CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA

FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

AVALIAÇÃO COMPARATIVA DE DESEMPENHO ENVOLVENDO UMA APLICAÇÃO SINGLE PAGE APPLICATION COM DESENVOLVIMENTO ISOMÓRFICO E TRADICIONAL UTILIZANDO TÉCNICAS BENCHMARK

JOÃO RICARDO ITO MESSIAS

ORIENTADOR: PROF. ME. FABIO PIOLA NAVARRO

Marília – SP
Dezembro/2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA

FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

AVALIAÇÃO COMPARATIVA DE DESEMPENHO ENVOLVENDO UMA APLICAÇÃO SINGLE PAGE APPLICATION COM DESENVOLVIMENTO ISOMÓRFICO E TRADICIONAL UTILIZANDO TÉCNICAS BENCHMARK

JOÃO RICARDO ITO MESSIAS

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em CIÊNCIA DA COMPUTAÇÃO.
Orientador: Prof. Me. Fabio Piola Navarro

Marília – SP

Dezembro /2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

João Ricardo Ito Messias

AVALIAÇÃO COMPARATIVA DE DESEMPENHO ENVOLVENDO UMA APLICAÇÃO SINGLE PAGE APPLICATION COM DESENVOLVIMENTO ISOMÓRFICO E TRADICIONAL UTILIZANDO TÉCNICAS BENCHMARK

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 8 (oito)

Orientador:Fabio Piola Navarro

1º.Examinador:Fábio Dacêncio Pereira

2º.Examinador:Jorge Luiz Barbosa Maciel Junior

Marília, 30 de novembro de 2017.

Dedico este trabalho, primeiramente, a Deus e aos meus pais por todo apoio, incentivo, amor e carinho.

AGRADECIMENTOS

Agradeço a Deus por todas as coisas que Ele tem feito em minha vida.

A minha família por todo amor e incentivo para a conclusão desta graduação.

Aos meus pais pelo apoio em todas as áreas da minha vida, pela cobrança necessária e pelo incentivo e apoio para concluir esta graduação.

A minha namorada e futura esposa Rafaela Fernandes, por todo amor e carinho recebido e pelo apoio e incentivo principalmente nas horas difíceis.

Aos meus amigos, em especial ao Thiago Costa e Leonardo Ademir por todo apoio, incentivo e cobrança durante o curso.

Deixo meu agradecimento a todos os professores, em especial ao Prof. Me. Ricardo José Sabatine, por compartilhar sua experiência e conhecimento auxiliando no desenvolvimento deste trabalho.

Ao Prof. Me. Fabio Piola Navarro agradeço, como meu orientador, por todas as horas dispostas a me auxiliar, apoiar, incentivar e compartilhar seu conhecimento possibilitando a conclusão deste trabalho e mais uma etapa em minha vida.

RESUMO

No início dos tempos das aplicações web, as páginas carregavam conteúdos leves e simples de serem retornados para o navegador. Com o decorrer dos anos, essas aplicações evoluíram drasticamente, consistindo em requisições mais pesadas para o servidor. Sendo assim, foi necessário criar um novo tipo de abordagem de desenvolvimento web para amenizar e solucionar os problemas existentes na época, no qual foi chamado de desenvolvimento *Single Page Application* (SPA). Este trabalho consiste na realização de uma comparação avaliativa entre dois tipos de desenvolvimento SPA - tradicional e isomórfico - envolvendo uma única aplicação *web* utilizando técnicas *benchmark*, no qual foi possível levantar os pontos positivos e negativos de cada abordagem para auxiliar na melhor tomada de decisão de arquitetos e desenvolvedores.

Palavras-chave: Single Page Application, Desenvolvimento Isomórfico, Benchmark, Desenvolvimento Web.

ABSTRACT

In the early days of web applications, pages loaded light and simple content to be returned to the browser. Over the years, these applications have developed dramatically, requiring heavier server. Therefore, it was necessary to create a new type of web development approach to soften and solve the existing problems at the time, in what was called the Single Page Application (SPA) development. This work consists in the accomplishment of an evaluation comparison between two types of SPA development - traditional and isomorphic - involving a single web application using benchmark techniques, in which it was possible to raise the positives and negatives of each approach to help in the better decision making of architects and developers.

Keywords: Single Page Application, Isomorphic Development, Benchmark, Web Development.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

HTML - *Hypertext Markup Language*

SPA - *Single Page Application*

HTTP - *Hypertext Transfer Protocol*

JSON - *JavaScript Object Notation*

AJAX - *Asynchronous JavaScript And XML*

CSS - *Cascading Style Sheets*

DOM - *Document Object Model*

WAMP - *Windows/Apache/MySQL/PHP*

SEO - *Search Engine Optimization*

MVC - *Model–view–controller*

JMETER - *Java Meter*

SUMÁRIO

INTRODUÇÃO

No início dos tempos das aplicações e navegações *web*, a comunicação entre cliente/servidor funcionava da seguinte maneira: o cliente, utilizando um navegador, estabelece uma página de seu desejo e, ao fazer essa solicitação, era enviado a determinada ação para o servidor *web* no qual retornava a página requisitada.

Esse tipo de abordagem era muito eficiente antigamente, pois os *HTMLs* eram simples e estáticos, no qual não era necessário ultra navegadores para realizar essas requisições.

Com o decorrer dos anos, as aplicações *web* evoluíram drasticamente, consistindo em requisições mais pesadas para o servidor. Sendo assim, foi necessário criar um novo tipo de abordagem de desenvolvimento web para amenizar e solucionar os problemas existentes na época, no qual foi chamado de desenvolvimento em *Single Page Application* (SPA).

O desenvolvimento *SPA* é uma aplicação no qual não necessita o carregamento de outras páginas durante o uso, ou seja, é tudo realizado em uma única página. Um bom exemplo dessa aplicação é o Gmail, no qual ele pode responder rapidamente com as interações do cliente sem a necessidade de fazer uma requisição de ida e volta para o servidor apenas para trazer uma página.

Grande parte da lógica dessas aplicações encontra-se no servidor do cliente, no qual ele consegue se comunicar com o servidor através de uma API de dados. O servidor pode ser programado em linguagens que tenham suporte para HTML.

Uma vez que os arquivos do lado do cliente são baixados pelo navegador, a aplicação do lado do servidor é inicializada, onde há uma busca dos dados via API para que a mesma retorne uma página HTML.

Quando o arquivo do lado do cliente já foi carregado, ele pode suportar navegações rápidas entre as páginas sem a necessidade de recarregá-las novamente, consistindo assim em uma rápida manipulação no site em tempo de execução.

Existem dois tipos de abordagem distintas de desenvolvimento SPA para realizar as requisições ao servidor: o tradicional (server-side) e o isomórfico (client-side).

O desenvolvimento SPA tradicional – abordagem mais utilizada atualmente – tem como principal característica a realização das requisições do cliente do lado do servidor, ou seja, em uma linguagem totalmente distinta da linguagem do cliente.

Já o desenvolvimento SPA isomórfico, utilizando o chamado "*Isomorphic Javascript*", é possível usar alguns frameworks baseados em *Javascript* (JS) para realizar a unção das requisições tanto do lado do cliente quanto do servidor de forma única.

Objetivo

O objetivo deste trabalho é realizar uma avaliação comparativa de desempenho em uma mesma aplicação web, utilizando abordagens distintas de programação (tradicional e isomórfico), no qual será extraído informações valiosas como latência, desvio padrão, porcentagem de erro, vazão e tempo das requisições. Com base nos dados coletados no teste será possível obter, a partir de um estudo, os pontos positivos e negativos de cada abordagem e assim, contribuir para a tomada de decisão de arquitetos e desenvolvedores sobre a melhor tecnologia a ser utilizada para determinada aplicação web.

Metodologia

Para que o objetivo deste trabalho possa ser alcançado, foram necessários adotar uma abordagem metodológica que envolvessem desenvolvimento web, aplicações *Single Page Application* (tradicional e isomórfico) e testes *Benchmark*.

Os procedimentos metodológicos utilizados para atender todas as necessidades deste trabalho foram:

- Levantamento e pesquisas bibliográficas sobre os temas utilizados neste trabalho como: desenvolvimento web, *Single Page Application*, desenvolvimento isomórfico e *Benchmark*;
- Pesquisas sobre termos e tecnologias que serão utilizadas no projeto, para melhor atender as necessidades do mesmo;
- Fundamentar teoricamente os termos e tecnologias anteriormente pesquisados;
- Implementar as aplicações web envolvendo um desenvolvimento em *Single Page Application* tradicional e isomórfico;
- Realizar todos os testes de desempenho necessários entre a aplicação SPA tradicional e isomórfica para obter-se uma avaliação comparativa;
- Escrever tecnicamente os resultados obtidos dos testes de desempenho.

Organização do Trabalho

É apresentado no Capítulo 1, após a realização do levantamento bibliográfico, uma fundamentação teórica envolvendo as principais técnicas de desenvolvimento web utilizadas neste trabalho como *Single Page Application* tradicional, isomórfico e o real funcionamento de suas respectivas arquiteturas.

No Capítulo 2 apresenta todas as tecnologias e suas respectivas definições utilizadas no trabalho em ambas as abordagens de programação SPA (tradicional e isomórfico) e a maneira como foi desenvolvido.

O Capítulo 3 demonstra todos os testes e seus respectivos resultados da análise comparativa envolvendo uma aplicação SPA tradicional e isomórfica, afim de apresentar as vantagens e desvantagens da utilização de cada uma.

O Capítulo 4 expõe a conclusão deste trabalho, analisada e avaliada a partir dos resultados apurados no Capítulo 3.

Capítulo 1

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados todas as tecnologias e assuntos que serão abordados no projeto, com o objetivo de mostrar aos leitores informações que levam ao bom entendimento e compreensão a área de desenvolvimento web.

1.1 Desenvolvimento Web

É indubitável que a área de Tecnologia da Informação vem em constante crescimento nesses últimos tempos até então. O conceito de Desenvolvimento Web envolve uma grande área de trabalho extraída de todo conteúdo relacionado a World Wide Web. Ademais, este desenvolvimento envolve tanto a criação de uma simples página da Internet até uma grande aplicação, no qual ambas são baseadas na web.

Criada em 1989 por Sir Tim Berners-Lee, a World Wide Web possuía o objetivo de facilitar o compartilhamento de informações entre um computador e outro onde, na época, não era uma tarefa simples de ser realizada. Segundo a W3, "[...] *In those days, there was different information on different computers, but you had to log on to different computers to get at it. Also, sometimes you had to learn a different program on each computer. Often it was just easier to go and ask people when they were having coffee…* [...]", comentou Berners-Lee (W3, 2017).

Sendo assim, Sir Tim Berners-Lee enviou uma proposta de resolução para o problema de compartilhamento de informações, utilizando uma tecnologia chamada hipertexto, onde ele não iria somente resolver este tipo de problema, mas também compartilhar informações de milhões de computadores que estavam conectados entre si através da internet.

Com a evolução da World Wide Web, foram necessários a criação de metodologias e ferramentas para o desenvolvimento de sites que compõem a internet, no qual chamamos de desenvolvimento web.

Em relação ao desenvolvimento web, existem dois tipos de linguagens que podem compor uma página web: client-side (front-end) e server-side (back-end).

1.1.1 Client-Side

As linguagens client-side conseguem comunicar-se através do navegador, ou seja, diretamente do browser. Geralmente são aplicações pequenas com o intuito de exibir e atualizar dados remotamente. Essas aplicações são empacotadas, permitindo com um único download, fazer a instalação do aplicativo em algum dispositivo do cliente (W3, 2017).

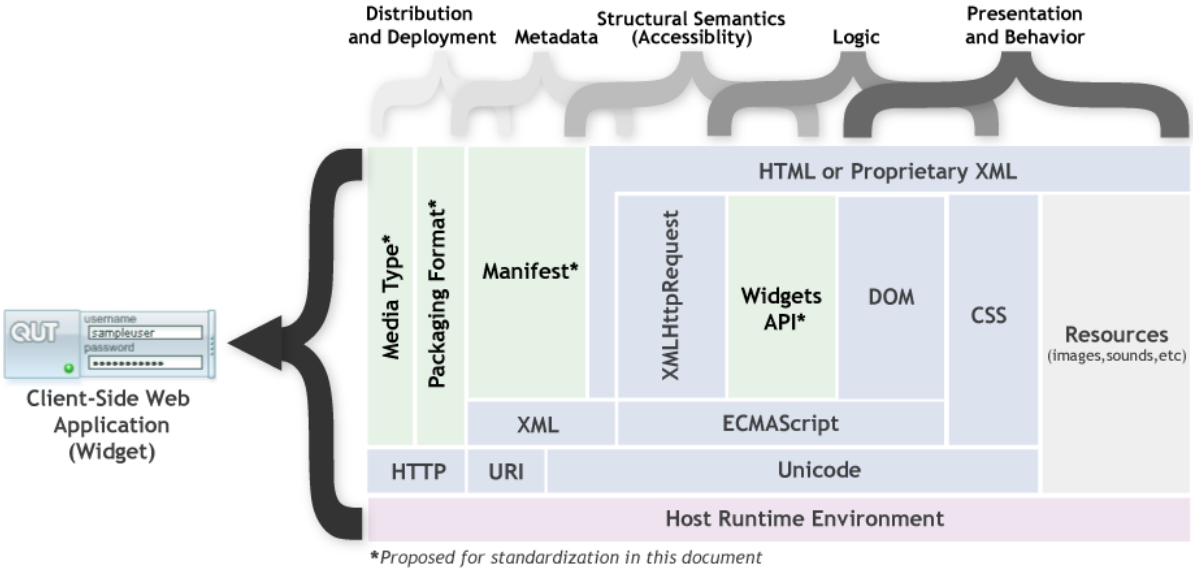

Figura 1 – Arquitetura de uma aplicação web do lado do cliente (W3, 2017).

A Figura 1 mostra uma arquitetura usual de uma aplicação Web do lado do cliente. A base dessa arquitetura consiste em pequenos softwares criados diretamente ou com funcionalidades parecidas a de um navegador Web. Eles existem para suportar o protocolo HTTP, URIs, Unicode, ECMAScript (JavaScript), CSS, DOM e recursos multimídia, como sons e imagens. Ademais, as aplicações cliente-side suportam também recursos para fazer solicitações de dados assíncronas sobre o HTTP, utilizando por exemplo o XMLHttpRequest (W3, 2017).

1.1.2 Server-Side

As linguagens server-side conseguem comunicar-se diretamente do servidor, ou seja, um código escrito para server-side é enviado e processado no servidor, e assim, devolvido para o navegador Web, diferentemente da linguagem front-side, que executam dentro do próprio navegador.

É possível observar que algumas Web pages não possuem a extensão .html, mas sim outros tipos de extensões como .php, .asp, .aspx, .jsp. Esses são alguns exemplos de linguagens server-side, no qual elas são utilizadas para criar Web pages com seções que podem variar dependendo dos valores solicitados e retornados do servidor para serem exibidos em um navegador Web (W3, 2017).

1.1.3 Single Page Application

O termo Single Page Application (SPA) é um dos vários tipos de desenvolvimentos existentes da World Wide Web. O SPA é uma aplicação Web que possibilita realizar requisições assíncronas dos usuários e retorna-las para o navegador Web sem a necessidade de atualizar/recarregar a Web page (Oh, Ahn, Jeong, Lim, Kim, 2013, p.292).

Para realizar esse processo, é necessário a utilização do AJAX (Asynchronous JavaScript and XML), no qual realiza requisições assíncronas para o servidor, onde ao retornar a resposta da solicitação para o navegador Web, esses dados podem ser manipulados e trabalhados através de scripts (com o Javascript).

Segundo (Mikowski, Powell, 2012, p.4), o SPA veio para acabar com o conceito das plataformas Flash e Java para os navegadores Web, no qual foram os mais utilizados por muito tempo, por sua alta capacidade, velocidade e consistência na área de renderização do navegador, sendo até mesmo superiores aos do Javascript.

Mas com o passar do tempo, o JavaScript superou os déficits existentes em relação a renderização do navegador, vencendo as dificuldades que existiam anteriormente e proporcionou várias outras vantagens expressivas em relação as outras plataformas (Mikowski, Powell, 2012, p.4).

1.1.4 AJAX

Segundo (Oh, Ahn, Jeong, Lim, Kim, 2013, p.292), a utilização do AJAX para a realização o processo de Single Page Application possibilita que a aplicação Web torne-se mais interativa, dinâmica e responsiva, sendo semelhante a um aplicativo desktop.

Para compreender melhor sobre uma requisição assíncrona via AJAX, (Emer, 2017) explica que em uma aplicação Web, possui um único fluxo de execução, que inicia no carregamento da página até o término do mesmo. Sendo assim, é possível criar requisições assíncronas (fluxos assíncronos) via Javascript que podem ser executadas dinamicamente durante a aplicação, sem alterar no fluxo principal da execução.

Segundo (Emer, 2017), “[...] Você pode disparar uma série dessas tarefas sem precisar esperar que cada uma se complete para prosseguir [...]”. Sempre que é feito uma solicitação via AJAX para o servidor, somente quando a resposta da requisição for realizada com sucesso, os dados solicitados são retornados para o fluxo principal de execução, e assim poderão ser utilizados.

Quando os dados de uma requisição AJAX são retornados para o navegador Web, é possível fazer a manipulação dessa resposta sem a necessidade do carregamento da página. Por exemplo, quando o navegador Web recebe a resposta de sucesso do servidor a partir de uma solicitação AJAX, o desenvolvedor do sistema consegue exibir essa resposta dentro da Web page atual que foi feita essa requisição, sem a necessidade da atualização da página.

Diferentemente da aplicação baseada em Single Page Application, uma aplicação Web tradicional realiza os mesmos passos da SPA, mas com algumas alterações. As respostas das solicitações por AJAX para o servidor são retornadas em formato JSON, já as aplicações tradicionais, retornam uma Web page.

