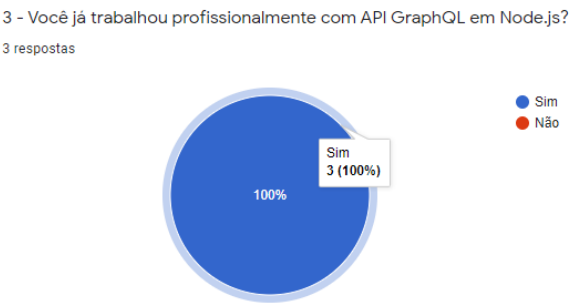
Figura 20 – Pergunta 2 do questionário



Fonte: Autoria própria.

A pergunta 2 busca saber se os contribuintes já tiveram experiência profissional com APIs REST em NODE.js. Essa informação é importante para a avaliação técnica das perguntas futuras. Neste caso, todos tiveram algum tipo de experiência profissional.

Figura 21 – Pergunta 3 do questionário



Fonte: Autoria própria.

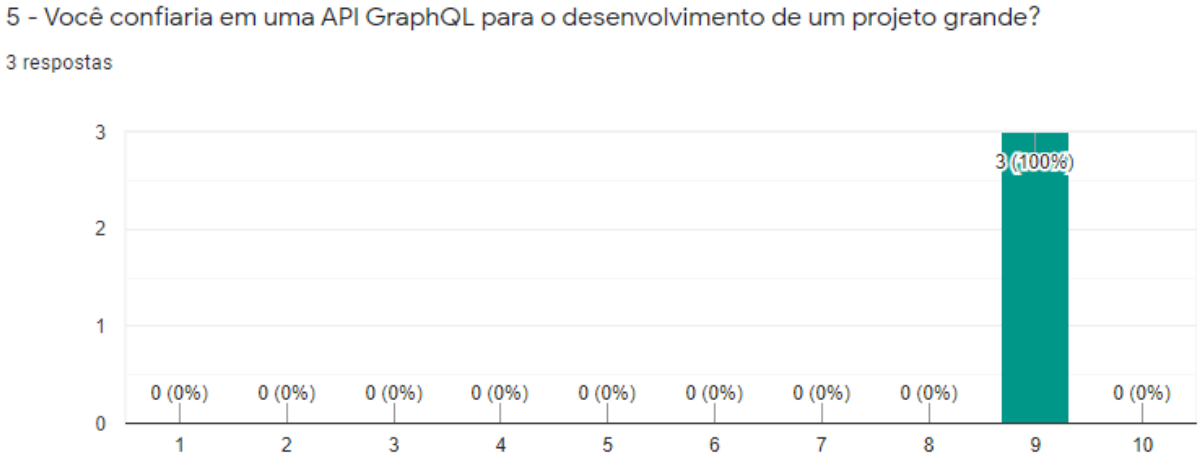
A pergunta 3, muito similar a pergunta anterior, busca saber se os contribuintes já tiveram contato profissional com APIs GraphQL em Node.JS. Essa também é uma pergunta importante para o seguimento da pesquisa. Neste caso, todos tiveram experiência profissional com GraphQL em Node.js.

Figura 22 – Pergunta 4 do questionário

Fonte: Autoria própria.

A pergunta 04, busca conhecer o nível de experiência de cada contribuinte através do tempo em que o desenvolvedor trabalha com a tecnologia. Essa pergunta é importante pois permite ter uma visão diferente de acordo com o nível de experiência de cada um.

Figura 23 – Pergunta 5 do questionário

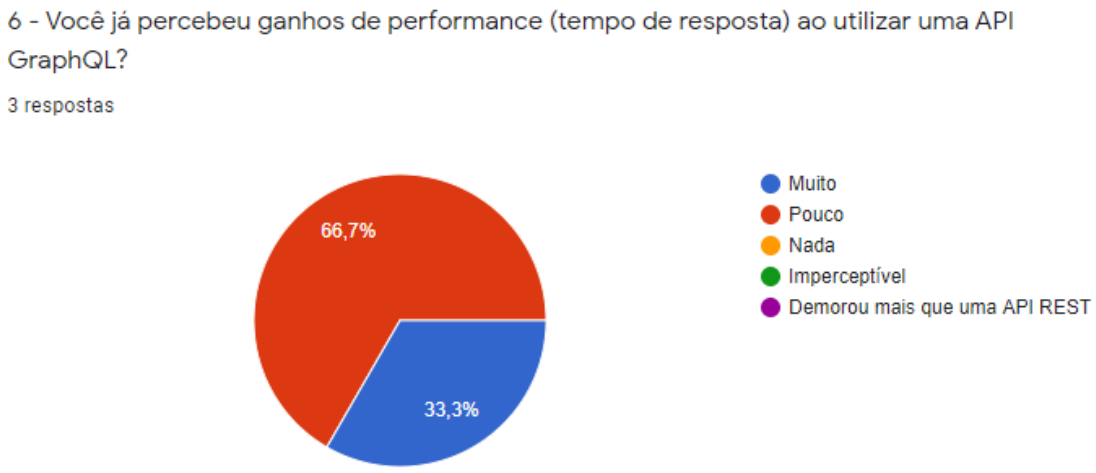


Fonte: Autoria própria.