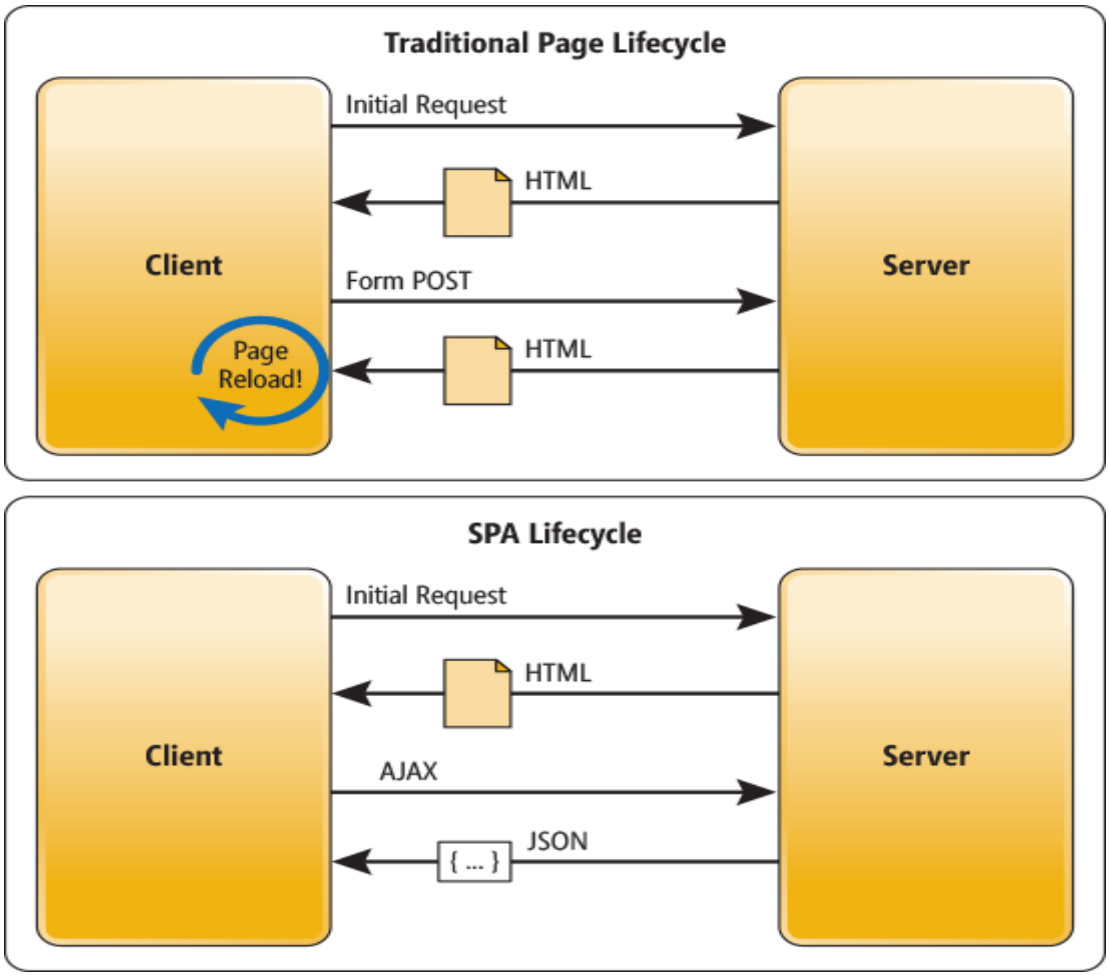

Figura 2 – Ciclo de vida de uma aplicação básica web contra uma SPA Tradicional (Wasson, 2017).

Segundo (Wasson, 2017), as diferenças de uma Web page baseado em SPA são bem transparentes e fáceis de compreender. “[...] *One benefit of SPAs is obvious: Applications are more fluid and responsive, without the jarring effect of reloading and re-rendering the page. Another benefit might be less obvious and it concerns how you architect a Web app. Sending the app data as JSON creates a separation between the presentation (HTML markup) and application logic (AJAX requests plus JSON responses).* [...]”.

1.1.4.1 JSON

O conceito de JSON (JavaScript Object Notation) possui o objetivo de formatar um determinado tipo de objeto baseado no subconjunto da linguagem de programação Javascript e pelo padrão ECMA-262. Suas formatações servem para facilitar leitura e escrita em relação aos seres humanos, e analisar e gerar em relação as máquinas (JSON, 2017).

O JSON é um padrão de formatação internacional, sendo assim, totalmente independente da linguagem que está utilizando-o. Algumas linguagens usuais que utilizam esse método de formatação são: C, C++, C#, Java, JavaScript, Perl, Python, PHP, etc. (JSON, 2017).

O JSON pode ser construído a partir de duas estruturas:

- Um conjunto de chave/valor. Na maioria das linguagens, é possível gerar um JSON através de objects, structs ou arrays;
- Uma lista ordenada de valores. Na maioria das linguagens, é possível gerar um JSON através de arrays, vector, list ou sequence.

1.2 Desenvolvimento Isomórfico

Com a evolução da World Wide Web, o termo isomórfico dentro da área de desenvolvimento web tem sido muito discutido nos últimos anos, desde que os navegadores progrediram de exibir/carregar não somente páginas com conteúdo simples, mas grandes aplicações Web.

O conceito de desenvolvimento isomórfico trata-se de realizar uma aplicação web onde o lado do cliente (client-side) e o lado do servidor (server-side) atuam lado a lado, em um mesmo ambiente de programação, utilizando linguagens que conseguem comunicar-se tanto com o Browser, quanto com o servidor, sem a necessidade de utilizar mais de uma linguagem de programação para a execução do mesmo.

Quando falamos de desenvolvimento isomórfico, o Javascript é extremamente importante para a construção do mesmo, pois ele possui

bibliotecas e frameworks que possibilitam a realização da comunicações client-server-side em uma única linguagem de programação, denominando assim o termo Isomorphic Javascript (Brehm, 2017).

Um dos principais frameworks baseados em Javascript para realizar esse tipo de desenvolvimento é o Node.js, um framework simples leve e eficiente, executando suas operações e requisições para o servidor diretamente do navegador Web do cliente (Node, 2017).

As linguagens baseadas em script tornaram-se cada vez mais completas e poderosas, em relação a desenvolvimentos de aplicações tanto do lado do cliente como do lado do servidor. Uma das mais completas linguagens para executar esse tipo de tarefa é o Javascript, onde ela é amplamente utilizada para a realização de desenvolvimentos de aplicações Web (Ogasawara, 2013, p.13).

Este tipo de desenvolvimento começou a ter mais visibilidade, quando um outro tipo de desenvolvimento Web começou a demonstrar algumas fraquezas em relação a sua execução, o chamado Single Page Application (seção 2.2).

A aplicação SPA foi de grande ajuda e auxílio para desenvolver Web ao decorrer da evolução da World Wide Web, no qual ela é utilizada até nos dias de hoje. Mas, com o decorrer do tempo, ela começou a apresentar alguns problemas significativos como a má indexação das Web pages em relação aos mecanismos de busca (SEO), a performance em relação ao tempo de carregamento da Web page e o custo de manutenção do código (Brehm, 2017).

Estes tipos de desenvolvimento são baseados em uma arquitetura Model-View-Controller (MVC). Ele é um padrão de arquitetura de software, onde possui o objetivo de auxiliar os desenvolvedores em relação a escrita e leitura da aplicação, tornando o código mais sustentável. Este tipo de padrão de projeto tem sido extremamente estudado por programadores, tornando-o um dos mais utilizados, por possuir como característica códigos de fácil manutenção e bem reaproveitados (Google, 2017).

O padrão MVC é constituído por três componentes:

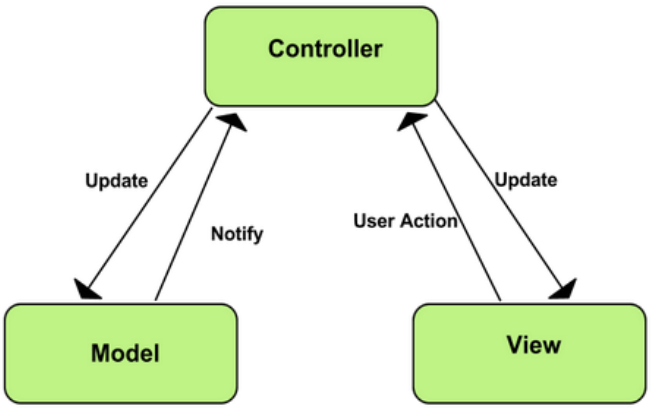

Figura 3 – Padrão MVC (Google, 2017).

- Model são todos os objetos de dados da aplicação são armazenados, ou seja, ele é uma modelagem do banco de dados. Ele não possui o conhecimento do que acontece na view e no controller. Quando acontece alguma alteração na model, ele notifica alguns observers que algo aconteceu (Google, 2017);
- View é o visual/apresentação da aplicação para o usuário e como eles interagem com o sistema. Ela é constituída por linguagens front-end como HTML, CSS e JavaScript. Ela não realiza interações com o modelo, pois esse é o trabalho do controlador (Google, 2017);
- Controller é o coração da aplicação, pois é nele que são feitas as tomadas de decisões sobre o que a model ou a view irão realizar. Quando é realizado alguma alteração na Model, o Controller atualiza a View para o usuário. Quando o usuário realiza uma alteração na View, o Controller atualiza a Model de acordo com as alterações do usuário (Google, 2017).

As figuras 4 e 5 demonstram o funcionamento dos desenvolvimentos SPA tradicional e isomórfico, respectivamente:

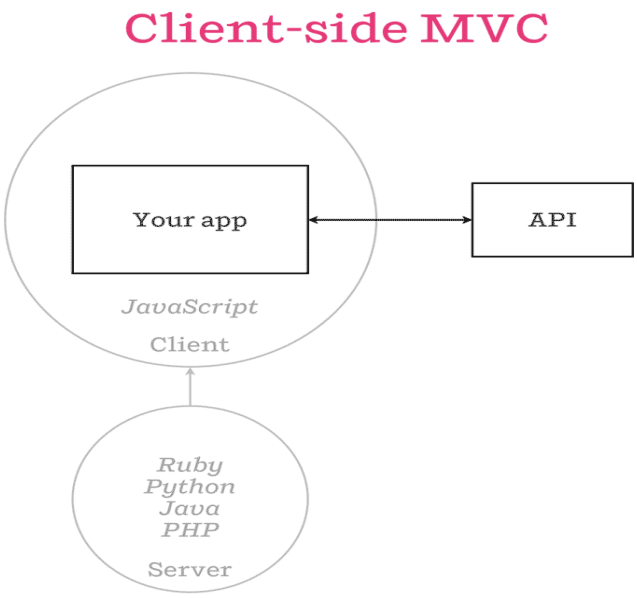

Figura 4 – Desenvolvimento SPA Tradicional (Brehm, 2017).

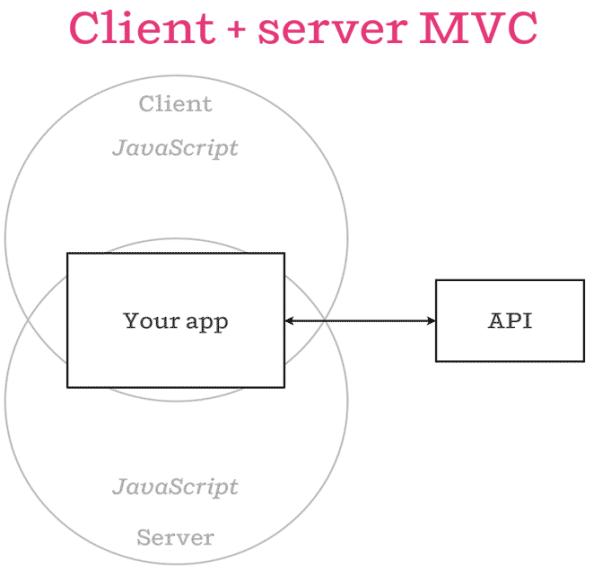

Figura 5 – Desenvolvimento SPA Isomórfico (Brehm, 2017).

Na Figura 4 é possível observar que a aplicação será construída dentro do lado do cliente utilizando uma linguagem front-end (como o Javascript) e outro tipo de linguagem para realizar as comunicações com o servidor (como o Ruby, Python, Java ou PHP), para fazer as comunicações com a API.

Já na Figura 5 é possível observar que a aplicação será construída tanto do lado do cliente quanto do lado do servidor, utilizando uma única linguagem (como o Javascript), para fazer as comunicações com a API.

Como demonstrado nas Figuras 4 e 5, no desenvolvimento isomórfico o mesmo código encontra-se no lado do servidor e do cliente, facilitando assim na manutenabilidade do projeto, pois em ambos os lados estarão na mesma linguagem, evitando assim lógicas duplicadas, diminuindo o tempo de programação e manutenção do código (Brehm, 2017).

Segundo (Brehm, 2017), Engenheiro de Software da Airbnb, umas das maiores empresas de cotação de hotéis online, comenta que com o desenvolvimento isomórfico a maioria das aplicações e lógicas de apresentação são realizadas tanto no servidor quanto no cliente e, com isso, a uma grande melhoria na aplicação desenvolvida em relação aos desempenhos otimizados da Web page, manutenabilidade menos complexa, de fácil entendimento e com uma melhor indexação da web page por mecanismos de busca.

De acordo com (Brehm, 2017), a maioria das empresas de desenvolvimento web não utilizam frameworks que funcionam tanto do lado do servidor quanto no lado do cliente (como o Node.js) mas, quando essas empresas obterem o conhecimento das vantagens e benefícios que o desenvolvimento isomórfico possui, é inevitável que essas organizações passem a compartilhar o mesmo código em ambos os lados. Em Javascript Isomórfico, a aplicação inteira não necessariamente precisa estar com o código inteiro concentrado em Javascript. É possível construir uma aplicação híbrida, sendo possível implementar aos poucos o desenvolvimento isomórfico aos poucos, sem realizar grandes alterações na aplicação original.

1.3 Arquitetura e comunicação

1.3.1 Arquiteturas bloqueantes

Uma arquitetura bloqueante possui como sua principal característica a paralização de um processamento enquanto é realizado uma entrada/saída (I/O) no servidor. Algumas plataformas de desenvolvimento como Java, PHP (utilizado no desenvolvimento SPA tradicional), Ruby e Python utilizam o

modelo bloqueante, que pode ser chamado também de *blocking-thread* (Pereira, C. R, 2017).

Para melhor exemplificar este tipo de arquitetura, ao considerarmos que um processo é uma requisição feita pelo usuário ao servidor e, com o decorrer da execução da aplicação, outros usuários também farão novas solicitações para o servidor. Em uma aplicação que utiliza este tipo de modelo bloqueante, as requisições que chegam no servidor são enfileiradas e processadas uma a uma, não permitindo múltiplos processamentos (Pereira, C. R, 2017).

Enquanto uma requisição é realizada, as outras requisições continuam ociosas na fila de espera até que o processo atual seja finalizado. Assim que a requisição em processamento termina, a próxima requisição da fila é executada.

Essa é uma das arquiteturas clássicas, utilizada em várias aplicações web. Com o aumento de acessos no sistema, a ocorrência de gargalos aumenta, pois, a fila ociosa das requisições que estão no servidor fica cada vez maior. Segundo Pereira (Pereira, C. R, 2017), para que o sistema fique mais eficiente, é possível realizar a atualização nos hardwares dos servidores (solução custosa) ou buscar novas tecnologias que façam um bom uso do hardware do servidor.

1.3.2 Arquiteturas não bloqueantes

Diferentemente das arquiteturas bloqueantes, o modelo não bloqueante é uma arquitetura *non-blocking thread*, ou seja, se o sistema desenvolvido necessita realizar muito I/O, utilizar este modelo é uma boa forma de aumentar a performance com relação ao consumo de memória e processamento dos servidores (Pereira, C. R, 2017).

Com a arquitetura não bloqueante, não existe uma fila ociosa de requisições feitas pelo usuário, sendo assim, não há a necessidade de esperar uma requisição em andamento terminar para que a próxima seja executada. Com isso é possível aproveitar ao máximo o processamento do servidor.

O Node.js é uma plataforma que trabalha com este tipo de modelo. Sendo assim, o desenvolvimento SPA isomórfico realizado neste trabalho utiliza a arquitetura não bloqueante.

1.3.3 Single-thread vs. Multi-thread

Os sistemas e aplicações baseados em single thread são desenvolvimentos que terão a instância de um único processo. Em Node.js é permitido trabalhar apenas com o esquema de *single-threads,* mas há outras formas de criar um sistema concorrente como, por exemplo, a utilização de clusters que são módulos nativo do Node.js (Pereira, C. R, 2017).

Já os sistemas que utilizam *multi-threads* possuem a capacidade de executar vários threads de uma vez sem que uma seja interferida pela outra. Mas, para que isso aconteça, o sistema deve utilizar processadores que possuem a capacidade de executar mais de um thread ao mesmo tempo.

1.3.4 Assíncrono vs. síncrono

Para fazer as solicitações HTTP para o servidor, é possível realiza-las de duas formas: utilizando requisições síncronas e assíncronas.

A grande diferença entre esses tipos de requisições é a espera para receber a resposta do servidor solicitado.

Uma aplicação utilizando o modelo síncrono, tanto a chamada quanto a resposta da requisição são feitas de maneira sequencial, ou seja, é necessário que seja retornada a resposta da requisição anterior para a execução de uma nova requisição (DevMedia, 2017).

Este tipo de modelo de requisição síncrono foi utilizado para a realização do desenvolvimento SPA tradicional deste projeto, embasado na linguagem PHP.

Ao contrário do modelo síncrono, uma aplicação utilizando o modelo assíncrono não necessita esperar a resposta de uma requisição para a execução de outra. Todas as requisições são enviadas ao mesmo tempo para o servidor e, à medida que elas são finalizadas, logo já são retornadas para o cliente.

Este tipo de modelo de requisição assíncrono foi utilizado para a realização do desenvolvimento SPA isomórfico deste projeto, embasado na linguagem Node.js.

1.3.5 Tecnologias baseadas em eventos

As tecnologias baseadas em eventos são um conjunto de ações que são realizadas nos eventos de uma página web, como por exemplo, em imagens e textos do HTML. Praticamente todas as ações que o usuário realiza em uma determinada página web podem ser consideradas eventos (DevMedia, 2017).

Uma das principais linguagens baseadas em eventos é o Javascript, que se tornou cada vez mais popular em relação as plataformas web.

1.3.5.1 Javascript

Para a construção de um desenvolvimento baseado inteiramente do lado do cliente (isomórfico), a principal ferramenta para a realização do mesmo é utilizando a linguagem JavaScript.

JavaScript é uma linguagem Web simples e compacta, totalmente orientada a objeto utilizando funções de primeira classe (possibilidade de passar funções como parâmetro para outras funções), no qual ela é conhecida como uma linguagem de script que pode ser tanto executada em um navegador Web, como em um ambiente não-navegado (Mozilla, 2017).

As aplicações desenvolvidas em um ambiente JavaScript é executado totalmente do lado do cliente (cliente-side) onde, baseado em eventos, é possível realizar diferentes tipos de ações dentro de uma Web page em tempo de execução, e até mesmo construir uma Web page por completo. Para desenvolvimento Web, além de possuir uma linha de aprendizado relativamente fácil, a linguagem JavaScript é uma poderosíssima ferramenta para o controle de uma página Web (Mozilla, 2017).

A programação em Javascript não tem nenhum relacionamento com a linguagem de programação Java, mas ela possui uma sintaxe básica como qualquer outra linguagem de programação como o Java ou C++, visando reduzir o número de novos métodos para o aprendizado dessa linguagem (Mozilla, 2017).

Por ser uma linguagem orientada a objetos, é possível realizar a criação desses objetos anexando ações e informações em tempo de execução, atuando diferentemente das linguagens convencionais como Java ou C++.

Quando um objeto é criado, é possível que ele sirva como modelo para a criação de outros objetos (Mozilla, 2017).

Para a desenvolvimento deste projeto, um dos principais requisitos para a realização de uma aplicação onde tanto o lado do cliente (client-side) quanto o lado do servidor (server-side) encontram-se no mesmo ambiente, é a utilização da linguagem JavaScript, pois com ferramentas e bibliotecas auxiliares (como o Node.js) é possível realizar a construção do mesmo.

1.3.5.1.1 Node.js

Node.js é uma biblioteca baseada na linguagem JavaScript no qual possui como objetivo criar plataformas para o desenvolvimento de programas e aplicações Web onde tanto as requisições do lado do servidor quanto as requisições do lado do cliente encontram-se no mesmo código, sem a necessidade da utilização de mais de uma linguagem de programação para o desenvolvimento de uma aplicação completa (Lopes, 2014).

Com o constante crescimento das aplicações relacionadas a Web, é indubitável que ao tratar-se de grandes projetos e desenvolvimentos, é necessário escolher as melhores ferramentas para a construção de um projeto, visando obter o melhor desempenho em relação ao carregamento de um Web Page, otimização dos mecanismos de busca e uma melhor manutenabilidade do código para os desenvolvedores envolvidos no projeto.

As plataformas criadas pelo Node.js são construídas e executadas sobre uma máquina virtual JavaScript chamado V8 JavaScript Engine, da Google. Ele foi desenvolvido juntamente com o projeto Chromium, da Google, no qual agora pode ser executado em diversas aplicações como Node.js e MongoDB.

O V8 JavaScript Engine é uma máquina virtual de alto desempenho baseado em JavaScript e codificado em C++. Ele é executado e compilado dentro do navegador da Google, o Chrome. Mas, por tratar-se de um código aberto, é possível utilizado em outras aplicações Web (Google, 2017).

Sendo assim, com a aplicação web rodando inteiramente em um mesmo ambiente de desenvolvimento, é possível realizar testes comparativos de desempenho entre ambientes de desenvolvimento tradicional (onde possuem o lado do cliente separado do lado do servidor), possibilitando demonstrar uma análise completa de desempenho em relação aos ambientes comparados, a

fim de classificar os melhores tipos de desenvolvimento para determinadas aplicações web.

1.3.5.1.2 NGINX

O NGINX é um servidor de HTTP de código aberto, com o objetivo de realizar um papel de intermediário entre o cliente e servidor. Ele utiliza o recurso de proxy reverso, onde é instalado entre a internet e o servidor web, onde toda a solicitação realizada em um site será transmitida primeiro para o NGINX e, somente depois disso, é redirecionado para o servidor (NGINX, 2017).

O NGINX é bem conhecido pelo seu alto desempenho, estabilidade, recursos amplos e completo, simples configurações e baixo consumo de recursos. Ele foi desenvolvido para solucionar um problema chamado C10K (processar requisições de 10 mil clientes ao mesmo tempo) (NGINX, 2017).

Diferentemente dos servidores tradicionais, o NGINX não é dependente de threads para processar as solicitações do servidor. Ele utiliza uma arquitetura escalável baseada em eventos assíncronos, o mesmo que o Node.js utiliza.

Ademais, por estar em uma camada antes do envio da requisição ao servidor, é possível realizar validações e verificações de segurança dessas requisições, identificando se a solicitação é segura ou não antes mesmo de chegar no servidor.

1.4 Trabalhos correlatos

Nesta sessão serão demonstrados os trabalhos correlatos utilizados para auxiliar e concretizar os resultados obtidos neste trabalho de conclusão de curso.

Um dos trabalhos utilizados na realização deste projeto foi desenvolvido por Kai Lei, Yining Ma e Zhi Tan, onde eles realizaram uma avaliação e comparação de desempenho em tecnologias de desenvolvimento web como PHP, Node.js e Python.

Segundo eles (Lei, K., Ma, Y., Tan, Z, 2014, p.661-668), conforme a o aumento em grande escala da quantidade de dados que são transportados na web e a alta concorrência para o acesso a esses dados, as tecnologias utilizadas para a realização de aplicações web serão de grande importância nas novas gerações de sites.

Eles também realizaram testes comparativos entre as linguagens abordadas (Node.js, PHP e Python-Web), realizando testes de benchmark e de cenário, sendo possível avaliar quais são os pontos positivos e negativos de seus diferentes modelos de programação em determinadas aplicações web.

Para a realização do teste comparativo, foram necessários realizar as configurações de ambiente físico, servidor e das ferramentas de teste.

Configuração do Ambiente Físico

Para a realização da comunicação de um cliente com o servidor, foram necessários configurar uma máquina para o lado do cliente e uma máquina para o lado do servidor, ambas utilizando uma Internet com cerca de 100-Mbps cada. O esquema de requisição e resposta entre o cliente-servidor funciona conforme demonstra a Figura 6.

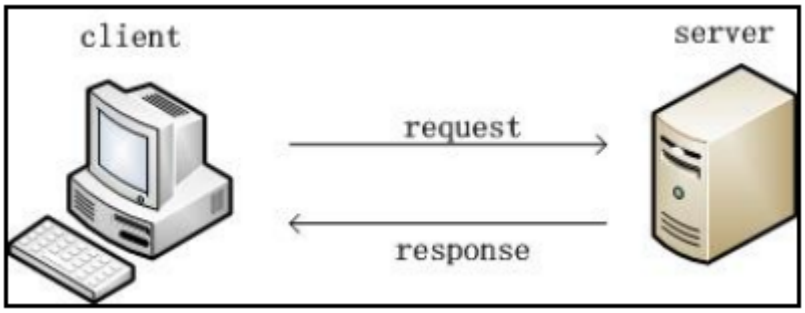

Figura 6 – Comunicação entre o cliente-servidor (Lei, K., Ma, Y., Tan, Z, 2014, p.662).

A máquina do lado do servidor rodava em um sistema operacional Linux Ubuntu 13.04, utilizando um processador Intel i3 3.30GHz com 4 GB de memória RAM e 500 GB de disco.

A máquina do lado do cliente rodava em um sistema Windows7 64-bit, utilizando um processador Intel i3 3.30GHz com 4 GB de memória RAM e 500 GB de disco.

Configuração do Servidor

Alguns serviços e linguagens foram instalados no servidor para realiza os devidos testes comparativos entre as abordagens de desenvolvimento web. Os elementos instalados foram:

- Apache 2.4.9
- PHP 5.5.12
- Python 2.7
- Node.js 0.10

Ferramenta de Teste

Os programas utilizados nos testes comparativos de desempenho e performance entre os desenvolvimentos abordados foram:

- ApacheBench: ApacheBench é uma ferramenta de teste de stress em servidores utilizando o Apache 2.4.
- LoadRunner: LoadRunner é uma ferramenta de teste de performance.

Resultados do Teste Comparativo

Em todas os servidores comparados (Node.js, PHP e Python) foi realizado uma operação de busca no banco de dados, no qual foi possível obter informações como vazão (quantidade de requisições feitas por segundo) e latência (tempo por requisição) das requisições executadas no servidor.

Para obter uma base de dados mais sólida e precisa, foram criados grupos de 10, 100, 200, 500 e 1000 usuários realizando e executando requisições ao servidor de maneira simultânea.

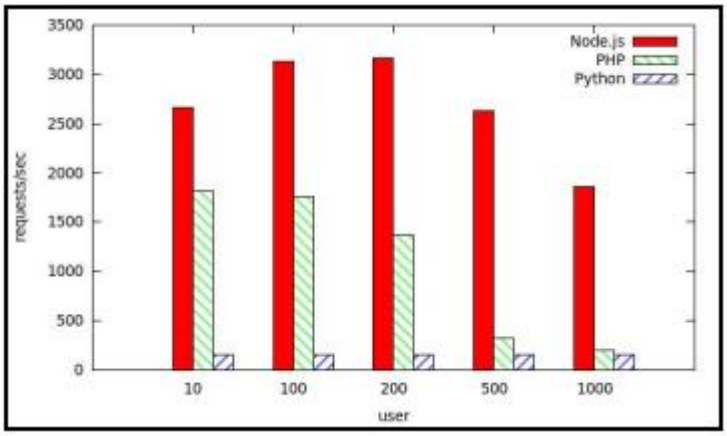

Gráfico 1 – Resultados para a operação de busca no banco de dados em relação a vazão (Lei, K., Ma, Y., Tan, Z, 2014, p.665).

No Gráfico 1 é possível observar que o Node.js consegue realizar mais requisições por segundo em todos os grupos de usuário em relação as demais linguagens, sendo Python a linguagem que consegue menos atender as requisições por segundo ao servidor.

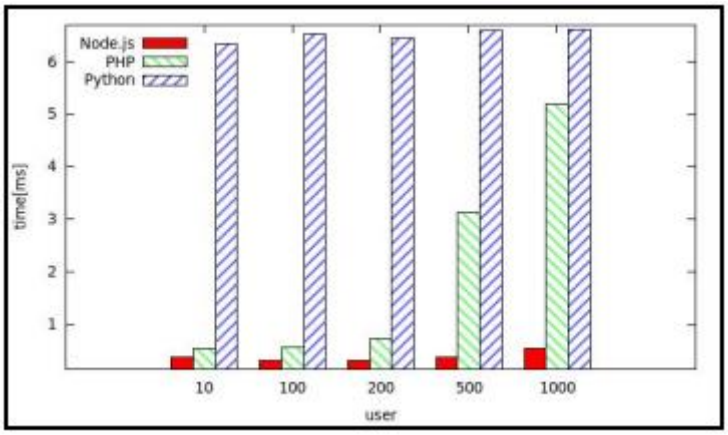

Gráfico 2 – Resultados para a operação de busca no banco de dados em relação a latência (Lei, K., Ma, Y., Tan, Z, 2014, p.665).

No Gráfico 2 é possível observar que o Node.js consegue realizar de maneira mais rápida as requisições para o servidor em todos os grupos de usuário em relação as demais linguagens, sendo mais expressiva e significante a diferença a partir do grupo de 500 usuários. Já Python é a linguagem que realiza de forma mais lenta em relação ao Node.js e ao PHP.

Conclusão

A partir dos dados analisados e comparados no Gráficos 1 e no Gráfico 2, é possível afirmar que um servidor utilizando a linguagem Node.js é mais eficiente em todas as métricas avaliadas (latência e vazão) que as demais linguagens comparadas (PHP e Python) em relação a operação de busca no banco de dados.

Capítulo 2

DESENVOLVIMENTO DO PROJETO

Para o desenvolvimento da aplicação Web nesse projeto, foram utilizadas várias tecnologias ingressadas tanto no lado do servidor, quando no lado do cliente.

2.1 Client-side

As linguagens e tecnologias utilizadas para a realização de uma aplicação do lado do cliente (desenvolvimento isomórfico) foram:

- Javascript é utilizado neste projeto para criação de uma aplicação com abordagem isomórfica, onde tanto o lado do cliente quanto o lado do servidor encontram-se em uma mesma linguagem. A definição completa desta tecnologia encontra-se no Capítulo 1 Seção 1.3.5.1 deste trabalho;
- Node.js é utilizado neste projeto para realizar requisições assíncronas para o lado do servidor, utilizando a linguagem de programação Javascript. Com ele é baseado em Javascript, é possível construir uma aplicação onde o lado do servidor e do cliente estão unidos em um só lugar, em um só código. O objetivo desta biblioteca é otimizar o rendimento, desempenho e escalabilidade das aplicações Web. A definição completa desta tecnologia encontra-se no Capítulo 1 Seção 1.3.5.1.1 deste trabalho;
- Express.js é um framework para Node.js onde ele oferece um conjunto de funcionalidades simples e dinâmicas para o fácil desenvolvimento de aplicações Web. Com a utilização dessa

ferramenta neste projeto, o desenvolvimento de APIs para a aplicação torna-se simples de serem construídas (Express.js, 2017).

2.2 Server-side

As linguagens e tecnologias utilizadas para a realização de uma aplicação do lado do servidor (desenvolvimento tradicional) foram:

- PHP é uma linguagem open source muito utilizada para a criação de desenvolvimentos Web nos dias de hoje. Ele é uma linguagem que consegue ser incorporada dentro do HTML, expandindo várias funcionalidades que o programa pode realizar, sendo assim uma ótima linguagem para construir aplicações para a Web. Ademais, ele é utilizado no projeto para realizar requisições diretamente para o servidor, utilizando o desenvolvimento baseado em Single Page Application (PHP, 2017);
- Laravel é um framework para linguagem de programação PHP, no qual ele foi utilizado neste projeto para construir uma aplicação MVC baseado no desenvolvimento Single Page Application. Este framework possui recursos para deixar as tarefas desenvolvidas mais simples e sustentáveis como a utilização de rotas, bibliotecas completas com muitas funcionalidades para serem utilizadas e sistemas de autenticações inteligíveis.

2.3 Banco de Dados

Ambos os desenvolvimentos (client-side e server-side) compartilham o mesmo banco de dados para a realização do teste comparativo entre as aplicações. O banco de dados escolhido para o projeto foi o Mysql.

O Mysql é um dos bancos de dados mais utilizados no mundo, no qual ele realiza a função de armazenar todos os tipos de dados e informações importantes para a aplicação. Ademais, este banco de dados possui um grande conjunto de funcionalidades avançadas e de gerenciamento de dados para atingir bons resultados em escalabilidade, segurança, confiabilidade e tempo de atividade do Mysql (Mysql, 2017).

Para a realização da conexão ao banco de dados em ambos os desenvolvimentos, foi utilizado como ferramenta o WAMP, um ambiente de desenvolvimento web Windows, no qual ele instala e configura automaticamente tudo o que a aplicação web necessita para funcionar, incluindo o Mysql 5.7.14 utilizado neste projeto (WAMP, 2017).

Conforme a Figura 7 informa, a base de dados utilizada para a realização das buscas tanto do desenvolvimento tradicional quanto do isomórfico foi uma tabela “Produtos” contendo cinco produtos pré-cadastrados.

id	name	description	sku	bars_code	price
1	Produto 1	Descricão Produto 1	1234567891	1234567891	10.00
2	Produto 2	Descricão Produto 2	1234567892	1234567892	20.00
3	Produto 3	Descricão Produto 3	1234567893	1234567893	30.00
4	Produto 4	Descricão Produto 4	1234567894	1234567894	40.00
5	Produto 5	Descricão Produto 5	1234567895	1234567895	50.00
NULL	NULL	NULL	NULL	NULL	NULL

Figura 7 – Tabela “Produtos” do banco de dados Mysql.

Em relação a criação dos produtos, as mesmas quantidades de informações foram enviadas para o banco em ambos os desenvolvimentos.

Com isso, por utilizarem um mesmo cenário de banco de dados nos desenvolvimentos tradicional e isomórfico, é possível garantir uma maior igualdade nos testes e resultados apurados.

2.4 Solução tradicional

Para a realização dos testes no servidor tradicional (utilizando o PHP), foi necessário configurar e implementar os seguintes métodos:

2.4.1 GET

Para a realização da busca de todos os produtos da tabela “Produtos” no banco de dados, foi necessário a implementação do método “index”, nome padronizado pelo próprio Laravel.

Para que a busca seja realizada, é necessário utilizar o método GET do HTTP Request e acessar a URL http://localhost:3001/api/products, criada e configurada tanto no PHP Server quanto no PHP.

```
public function index()
{
    try {
        return response(Models\Product::query()->get(), status: 200);
    } catch (\Exception $error) {
        return response(['error' => trans( id: 'error.unknown')], status: 500);
    }
}
```

Figura 8 – Método GET desenvolvimento tradicional.

Conforme demonstra a Figura 8, ao acessar o método “index” é feita a busca no banco de dados e, assim que ela é realizada com sucesso, os dados são retornados para o cliente com o status 200.

Caso essa solicitação encontre algum erro, esta requisição é retornada com o status 500.

2.4.2 POST

Para a realização da criação de um produto na tabela “Produtos” no banco de dados, foi necessário a implementação do método “store”, nome padronizado pelo próprio Laravel.

Para que a criação seja realizada, é necessário utilizar o método POST do HTTP Request e acessar a URL http://localhost:3001/api/products, criada e configurada tanto no PHP Server quanto no PHP.

Ademais, é necessário enviar através do parâmetro que existe no método POST do HTTP Request os dados do banco como name, description, sku, bars_code, price e seus respectivos valores.

```
public function store()
{
    try {
        return response(Models\Product::create(request()->all()),  status: 201);
    } catch (\Exception $error) {
        return response([ 'error' => trans( id: 'error.unknown') ],  status: 500);
    }
}
```

Figura 9 – Método POST desenvolvimento tradicional.

Conforme demonstra a Figura 9, ao acessar o método “store” é realizado a criação do produto com os dados enviados por parâmetro e, caso tenha sucesso, o produto criado é retornado para o cliente com o status 201.

Caso essa solicitação encontre algum erro, esta requisição é retornada com o status 500.

2.5 Solução isomórfica

Para a realização dos testes no servidor isomórfico (utilizando o Node.js), foi necessário configurar e implementar os seguintes métodos:

2.5.1 GET

Para a realização da busca de todos os produtos da tabela “Produtos” no banco de dados, foi necessário a implementação do método “get”.

Para que a busca seja realizada, é necessário utilizar o método GET do HTTP Request e acessar a URL http://localhost:3000/products, criada e configurada tanto no Node.js Server quanto no Node.js.

```
router.get('/',function(req,res,next){

    products.getAllProducts(function(err,rows){
        if(err)
        {
            res.json(err);
        }
        else
        {
            res.json(rows);
        }
    });
});
```

Figura 10 – Método GET desenvolvimento isomórfico.

Conforme demonstra a Figura 10, ao acessar o método “get” é feita a busca no banco de dados e, assim que ela é realizada com sucesso, os dados são retornados para o cliente.

Caso essa solicitação encontre algum erro, esta requisição é retornada com seus respectivos erros.

2.5.2 POST

Para a realização da criação de um produto na tabela “Produtos” no banco de dados, foi necessário a implementação do método “post”.

Para que a criação seja realizada, é necessário utilizar o método POST do HTTP Request e acessar a URL http://localhost:3000/products, criada e configurada tanto no Node.js Server quanto no Node.js.

Ademais, é necessário enviar através do parâmetro que existes no método POST do HTTP Request os dados como do banco como name, description, sku, bars_code, price e seus respectivos valores.

```
router.post('/',function(req,res,next){

    products.addProducts(req.body,function(err,count){
        if(err)
        {
            res.json(err);
        }
        else{
            res.json(req.body);
        }
    });
});
```

Figura 11 – Método POST desenvolvimento isomórfico.

Conforme demonstra a Figura 11, ao acessar o método “post” é realizado a criação do produto com os dados enviados por parâmetro e, caso tenha sucesso, o produto criado é retornado para o cliente.

Caso essa solicitação encontre algum erro, esta requisição é retornada com seus respectivos erros.

Capítulo 3

TESTES E RESULTADOS

Neste capítulo será apresentado os testes executados em um mesmo banco de dados, utilizando estilos de desenvolvimento SPA distintos (isomórfico e tradicional). Além disso, com os resultados obtidos nos testes, será possível obter uma análise comparativa de desempenho, avaliando os pontos positivos e negativos de cada um dos desenvolvimentos.

3.1 Benchmark

O conceito de Benchmark baseia-se na utilização de programas com o objetivo de realizar testes em determinadas aplicações na área da computação, com o intuito de avaliar positivamente ou negativamente o desempenho de uma determinada tarefa. Este tipo de técnica é muito utilizado para realizar avaliações comparativas entre programas e tarefas, com a finalidade de definir quais são mais viáveis em determinados tipos de aplicações.

Os programas Benchmark são utilizados para avaliar o desempenho, velocidade e a capacidade do sistema. Existe uma grande variedade de programas Benchmark padronizados para industrias e companhias, aplicadas nas mais variadas áreas dentro da computação (Koeppe, Schneider, 2010, p.687).