A pergunta 05, questiona os desenvolvedores sobre a confiança que eles possuem no GraphQL para o desenvolvimento de uma grande aplicação. O resultado dessa pergunta impacta

diretamente na problemática da pesquisa. Os resultados indicam que os contribuintes confiam o suficiente na tecnologia para desenvolver um grande projeto.

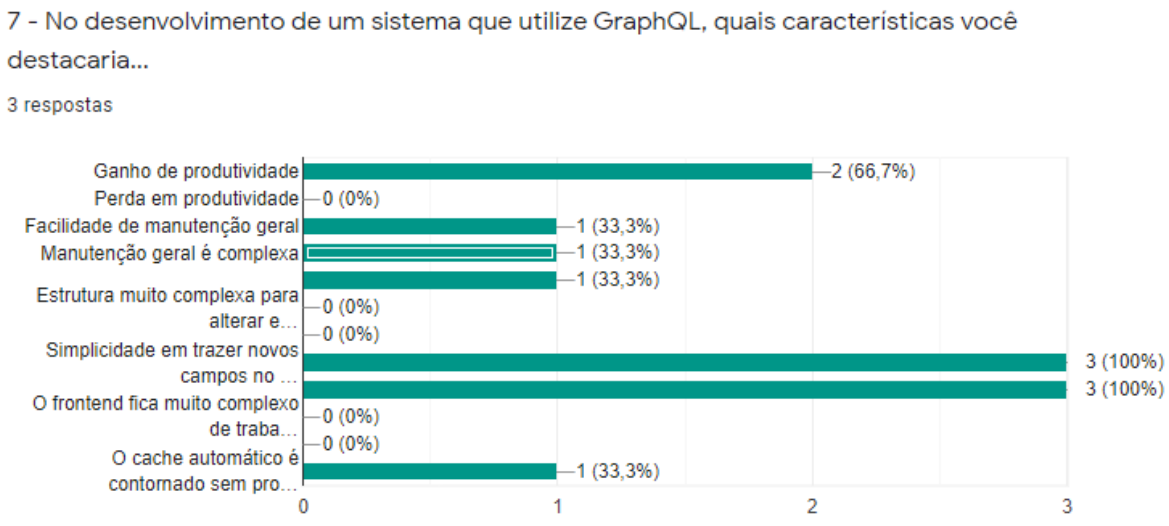
Figura 24 – Pergunta 6 do questionário



Fonte: Autoria própria.

A pergunta 6, questiona os desenvolvedores sobre o tempo de resposta das chamadas HTTP feitas pela API GraphQL. Um dos contribuintes indicou ter encontrado muito ganho de performance enquanto os outros dois perceberam pouco ganho de performance. Essas respostas indicam que mesmo não sendo um grande ganho de performance, ainda é perceptível que GraphQL possui um melhor tempo de resposta.

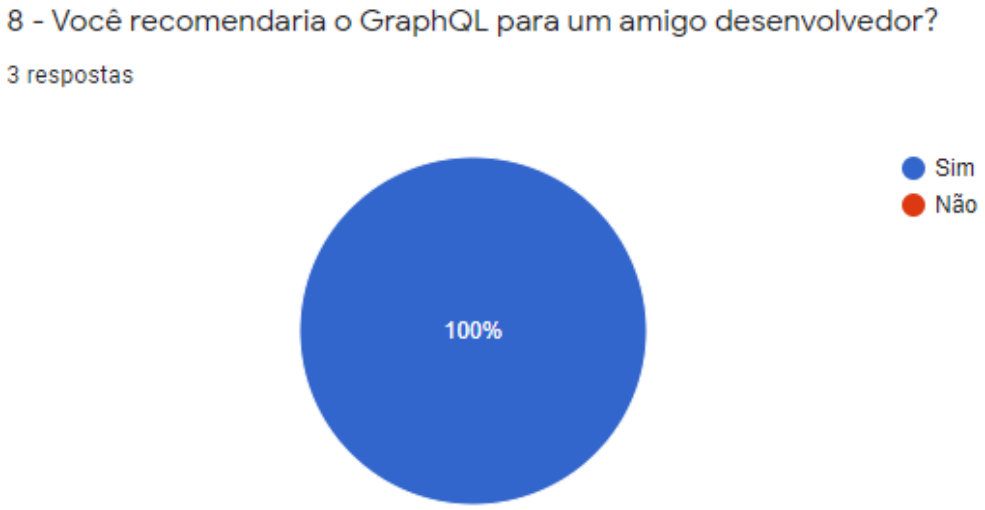
Figura 25 – Pergunta 7 do questionário



Fonte: Autoria própria.