Existem vários tipos de programas Benchmark, no qual cada um possui uma maneira diferente de avaliação de desempenho em relação ao computador. Alguns Benchmarks são focados em medir e avaliar o desempenho do processador, outros analisam os dispositivos de entrada e saída, ou unem uma ou mais aplicações que se assemelham para realizar ensaios e testes comparativos entre eles (Koeppe, Schneider, 2010, p.687).

Outro fator de difere os programas Bechmark é em relação ao tempo de execução de cada teste e analise, ou seja, dos algoritmos que são utilizados em cada um deles. Alguns testes podem levar alguns minutos para serem testados, já outros podem levar horas, tudo depende da real necessidade da utilização de determinado Benchmark (Koeppe, Schneider, 2010, p.687).

Neste trabalho, será realizado um teste comparativo Benchmark em um mesmo ambiente, utilizando linguagens e arquiteturas distintas de programação e desenvolvimento, respectivamente. O programa Benchmark analisará todo tipo de carregamento e processamento de uma aplicação Web, no qual a partir dos dados recolhidos, demonstrar qual é a metodologia mais eficiente para determinadas aplicações Web.

Existem alguns tipos de programas Benchmark para analisar o desempenho de uma aplicação Web. Alguns desses programas são:

- Benchmark.js: este Benchmark é uma biblioteca baseado em JavaScript no qual suporta realizar testes em aplicações de alta escala e retorna seus respectivos resultados em estatísticas (Benchmark.js, 2017);
- Octane: Octane é um Benchmark criado pela Google que possui o objetivo mensurar o desempenho de aplicações JavaScript executando um conjunto de mecanismos de testes específicos para cada programa baseado em JavaScript (Google, 2017);
- JetStream: utilizando resultados geométricos, ele possui uma combinação de vários Benchmarks JavaScript para avaliar o desempenho da aplicação Web em relação as técnicas de programação utilizadas (JetStream, 2017);
- JMeter: o JMeter é um programa de código aberto em Java e tem como objetivo analisar o comportamento funcional dos testes e medir o desempenho do mesmo. Ele foi desenvolvido exclusivamente para realizar testes em aplicações web (JMeter, 2017).

Para a realização dos testes comparativos entre os desenvolvimentos tradicional e isomórfico que constituem este projeto, o JMeter será utilizado como ferramenta para medir e avaliar o desempenho das aplicações.

3.1.1 Configuração do JMeter

Para a realização dos testes de desempenho, é necessário realizar algumas configurações dentro do programa JMeter. Algumas das principais configurações para o desenvolvimento deste projeto serão demonstradas e exemplificadas nas próximas sessões, sendo elas classificadas em: plano de teste, grupo de usuários, requisição HTTP e relatório de sumário.

3.1.1.1 Plano de Teste

O plano de teste é um dos componentes iniciais para a criação de um teste de desempenho no JMeter, e ele tem como objetivo informar os passos que a ferramenta irá executar quando os testes forem executados (Integração, PAPPE, 2013, p.29).

As principais componentes que poderão ser adicionados ao plano de teste são:

- Listeners: os Listeners são elementos que abstraem e exibem os resultados apurados no teste;
- Assertions: os Assertions são pontos de afirmação para verificar determinadas respostas vindas do elemento Samplers;
- Thread Groups: os Threads Groups são grupos de usuários configurados no sistema para executar determinadas solicitações;
- Samplers: os Samplers são as solicitações que poderão ser feitas para o sistema, incluindo HTTP, FTP, SOAP, JDBC, LDAP e Java.

3.1.1.2 Grupo de Usuários

Na configuração de Grupo de Usuários é possível configurar as ações que um pseudo usuários realizará no sistema. Este componente no JMeter é chamado de “Thread Groups”. Para adiciona-lo no plano de teste, é necessário

ir em “Edit/Add/Threads(Users)/Thread Groups” (Integração, PAPPE, 2013, p.29).

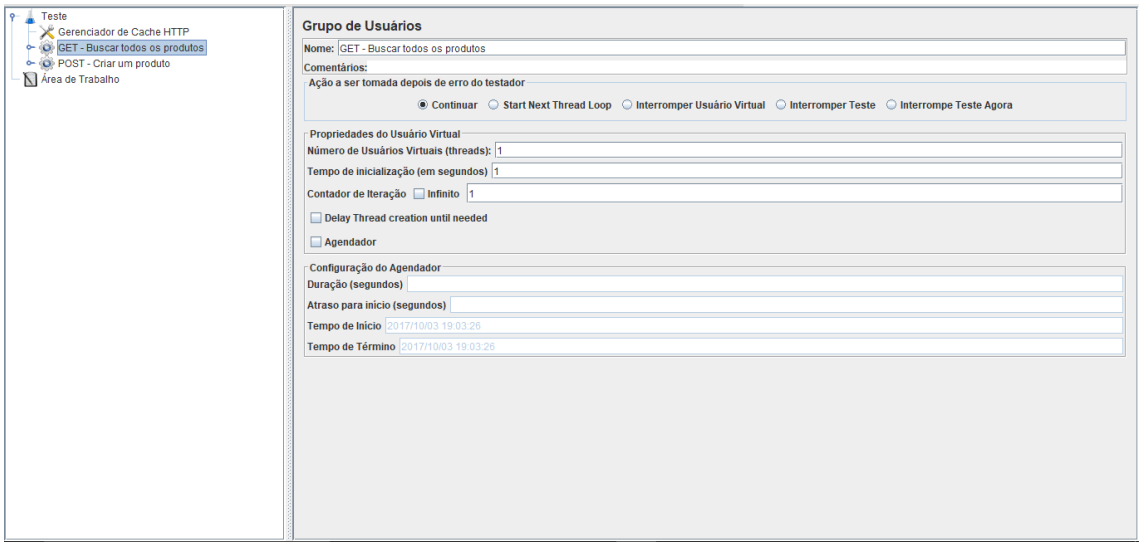

Figura 12 – Configuração dos grupos de usuários.

Como mostra na Figura 12, as principais configurações para a realização dos testes deste projeto são:

- Número de usuários virtuais: quantidade de usuários que irão realizar as solicitações HTTP simultaneamente. Para os testes deste projeto, esse número varia entre 1, 10, 100 e 1000 usuários realizando solicitações simultâneas;
- Tempo de inicialização: intervalo de tempo em que cada usuário irá realizar a solicitação. Para os testes deste projeto, esse número será sempre 1;
- Contador de iteração: quantidade de vezes que a solicitação será feita por usuário. Para os testes deste projeto, esse número será sempre.

3.1.1.3 Requisição HTTP

Nas configurações de Requisições HTTP é possível gerenciar as requisições Web enviadas de um servidor local ou de uma determinada página Web. Ela deve ser inserida dentro de um “Thread Group” (Integração, PAPPE, 2013, p.29).

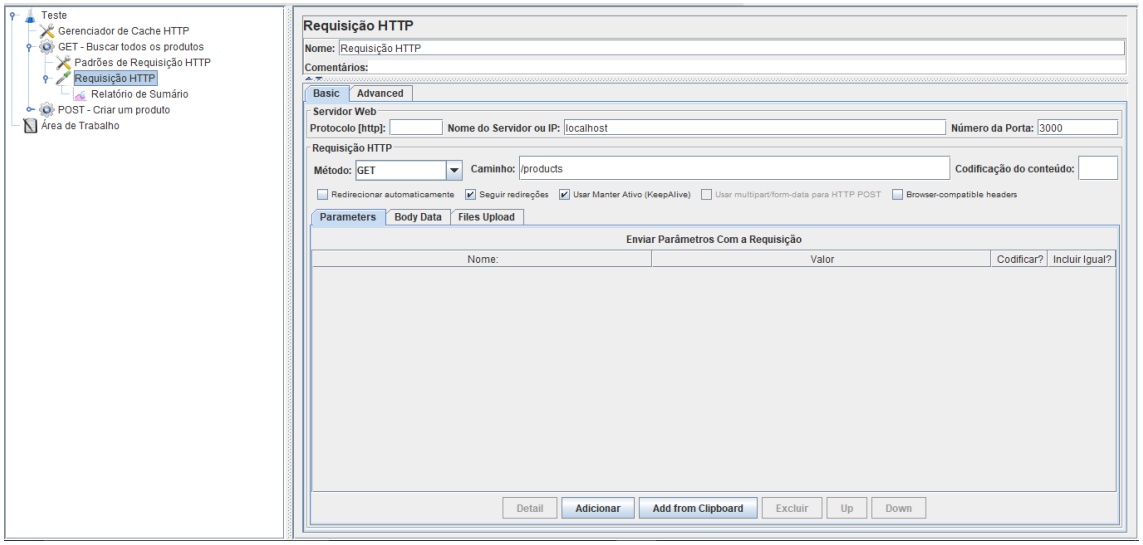

Figura 13 – Configuração da requisição HTTP.

Conforme a Figura 13 demonstra, é possível configurar qual o método que será acionado nesta solicitação (GET, POST, PUT ou DELETE), o servidor/página web que será o alvo dos testes (inserindo a URL ou IP) e a porta de comunicação do servidor/página web.

3.1.1.1 Relatório de Sumário

Para coletar os resultados obtidos no teste, é necessário configurar alguns “Listeners” dentro, por exemplo, das Requisições HTTP, como foi discutido na seção anterior. O “Listener” utilizado neste projeto foi o chamado Relatório de Sumário e ele pode ser adicionado em “Edit/Add/Listerner” (Integração, PAPPE, 2013, p.29).

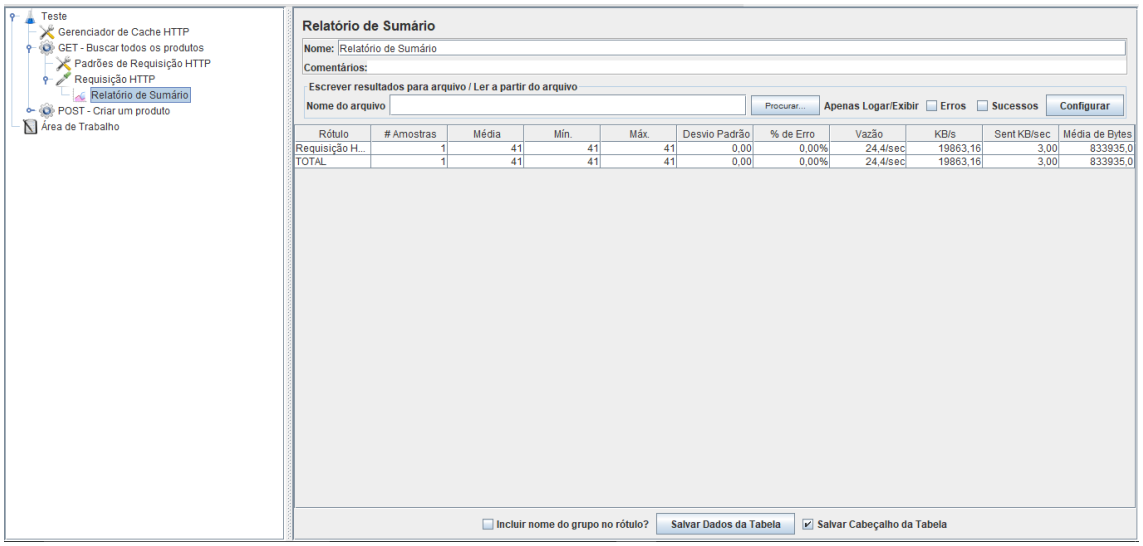

Figura 14 – Configuração do relatório de sumário.

Como mostra a Figura 14, com esta configuração é possível obter os principais resultados de desempenho para este projeto como, por exemplo, as avaliações de latência média, desvio padrão, porcentagem de erro, vazão e tempo de execução total das solicitações.

3.2 Cenários de testes

Nesta seção serão abordadas as informações do cenário que utilizou tanto o desenvolvimento tradicional quanto o isomórfico. O objetivo deste trabalho é realizar a busca e a criação de um produto utilizando o mesmo cenário (banco de dados e lado do cliente) para ambos os desenvolvimentos, tendo como diferencial o lado do servidor.

3.2.1 Desenvolvimento Tradicional

3.2.1.1 Tecnologia Utilizada

As principais tecnologias utilizadas para o desenvolvimento desta aplicação são o PHP Server como o servidor web; PHP como linguagem do lado do servidor; MySql como banco de dados.

Ademais, todos os frameworks utilizados para o desenvolvimento deste programa estão listados no Capítulo 2 Seção 2.2 deste mesmo trabalho.

3.2.1.2 Arquitetura

A arquitetura desta aplicação funciona da seguinte maneira: o cliente faz a interação com a aplicação no qual chamamos de lado do cliente (Javascript Client), onde ele pode realizar uma solicitação para o servidor web (PHP Server) utilizando uma Requisição HTTP para buscar ou criar um produto no banco de dados. Após estes passos, o PHP Server acessa o método de busca/criação solicitado pela requisição HTTP dentro do PHP para, assim, realizar a leitura ou escrita no banco de dados MySql e retornar para a página web.

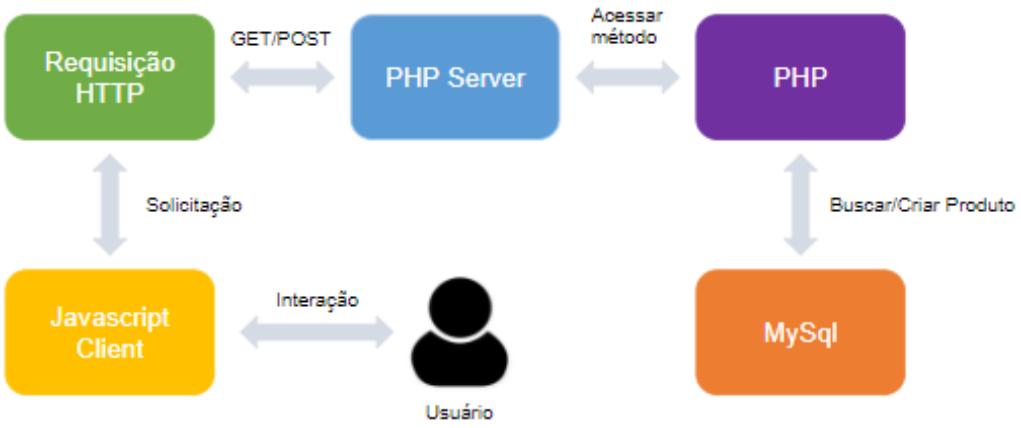

Figura 15 – Arquitetura desenvolvimento tradicional.

3.2.2 Desenvolvimento Isomórfico

3.2.2.1 Tecnologia Utilizada

As principais tecnologias utilizadas para o desenvolvimento desta aplicação são o Node.js Server como o servidor web; Node.js como linguagem do lado do servidor; MySql como banco de dados.

Ademais, todos os frameworks utilizados para a desenvoltura deste programa estão listados no Capítulo 2 Seção 2.1 deste mesmo projeto.

3.2.2.2 Arquitetura

A arquitetura desta aplicação funciona da seguinte maneira: o cliente faz a interação com a aplicação no qual chamamos de lado do cliente (Javascript Client), onde ele pode realizar uma solicitação para o servidor web (Node.js Server) utilizando uma Requisição HTTP para buscar ou criar um produto no banco de dados. Após estes passos, o Node.js Server acessa o método de busca/criação solicitado pela requisição HTTP dentro do Node.js para, assim, realizar a leitura ou escrita no banco de dados MySql e retornar para a página web.

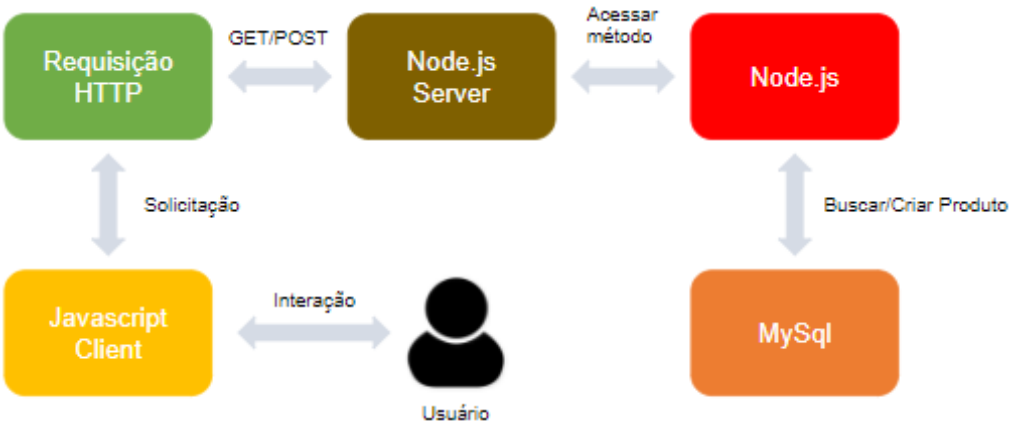

Figura 16 – Arquitetura desenvolvimento isomórfico.

3.3 Testes

Este trabalho foi construído e implementado com o objetivo de realizar uma análise comparativa de uma aplicação web utilizando desenvolvimentos tradicionais e isomórficos em um ambiente SPA.

Para a realização desses testes, foram utilizados os seguintes ambientes e recursos computacionais:

- O servidor é uma máquina com 4 processadores virtuais Intel® Core™ i5-6200U com dois núcleos cada, 8GB de memória e disco de 1TB;
- O sistema operacional utilizado é o Windows 10 (64-bit).

Para obter os resultados deste projeto, foram realizados testes de carga em ambos os servidores (tradicional e isomórfico). Este teste possui o objetivo verificar o comportamento de uma requisição HTTP com uma determinada quantidade de usuários realizando essas solicitações (Integração, PAPPE, 2013, p.29).

Como pode ser visto na Seção 3.1, o software utilizado para realizar os testes de carga foi o JMeter.

Foram elaborados 4 casos de testes no JMeter, dividindo os mesmo por grupos de:

- 1 usuário realizando 1 requisição até o término de todas as requisições.
- 10 usuários realizando 1 requisição até o término de todas as requisições.
- 100 usuários realizando 1 requisição até o término de todas as requisições.
- 1000 usuários realizando 1 requisição até o término de todas as requisições.

Esses quatro grupos de usuários foram testados por 5 vezes. Os resultados apurados e apresentados nas próximas sessões é uma média de cada grupo.

3.3.1 Métricas

As métricas utilizadas neste projeto para realizar a comparações entre os desenvolvimentos tradicionais e isomórficos foram:

- **Latência Média:** Tempo médio, em milissegundos, de resposta para determinado pedido de requisição HTTP;
- **Desvio Padrão:** O desvio padrão apresenta os casos em que determinadas amostras se distanciam do comportamento médio das demais amostras em razão do tempo de resposta. Quanto menor este valor mais consistente é o padrão de tempo das amostras coletadas;

- **Porcentagem de Erro (%):** Porcentagem de erros nas amostras executadas;
- **Vazão:** Quantidade de requisições executadas por segundo;
- **Tempo:** Tempo total para execução de todas as requisições.

3.3.2 Resultados do Desenvolvimento Tradicional

Os resultados dos testes com desenvolvimento tradicional foram organizados separadamente em relação aos métodos GET e POST nas seguintes tabelas:

Tradicional - GET					
Usuários	Latência Média	Desvio Padrão	Porcentagem de erro (%)	Vazão	Tempo
1	1117/ms	2,42	0,00	0,51/sec	1/sec
10	1196/ms	54,77	0,00	3,2/sec	2/sec
100	6592/ms	3205,29	0,00	5,8/sec	12,4/sec
1000	4215/ms	5298,42	79,74	32,6/sec	24/sec

Tabela 1 – Desempenho desenvolvimento tradicional método GET.

Tradicional - POST					
Usuários	Latência Média	Desvio Padrão	Porcentagem de erro (%)	Vazão	Tempo
1	1122/ms	2,42	0,00	0,5/sec	1/sec
10	1239/ms	90,88	0,00	3,2/sec	2/sec
100	6694/ms	3193,07	0,00	4,1/sec	12,4/sec
1000	4334/ms	5585,19	79,66	25,4/sec	25/sec

Tabela 2 – Desempenho desenvolvimento tradicional método POST.

3.3.2.1 Latência

Conforme demonstrado na Tabela 1, Tabela 2, Gráfico 3 e Gráfico 4, nota-se que a diferença entre as latências apuradas com 1 e 10 usuários é praticamente a mesma. Já com a quantidade de 100 usuários realizando as requisições simultaneamente, a sua latência média teve um aumento significativo de aproximadamente 5,5 vezes mais alta que o grupo com 10 usuários.

No grupo de 1000 usuários houve uma queda na latência média em relação ao grupo de 100 por conta da alta taxa de erro que é possível observar no Gráfico 3 e 4, pois a latência dos usuários que não conseguiram realizar com sucesso sua requisição é muito menor do uma requisição que retorna uma resposta completa, diminuindo assim a latência média do total de 1000 requisições.

Esta análise serviu para ambos os métodos (GET e POST) conforme demonstrados no Gráfico 3 e no Gráfico 4, por terem uma base de dados similar.

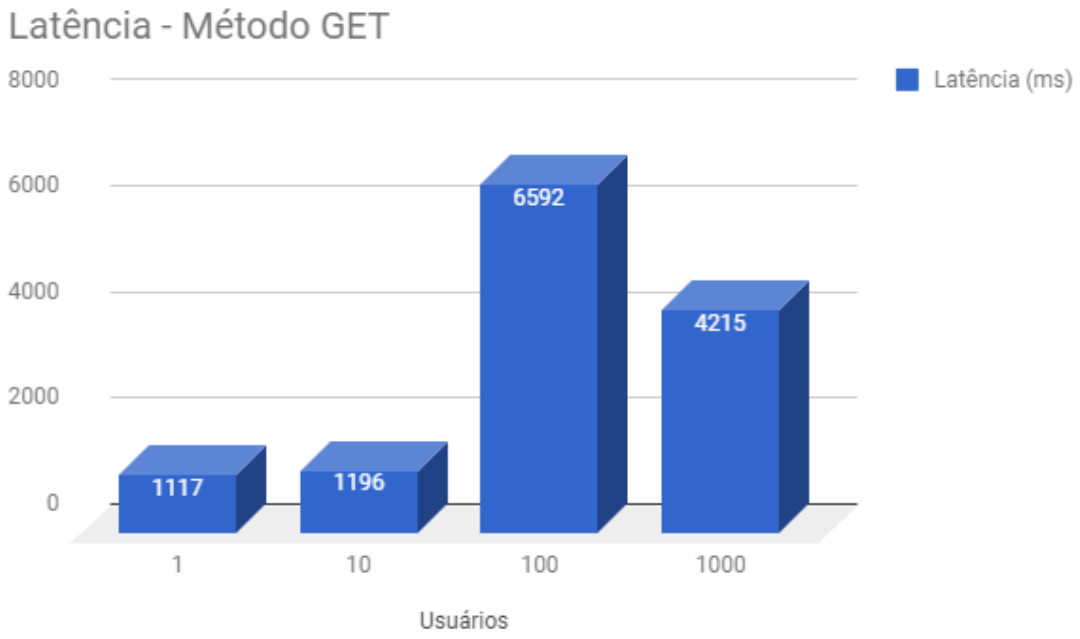

Gráfico 3 – Latência desenvolvimento tradicional método GET.

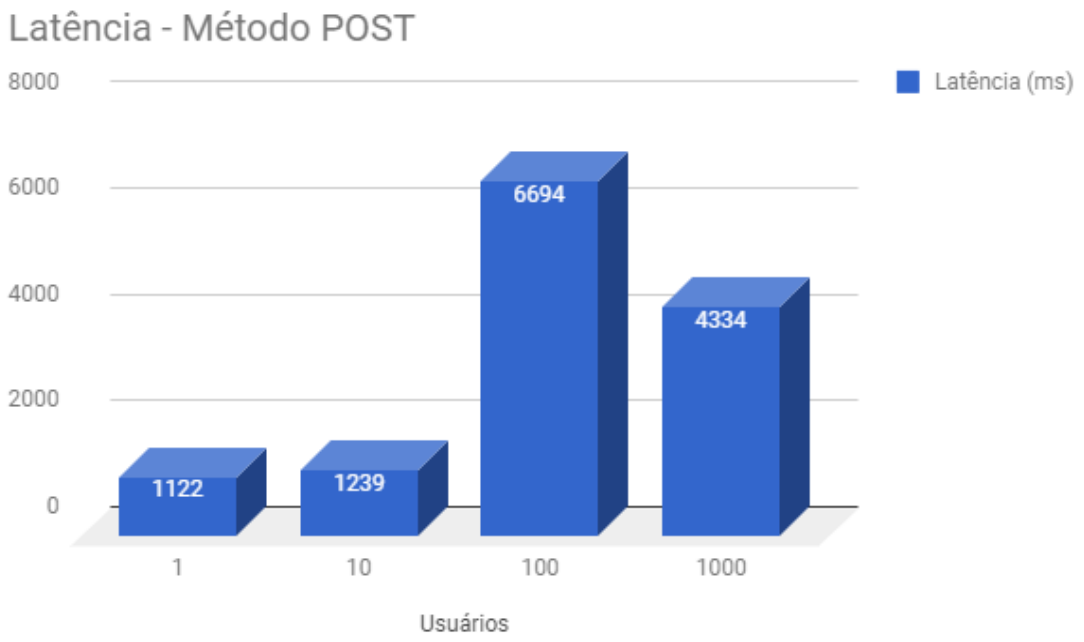

Gráfico 4 – Latência desenvolvimento tradicional método POST.

3.3.2.2 Desvio padrão

Avaliando os valores demonstrados na Tabela 1, Tabela 2, Gráfico 5 e Gráfico 6 nota-se que a diferença entre o desvio padrão apurado 10 usuários é cerca de 23 vezes mais alto que o do grupo com somente 1 usuário. Já com a quantidade de 100 usuários realizando as requisições simultaneamente, o seu desvio padrão teve um aumento de aproximadamente 59 vezes mais alta que o grupo com 10 usuários.

No grupo de 1000 usuários também houve um aumento significativo no desvio padrão em relação ao grupo de 100 de aproximadamente 1,5 vezes.

Esta análise serviu para ambos os métodos (GET e POST) conforme demonstrados no Gráfico 5 e no Gráfico 6, por terem uma base de dados similar. Somente no grupo com 10 usuários que possui uma diferença considerável entre os métodos GET e POST, onde o desvio padrão do método POST é quase o dobro mais alto que o método GET.

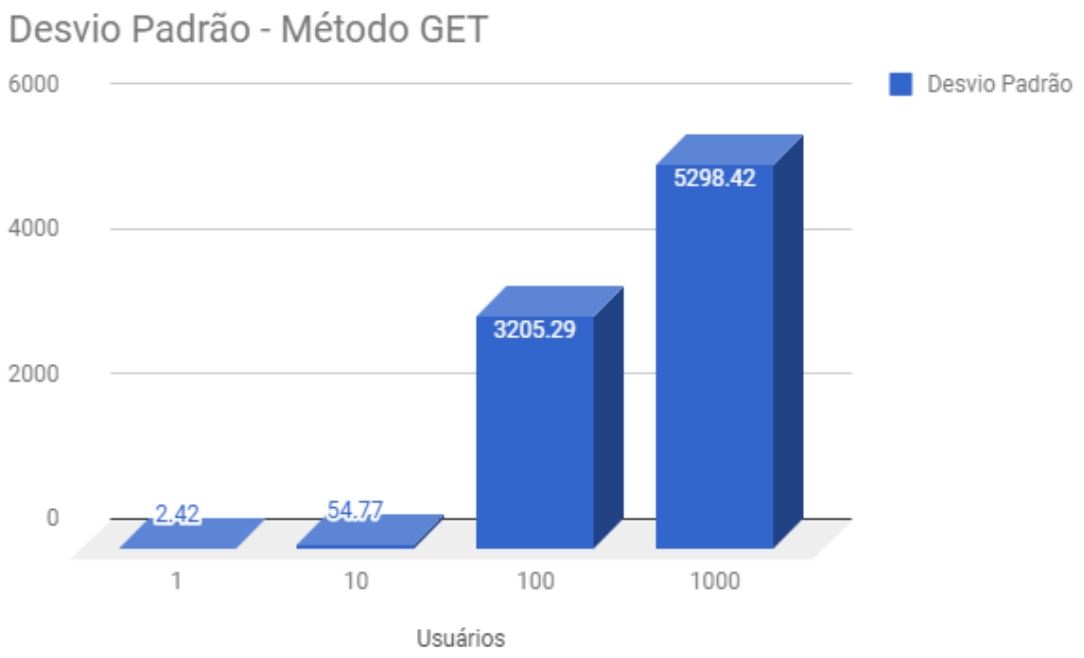

Gráfico 5 – Desvio padrão desenvolvimento tradicional método GET.

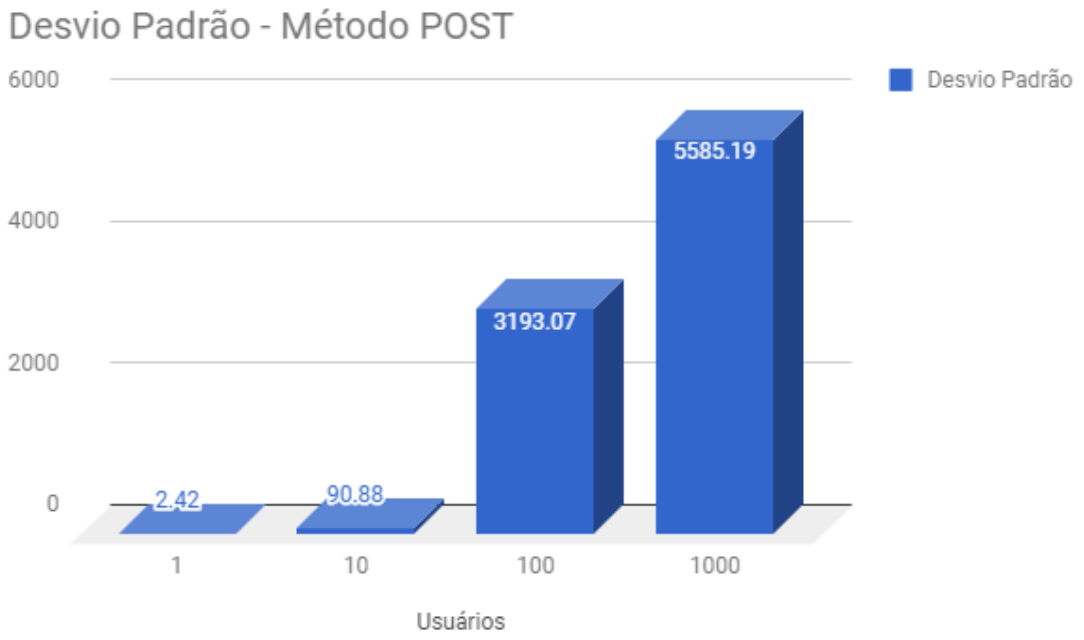

Gráfico 6 – Desvio padrão desenvolvimento tradicional método POST.

3.3.2.3 Porcentagem de erro

Em relação a porcentagem de erro em cada grupo de usuários, conforme demonstra a Tabela 1, Tabela 2, Gráfico 7 e Gráfico 8 os grupos 1,10 e 100 usuários conseguem realizar todas as suas requisições com sucesso em ambos os métodos. Já o grupo com 1000 usuários, cerca de 79,74% das requisições solicitadas não foram completadas no método GET e 79,66% no método POST.

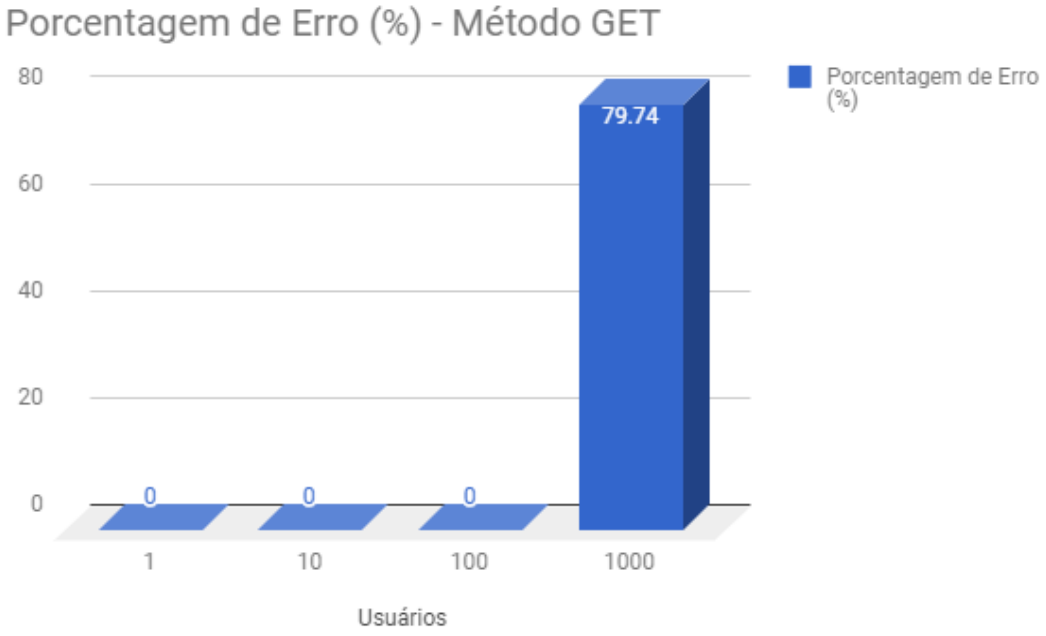

Gráfico 7 – Porcentagem de erro desenvolvimento tradicional método GET.

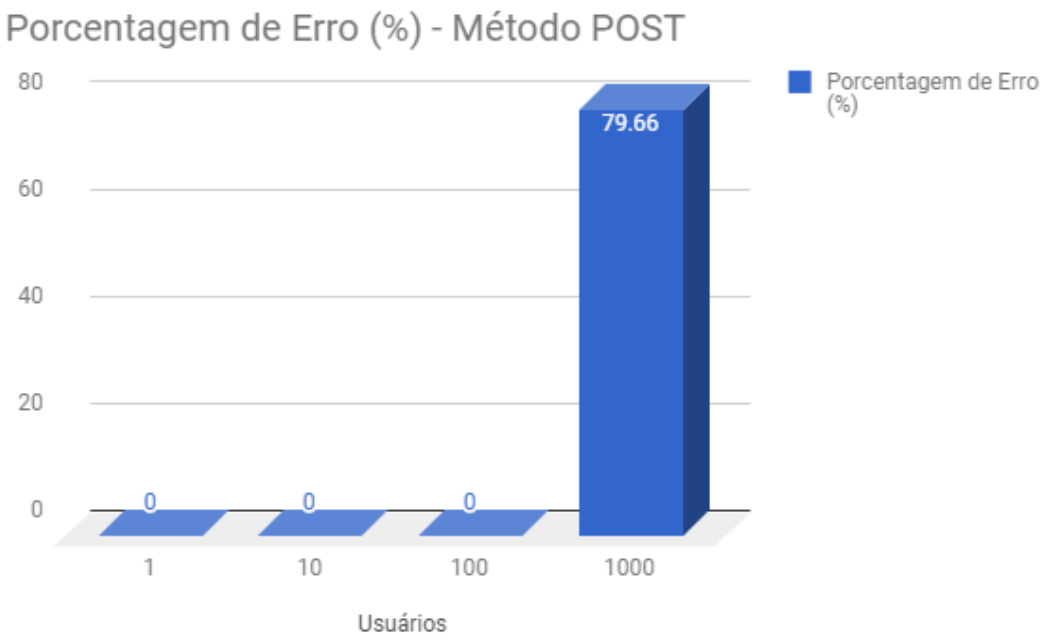

Gráfico 8 – Porcentagem de erro desenvolvimento tradicional método POST.

3.3.2.4 Vazão

Avaliando os valores demonstrados na Tabela 1 e Tabela 2, foi criado o Gráfico 9 e Gráfico 10 onde é possível notar que a vazão entre os grupos de usuários 1, 10, 100 e 1000 continua crescente e similar em ambos os métodos, mesmo com a existência da taxa de erro no grupo de 1000 usuários, conforme mostrado no Gráfico 7 e Gráfico 8.

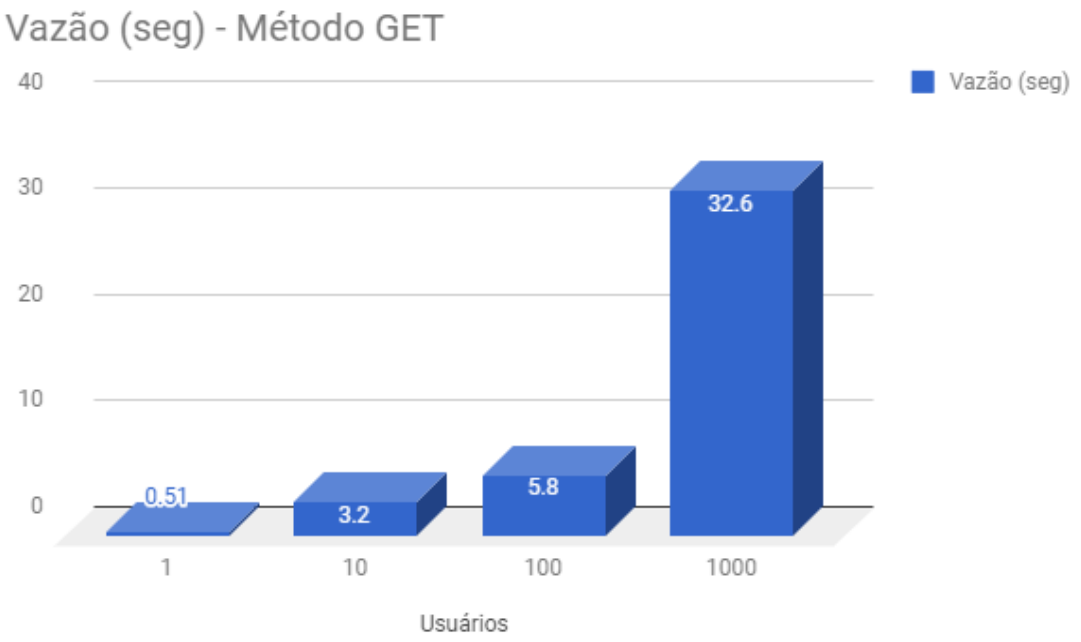

Gráfico 9 – Vazão desenvolvimento tradicional método GET.