A pergunta 7, busca saber quais as principais características que contribuintes percebem relacionados ao GraphQL. As respostas indicaram que a tecnologia tem em sua maioria um impacto positivo no desenvolvimento.

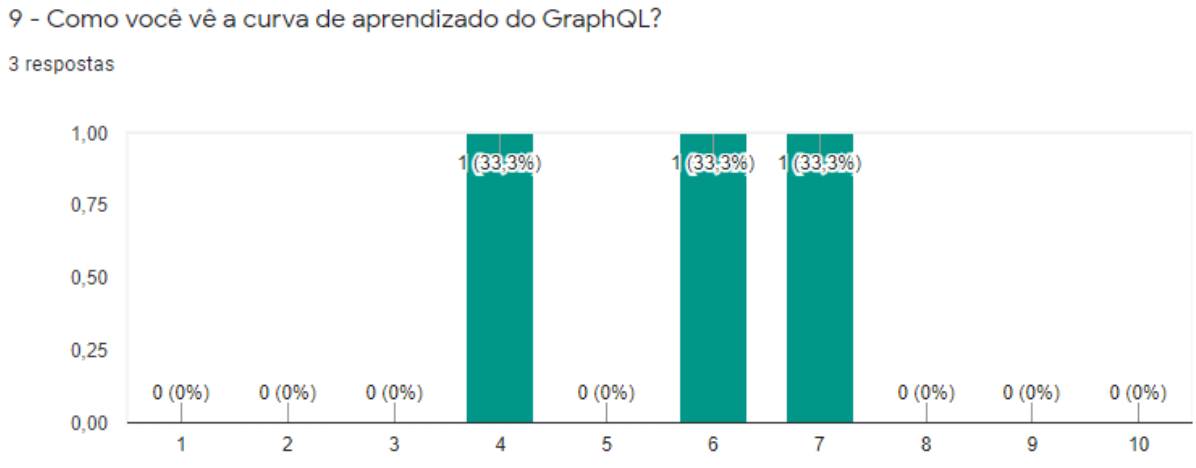
Figura 26 – Pergunta 8 do questionário



Fonte: Autoria própria.

A pergunta 8, questiona se os contribuintes recomendariam o GraphQL para um amigo desenvolvedor. Todas as respostas foram positivas.

Figura 27 – Pergunta 9 do questionário



Fonte: Autoria própria.

A pergunta 9, fala sobre a curva de aprendizado do GraphQL. Essa questão teve um resultado um pouco mais amplo. O tempo de convivência com a tecnologia e a assistência que

tiveram durante o aprendizado, influenciou diretamente no resultado dessa pergunta. Onde olhando os resultados separadamente, foi visto que os desenvolvedores mais experientes percebem uma curva de aprendizado mais otimista do que o contribuinte que possui menos experiência com a tecnologia.

6 CONCLUSÃO

Durante a pesquisa dessa monografia, foi apresentado duas tecnologias para transporte de dados, sendo uma delas o GraphQL. Ao longo do projeto, foi mostrado as possibilidades de melhoria que o GraphQL poderia trazer para o desenvolvimento de softwares.

A pesquisa trouxe resultados que indicam uma melhora na performance e agilidade durante o desenvolvimento de um software.

A pesquisa também traz um dado que apresenta a curva de aprendizado do GraphQL, mostrando que ela pode ser lenta, visto que, olhando separadamente as respostas dos contribuintes, os desenvolvedores mais experientes percebem uma curva de aprendizado mais otimista, enquanto o desenvolvedor com menos tempo de experiencia entende que a curva de aprendizado da tecnologia é lenta.

Sendo assim, pode-se concluir que existe uma melhora, mesmo que pequena, na eficiência da construção de um projeto de software utilizando GraphQL.

As respostas do questionário somado a experiência em desenvolver o protótipo funcional trouxeram resultados que dividem a opinião dos contribuintes da pesquisa.

Olhando separadamente os resultados da pesquisa, o desenvolvedor mais experiente percebeu um bom ganho de performance quando utilizado o GraphQL. Outros dois contribuintes com menos tempo de experiência, relataram ter percebido pouco ganho de performance na API GraphQL em relação a uma API REST.