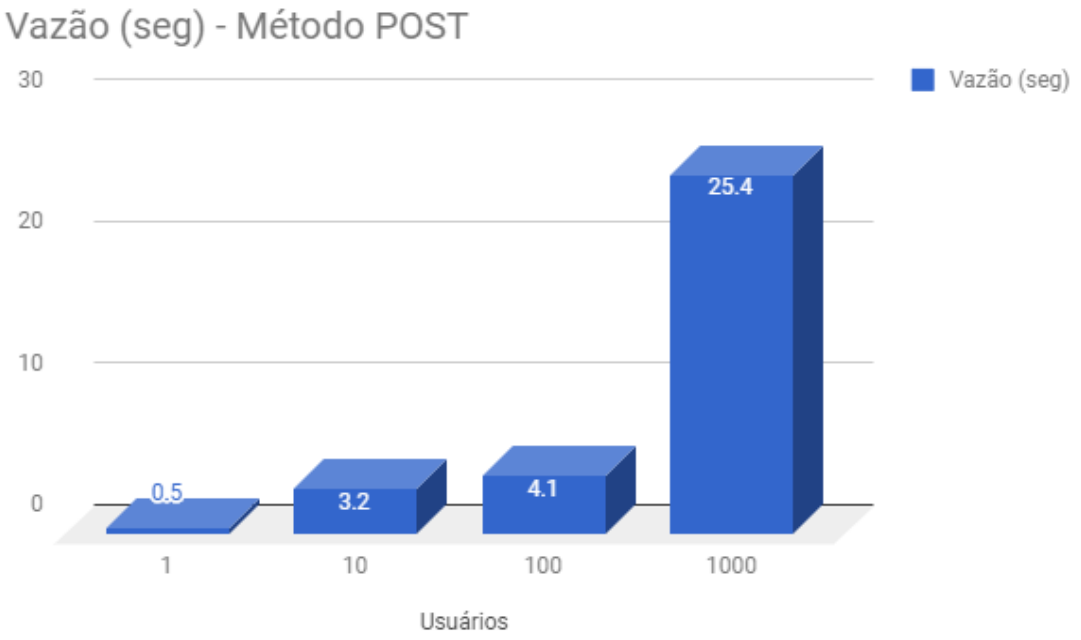

Gráfico 10 – Vazão desenvolvimento tradicional método POST.

3.3.2.5 Tempo

Com os dados recolhidos da Tabela 1 e Tabela 2, foi possível gerar o Gráfico 11 e o Gráfico 12 onde observa-se que em ambos os métodos os valores encontram-se bem similares, sendo mais rápido a realização das requisições com 1 e 10 usuários e mais lenta com 100 e 1000 usuários.

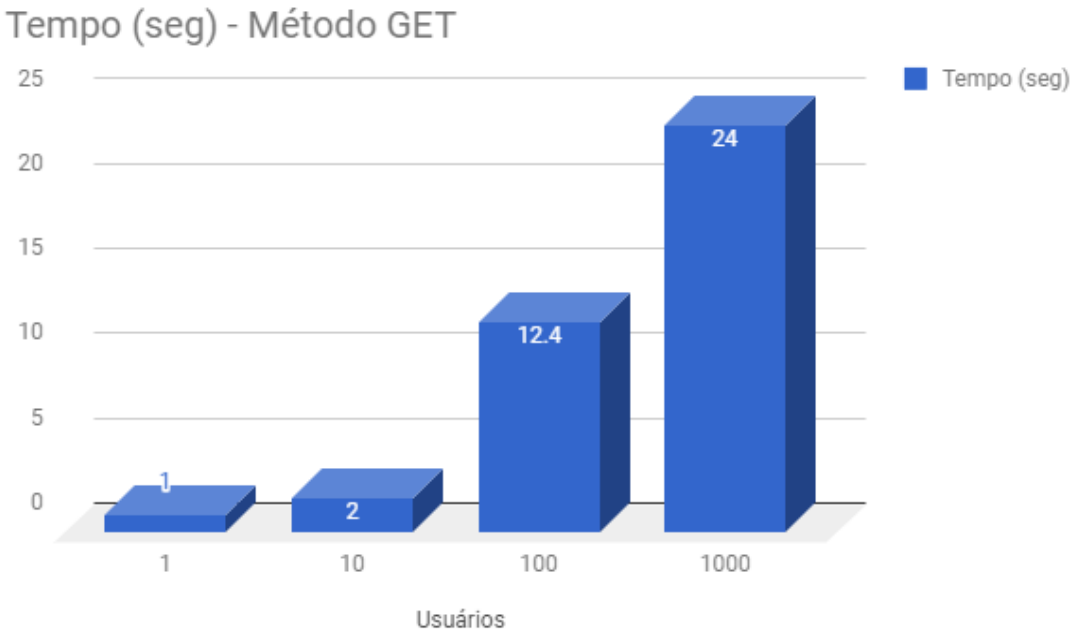

Gráfico 11 – Tempo desenvolvimento tradicional método GET.

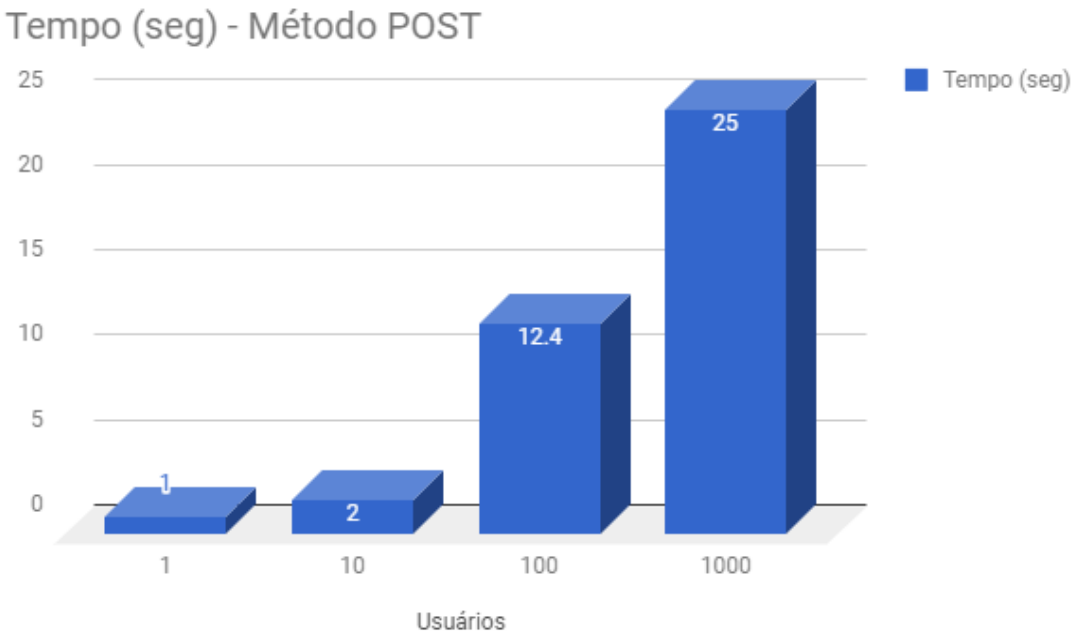

Gráfico 12 – Tempo desenvolvimento tradicional método POST.

3.3.3 Resultados do Desenvolvimento Isomórfico

Os resultados dos testes com desenvolvimento tradicional foram organizados separadamente em relação aos métodos GET e POST nas seguintes tabelas:

Isomórfico - GET					
Usuários	Latência Média	Desvio Padrão	Porcentagem de erro (%)	Vazão	Tempo
1	4/ms	1,17	0,00	1,4/sec	1/sec
10	2/ms	2,67	0,00	6,0/sec	1/sec
100	2/ms	1,02	0,00	56,0/sec	1/sec
1000	545/ms	215,48	0,00	350,2/sec	2/sec

Tabela 3 – Desempenho desenvolvimento isomórfico método GET.

Isomórfico - POST					
Usuários	Latência Média	Desvio Padrão	Porcentagem de erro (%)	Vazão	Tempo
1	5/ms	0,63	0,00	1,0/sec	1/sec
10	4/ms	0,75	0,00	5,4/sec	1/sec
100	4/ms	0,90	0,00	53,7/sec	1/sec
1000	811/ms	289,65	0,00	303,1/sec	2/sec

Tabela 4 – Desempenho desenvolvimento isomórfico método POST.

3.3.3.1 Latência

Conforme demonstrado na Tabela 3, Tabela 4, Gráfico 13 e Gráfico 14, nota-se que não há diferença entre as latências apuradas com 1, 10 e 100 usuários em ambos os métodos. A sua latência só veio a aumentar com 1000 usuários realizando as requisições, no qual ele chega a ser 272 vezes mais alto que o grupo com 100 usuários no método GET e 202 vezes no método POST.

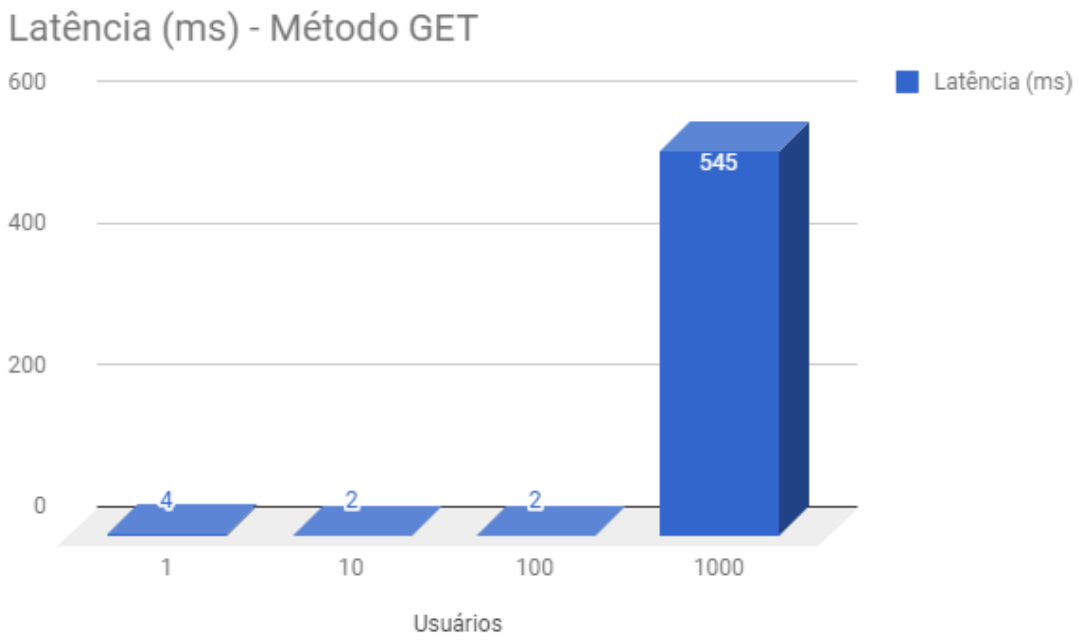

Gráfico 13 – Latência desenvolvimento isomórfico método GET.

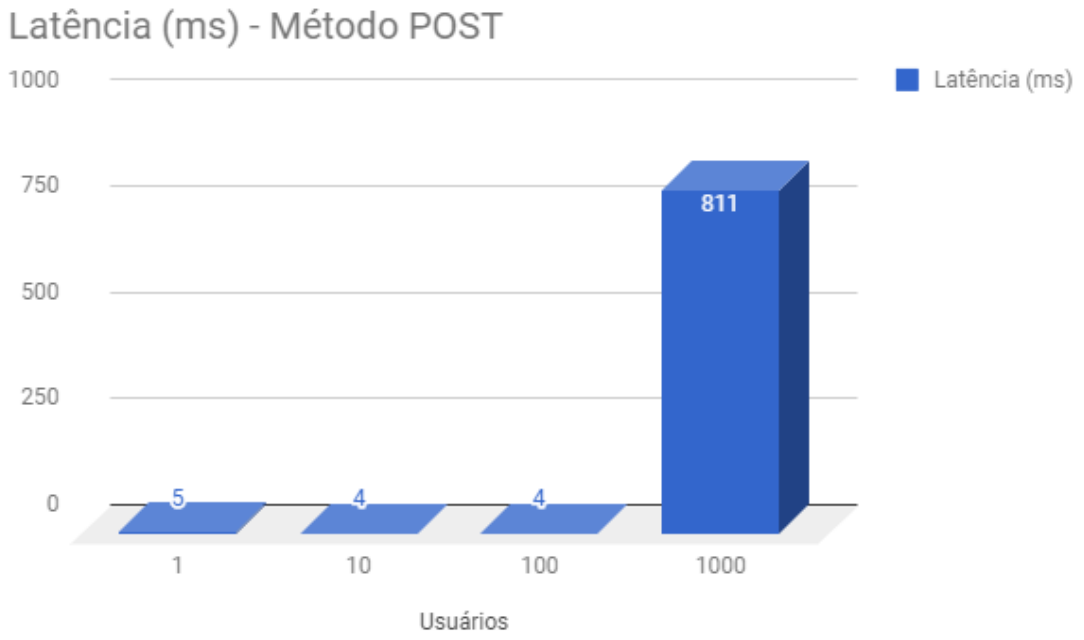

Gráfico 14 – Latência desenvolvimento isomórfico método POST.

3.3.3.2 Desvio padrão

Ao analisar os dados obtidos na Tabela 3 e Tabela 4, foi possível gerar o Gráfico 15 e Gráfico 16, no qual nota-se que não há diferença entre o desvio padrão com 1, 10 e 100 usuários em ambos os métodos. O desvio padrão chegou a aumentar somente com 1000 usuários realizando as requisições.

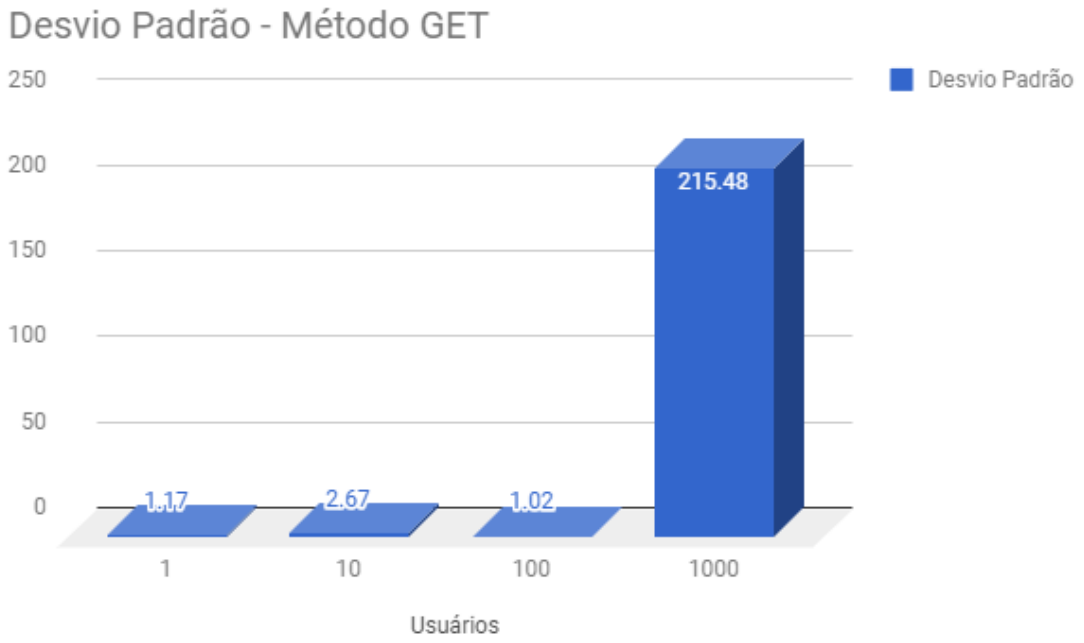

Gráfico 15 – Desvio padrão desenvolvimento isomórfico método GET.

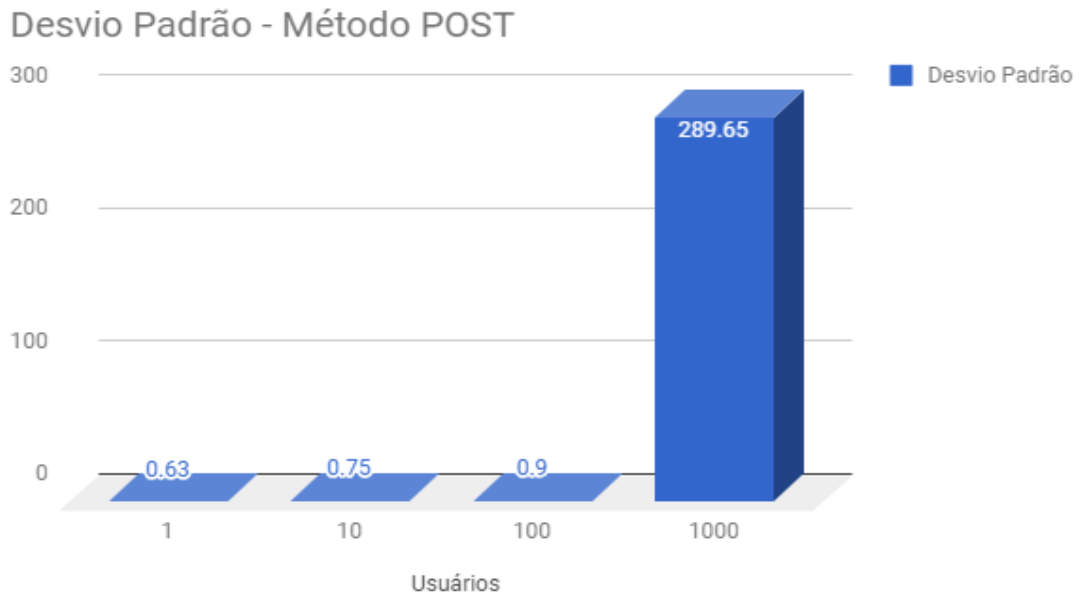

Gráfico 16 – Desvio padrão desenvolvimento isomórfico método POST.

3.3.3.3 Porcentagem de erro

Conforme os resultados obtidos na Tabela 3 e Tabela 4, observa-se no Gráfico 17 e no Gráfico 18 que em todos os grupos de usuários não foi constado nenhum erro durante a execução de suas requisições em ambos os métodos.