O desenvolvimento do protótipo permitiu ver os dois lados das respostas. Pelo fato de o GraphQL ter a possibilidade de usar um cache interno, essa funcionalidade permite carregar os dados que estão em cache na tela do usuário enquanto é feito a atualização dos dados em background.

Por outro lado, os tempos de respostas para as requisições não tiveram uma grande diferença, visto que o conjunto de informações transmitas no protótipo não era grande o suficiente para tornar visível a diferença de performance.

Outra conclusão é que sem a funcionalidade de cache, os ganhos em aplicações de pequeno porte são poucas ou imperceptíveis.

6.1 TRABALHOS FUTUROS

Como proposta para trabalhos futuros, foi indicado aprimorar o uso do cache disponibilizado pelo GraphQL.

Outra proposta seria implementar um aplicativo para testar a performance, visto que a proposta inicial do Facebook para a tecnologia envolvia seu aplicativo em rede móvel muito lenta.

REFERÊNCIAS

BONDI, André. **Characteristics of Scalability and Their Impact on Performance.** Disponível em: https://www.win.tue.nl/~johanl/educ/2II45/2010/Lit/Scalability-bondi%202000.pdf. Acesso em: 11 Junho 2020.

BOOCH, Grady. **UML Guia do Usuário**. Rio de Janeiro: Elsevier Editora Ltda, 2012.

CONALLEN, Jim. **Modeling Web Application Architectures with UML.** Disponível em: https://dl.acm.org/doi/pdf/10.1145/317665.317677. Acesso em: 13 Junho 2020.

DUARTE, Luiz. **Boas práticas com MongoDB**. Disponível em: https://blog.umbler.com/br/boas-praticas-com-mongodb/. Acesso em: 23 Junho. 2020.

DURÃES, Geison. **ESCALABILIDADE DE SOFTWARE: O QUE É? COMO OBTER?**. Disponível em: https://blog.tecnospeed.com.br/escalabilidade-de-software-o-que-e/. Acesso em: 10 Setembro. 2020.

FREDRICH, Todd. **What Is REST?**. Disponível em: https://www.restapitutorial.com/lessons/whatisrest.html. Acesso em: 21 Maio. 2020.

GODINHO, Rafael. **Criando serviços REST com WCF**. Disponível em: https://docs.microsoft.com/pt-br/previous-versions/technical-articles/dd941696(v=msdn.10)?redirectedfrom=MSDN. Acesso em: 10 Maio. 2020.

HANASHIRO, Akira. **GraphQL: A revolucionária linguagem de consulta e manipulação de dados para APIs eBook Kindle**. Primeira edição. São Paulo: Casa do Código, 2019.

HUFFINE, Trey. **Polling in Javascript**. Disponível em: https://levelup.gitconnected.com/polling-in-javascript-ab2d6378705a. Acesso em: 27 Maio 2020.

JUNIOR, Antonio. **ARQUITETURA REST API E DESENVOLVIMENTO DE UMA APLICAÇÃO WEBSERVICE**. Disponível em: http://raam.alcidesmaya.com.br/index.php/projetos/article/view/97/95. Acesso em: 21 Maio 2020.

MEDEIROS, Higor. **Introdução ao MongoDB**. Disponível em: https://www.devmedia.com.br/introducao-ao-mongodb/30792. Acesso em: 23 Junho 2020.

NETO, Waldemar. **Construindo APIs testáveis com Node.js**. Disponível em: https://leanpub.com/construindo-apis-testaveis-com-nodejs/read. Acesso em: 26 Maio 2020.

PORCELLO, Eve. **Introdução ao GraphQL** . Primeira edição. São Paulo: Novatec Editora LTDA, 2018.

SEABRA, Matheus. **REST or GraphQL A Performance Comparative Study**. Disponível em: https://dl.acm.org/doi/epdf/10.1145/3357141.3357149. Acesso em: 23 Maio 2020.

SILVA, Edna. **Metodologia da Pesquisa e Elaboração de Dissertação**. Disponível em: https://projetos.inf.ufsc.br/arquivos/Metodologia_de_pesquisa_e_elaboracao_de_teses_e_dissertacoes_4ed.pdf. Acesso em: 21 Junho 2020.

SOMMERVILLE, Ian. **Engenharia de Software**. 9ª edição. São Paulo: Pearson Education, 2011.

SOUSA, Hitalo. **USO DE API REST PARA GESTÃO DE EVENTOS**. Disponível em: https://www.restapitutorial.com/lessons/whatisrest.html. Acesso em: 22 Maio 2020.

YFANTIS, Vissarion. **REST API | What Does It Mean When an Application Has a REST API?.** Disponível em: https://www.parallels.com/blogs/ras/rest-api/. Acesso em: 22 Maio 2020.