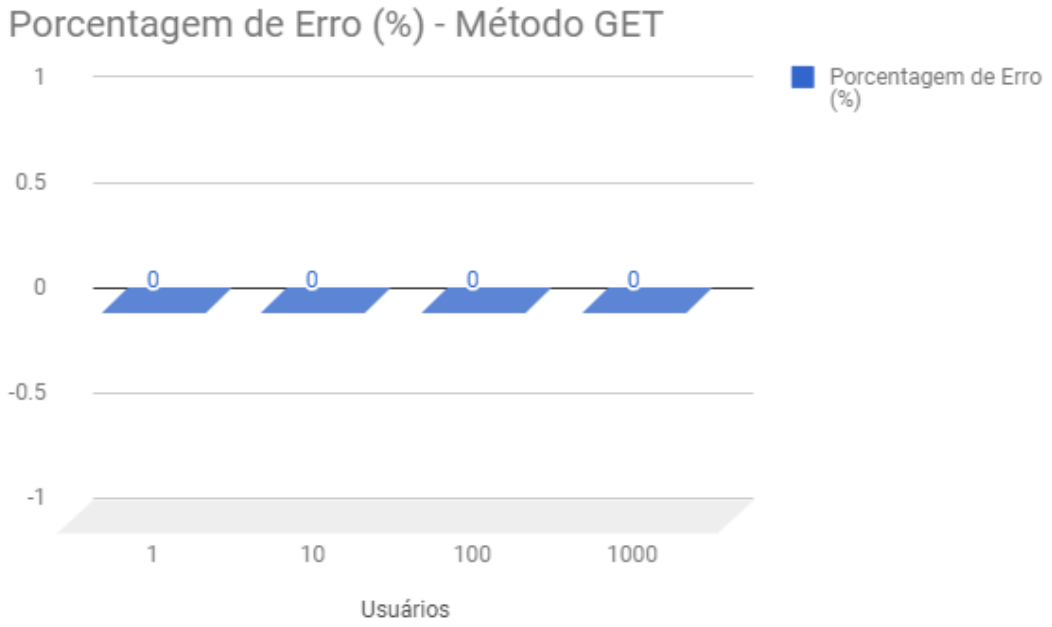

Gráfico 17 – Porcentagem de erro desenvolvimento isomórfico método GET.

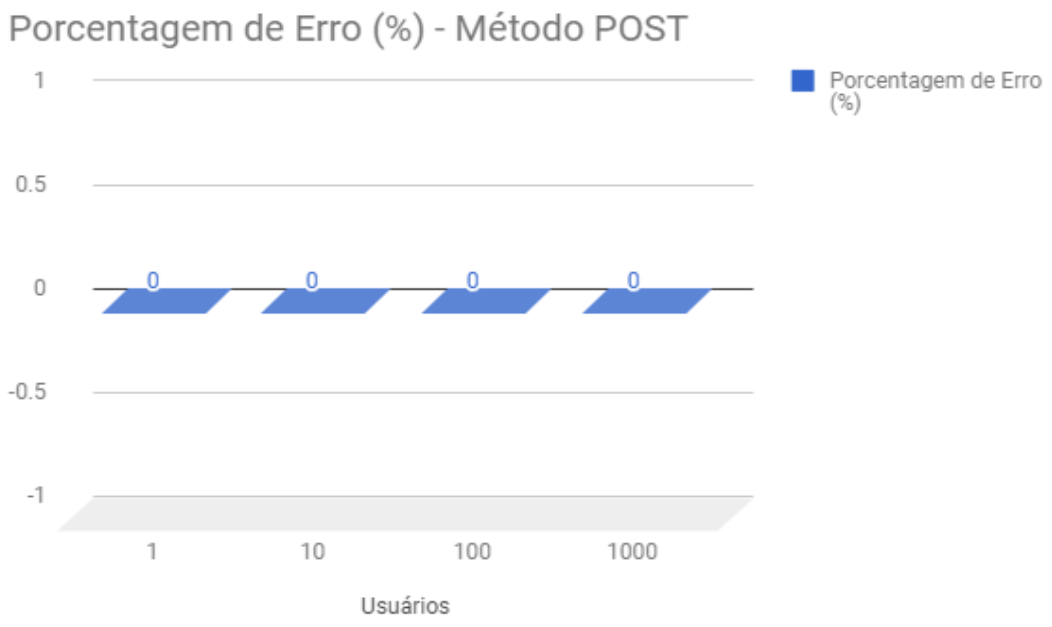

Gráfico 18 – Porcentagem de erro desenvolvimento isomórfico método POST.

3.3.3.4 Vazão

Avaliando os valores demonstrados na Tabela 3 e Tabela 4, foi criado o Gráfico 19 e Gráfico 20 onde é possível notar que a vazão entre os grupos de usuários 1, 10, 100 e 1000 continua crescente e similar em ambos os métodos.

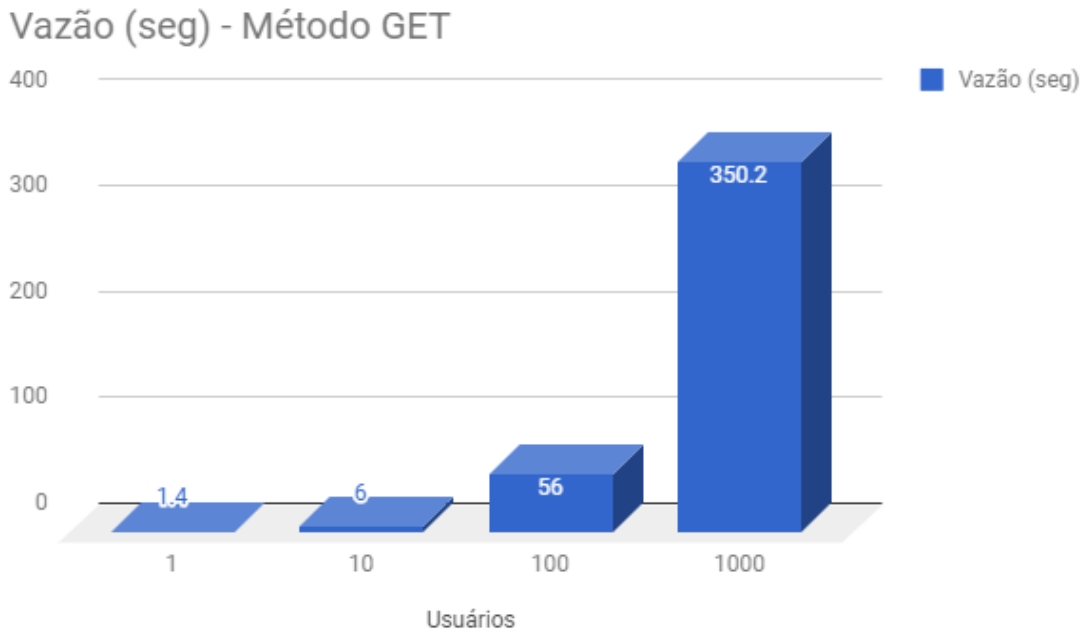

Gráfico 19 – Vazão desenvolvimento isomórfico método GET.

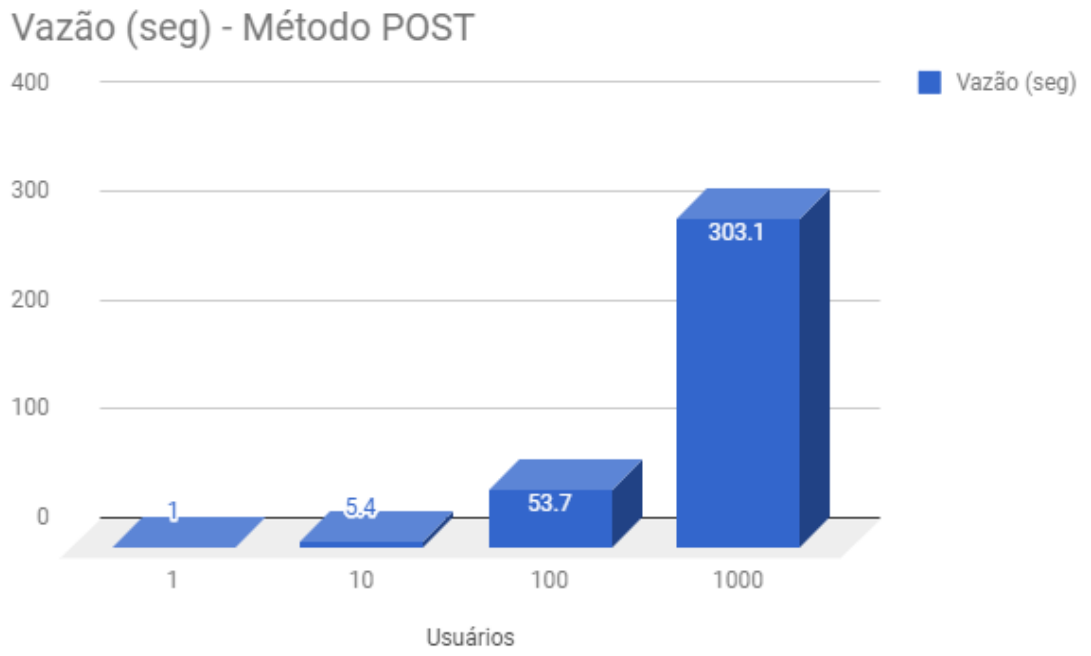

Gráfico 20 – Vazão desenvolvimento isomórfico método POST.

3.3.3.5 Tempo

Com os dados recolhidos da Tabela 3 e Tabela 4, foi possível gerar o Gráfico 21 e o Gráfico 22 onde observa-se que em ambos os métodos os valores encontram-se bem similares, possuindo uma rapidez na realização das requisições de todos os grupos de usuários, com uma leve diferença no grupo com 1000 usuários.

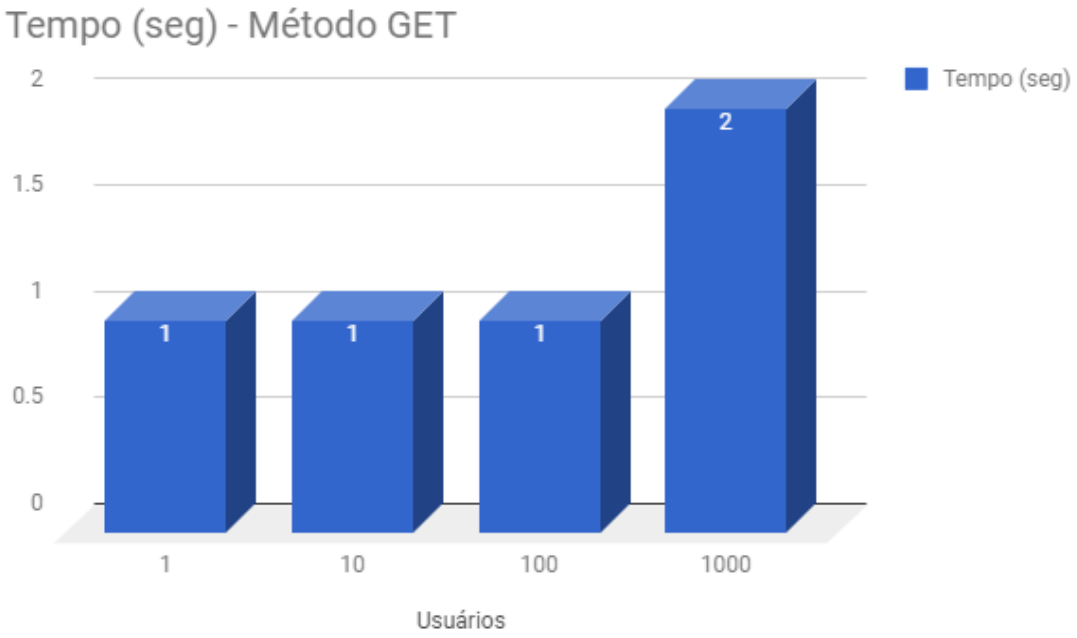

Gráfico 21 – Tempo desenvolvimento isomórfico método GET.

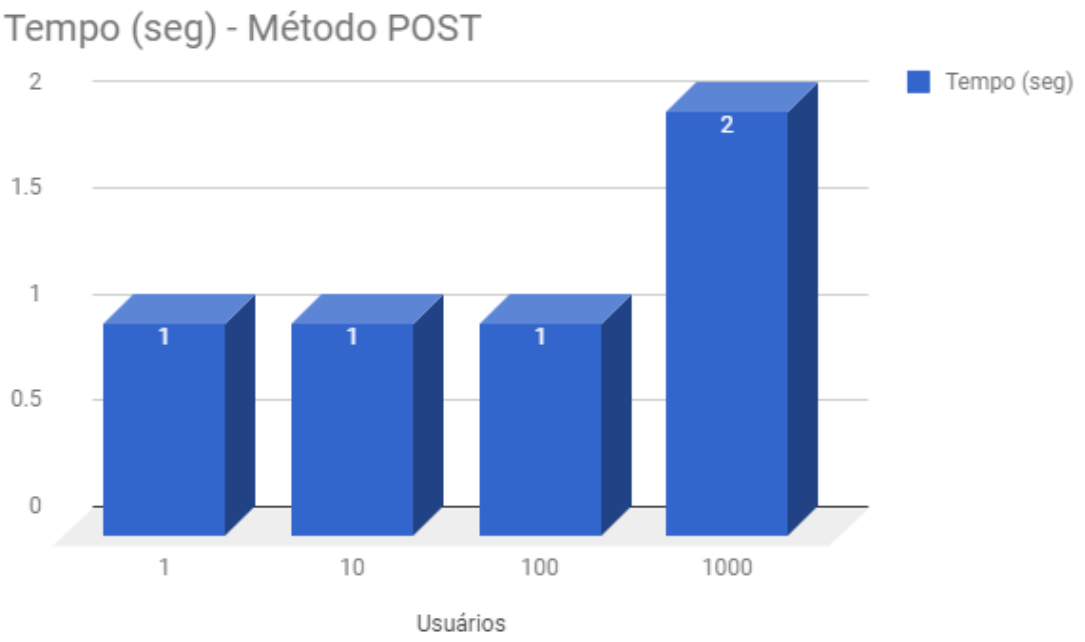

Gráfico 22 – Tempo desenvolvimento isomórfico método POST.

3.4 Análise dos resultados

Nesta seção serão analisados e comparados ambos os desenvolvimentos envolvidos no projeto, para melhor compreensão e visualização dos resultados.

3.4.1 Latência

Em relação a latência média, o desenvolvimento isomórfico é visivelmente melhor que o desenvolvimento tradicional conforme demonstrado no comparativo do Gráfico 23 e Gráfico 24, devido a utilização de técnicas e metodologias como a programação assíncrona e arquitetura não bloqueante.

Portanto, é possível afirmar que neste cenário, o desenvolvimento isomórfico leva grande vantagem em relação ao desenvolvimento tradicional.

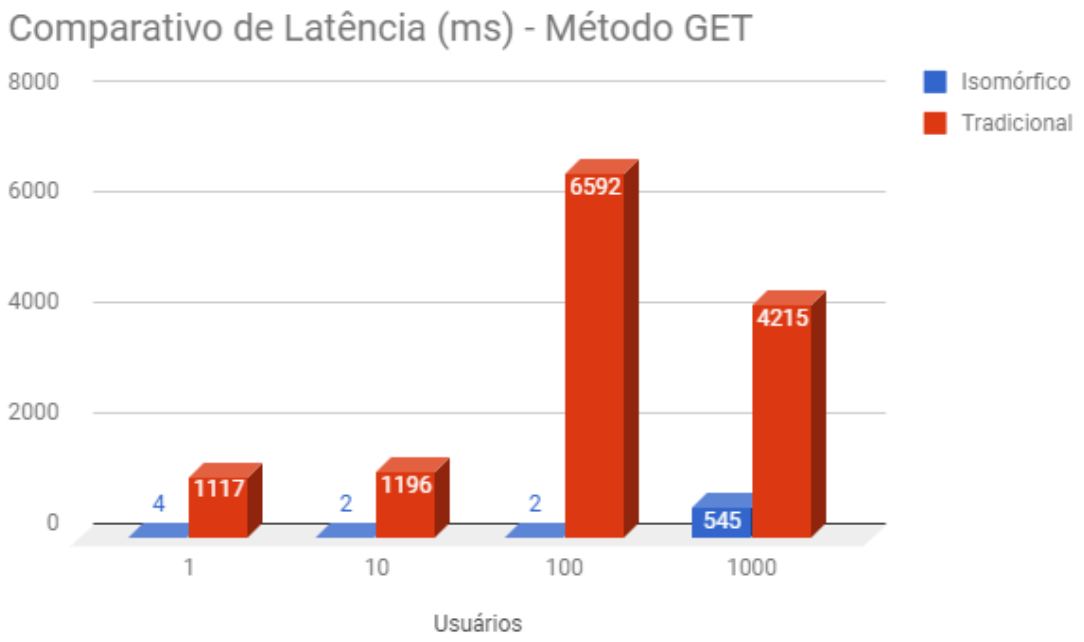

Gráfico 23 – Comparativo de latência método GET.

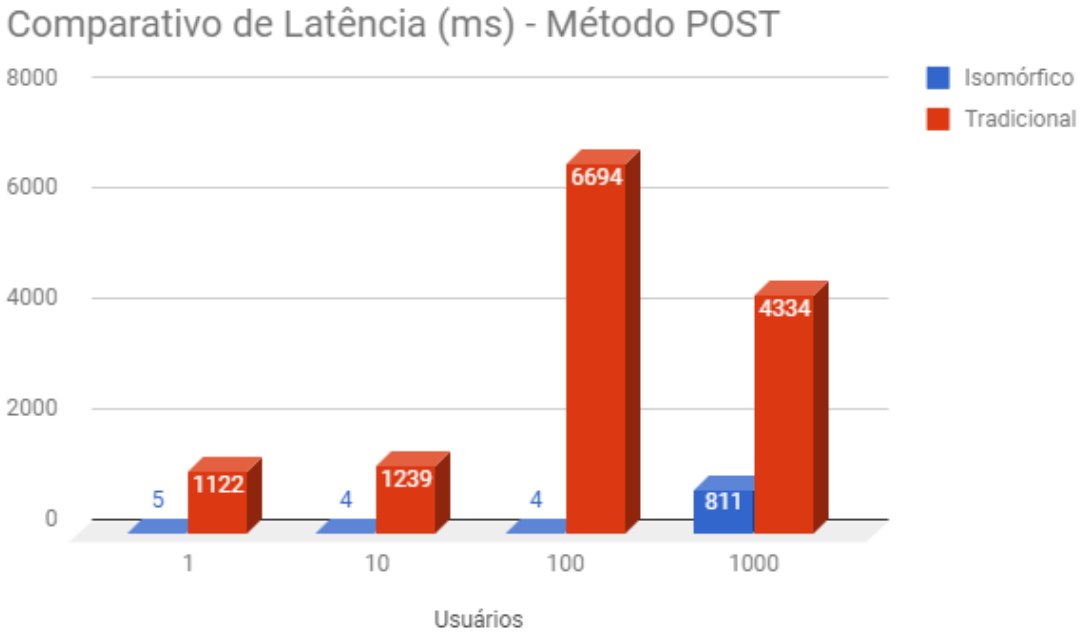

Gráfico 24 – Comparativo de latência método POST.

3.4.2 Desvio padrão

Em relação ao desvio padrão, é possível observar no Gráfico 25 e no Gráfico 26 que no grupo com 1 usuário, o desvio padrão encontra-se instável e igualitário em ambos os desenvolvimentos e métodos. Já nos grupos com 10 usuários, é possível visualizar que o desvio padrão do desenvolvimento tradicional é cerca de 20 vezes mais alto que o do desenvolvimento isomórfico no método GET e no método POST é cerca de 121 vezes mais alto.

Nos grupos com 100 e 1000 usuários, o desenvolvimento isomórfico leva uma vantagem gigantesca em relação ao desenvolvimento tradicional, portanto, é possível afirmar que neste cenário, o desenvolvimento isomórfico possui uma maior estabilidade para a realização das requisições HTTP do que o desenvolvimento tradicional.

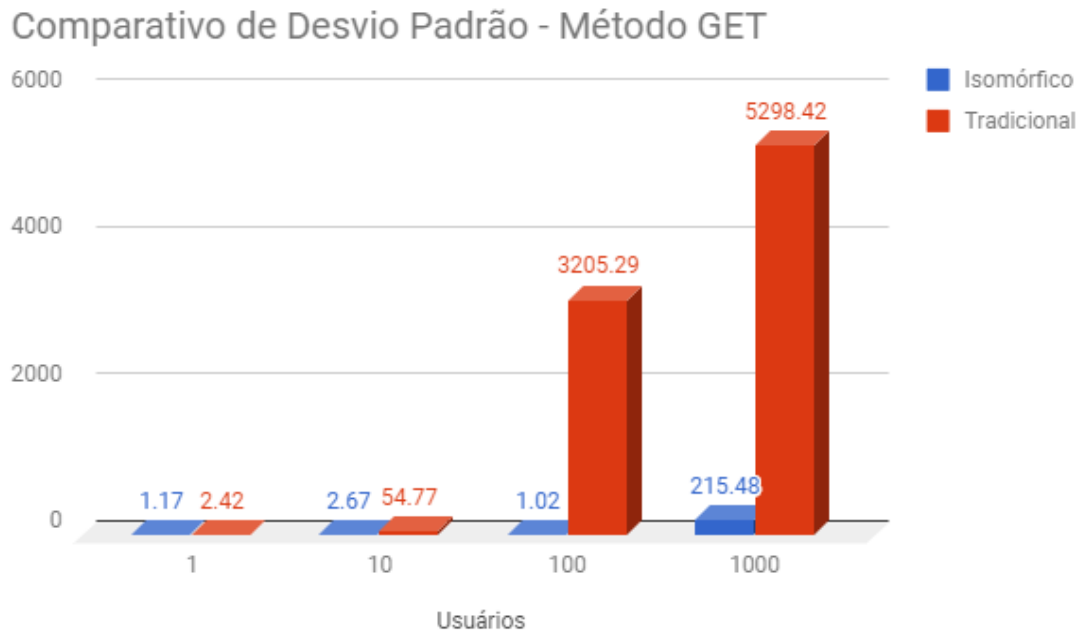

Gráfico 25 – Comparativo de desvio padrão método GET.

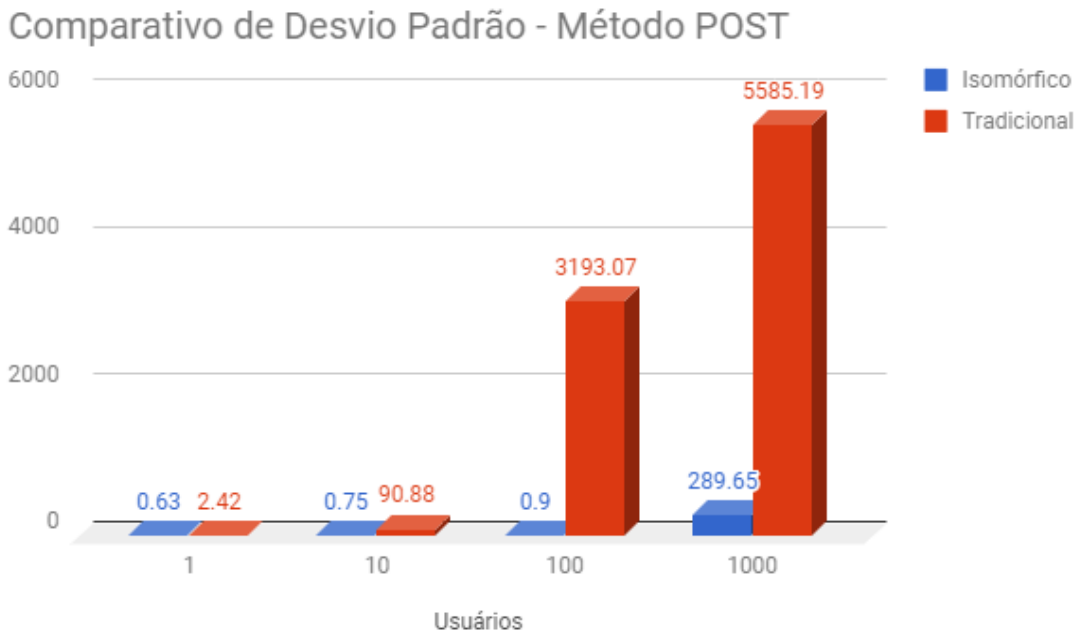

Gráfico 26 – Comparativo de desvio padrão método POST.

3.4.3 Taxa de erro

Como exibido no Gráfico 27 e Gráfico 28, a taxa de erro apresentou um dos resultados mais expressivos. Enquanto o desenvolvimento tradicional teve cerca de 80% de erro no grupo de 1000 usuários, o desenvolvimento isomórfico conseguiu realizar todas as suas requisições com sucesso.

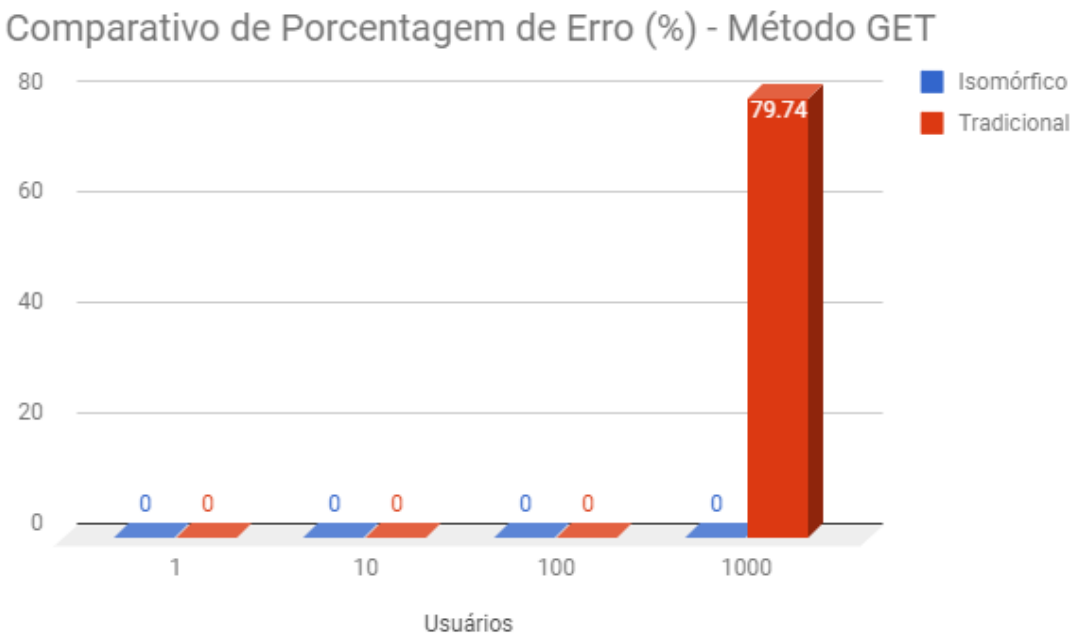

Gráfico 27 – Comparativo de porcentagem de erro método GET.

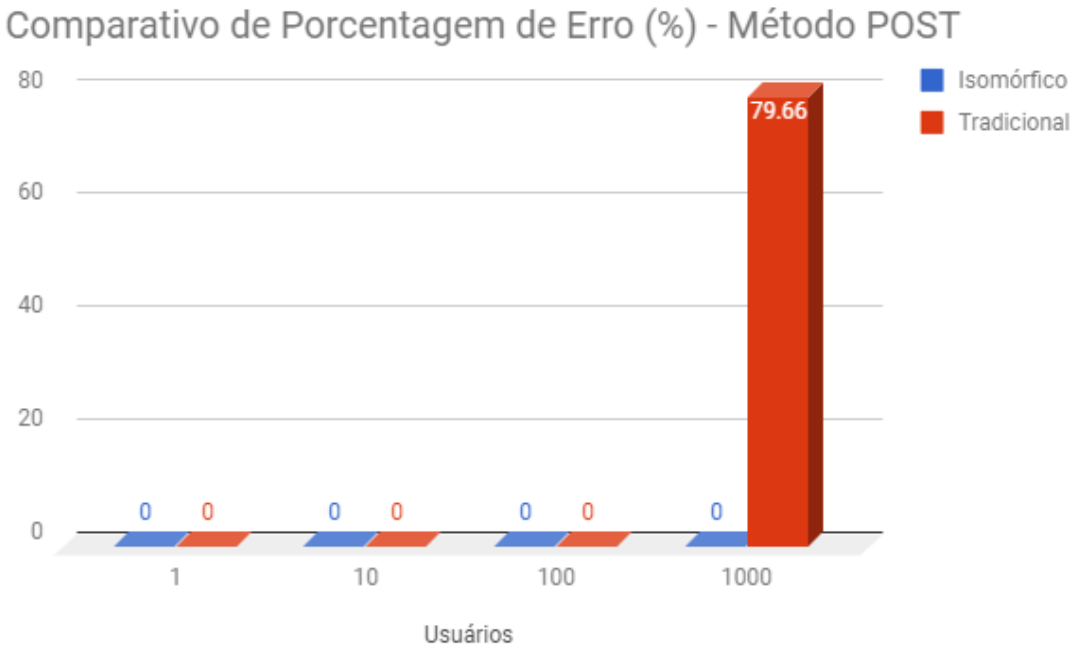

Gráfico 28 – Comparativo de porcentagem de erro método POST.

3.4.4 Vazão

De acordo com o Gráfico 29 e o Gráfico 30, a vazão entre os desenvolvimentos manteve-se relativamente instalável até o grupo de 10 usuários e possuindo uma diferença mais expressiva nos grupos de 100 e 1000 usuários. Sendo assim, o desenvolvimento isomórfico consegue realizar mais requisições por segundo que o desenvolvimento tradicional.

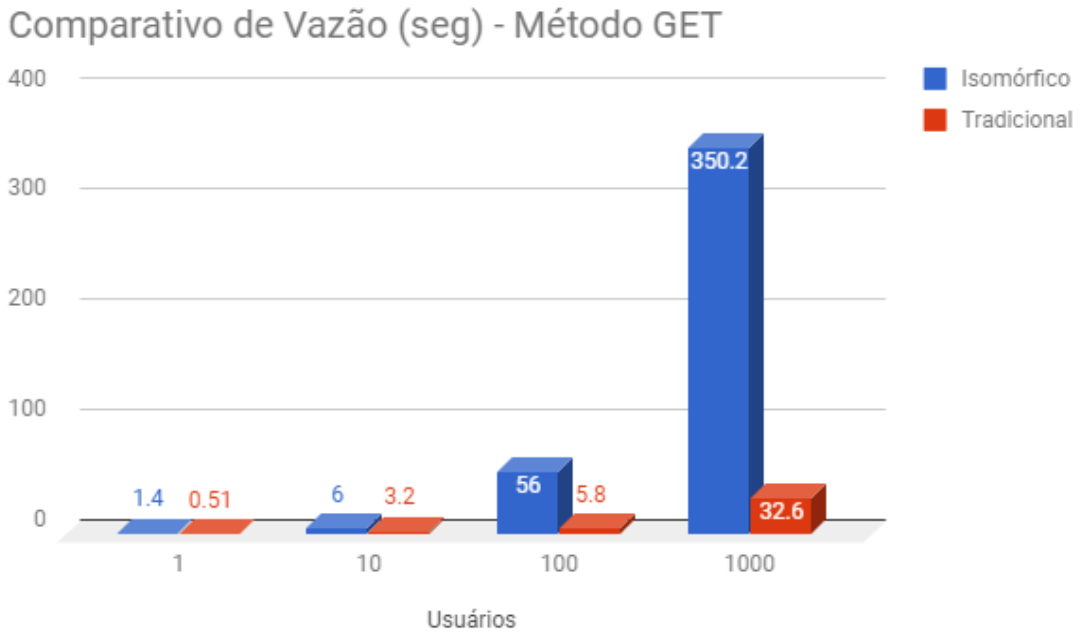

Gráfico 29 – Comparativo de vazão método GET.

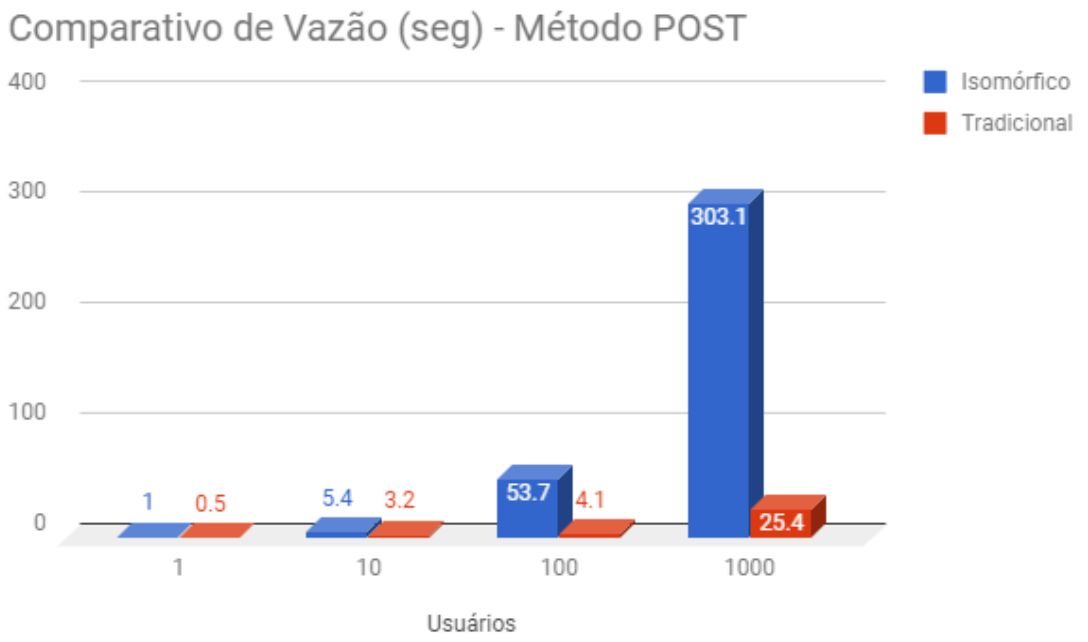

Gráfico 30 – Comparativo de vazão método POST.

3.4.5 Tempo

De acordo com o tempo total para a realização de todas as requisições no qual é possível visualizar no Gráfico 31 e no Gráfico 32, nota-se uma grande diferença em relação aos grupos de 100 e 1000 usuários, onde o desenvolvimento tradicional acaba sendo ineficiente comparado ao isomórfico, gerando um grande problema para o usuário final que utilizará um sistema/aplicação com esse tipo de desenvolvimento.

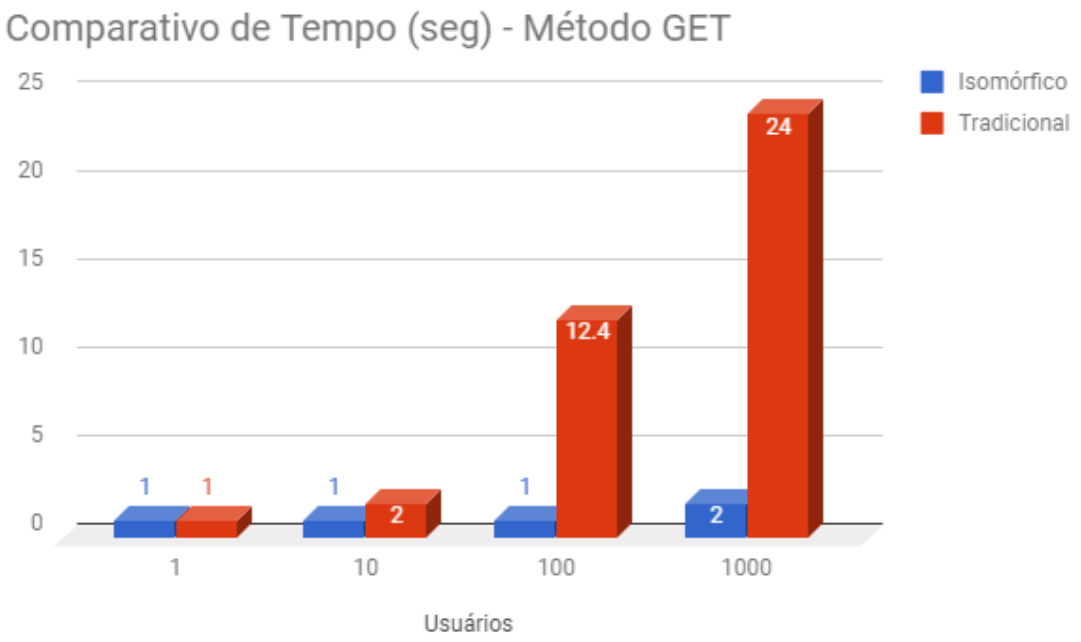

Gráfico 31 – Comparativo de tempo método GET.

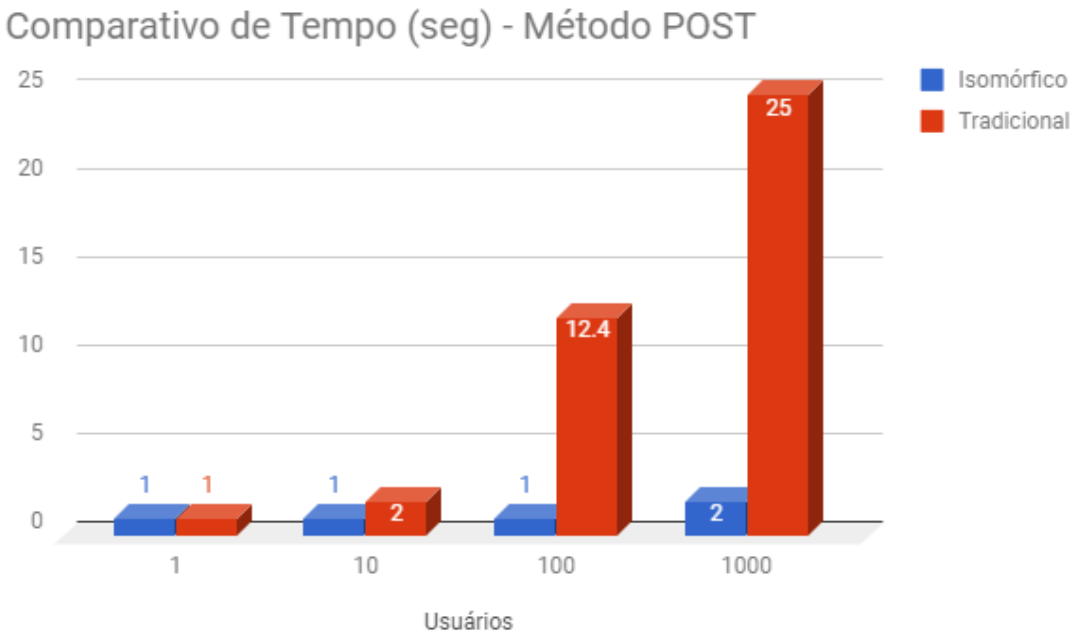

Gráfico 32 – Comparativo de tempo método POST.

Capítulo 4

CONCLUSÕES

Com o grande número de sistemas e aplicações web que são desenvolvidos atualmente pelas empresas de TI em geral, os desenvolvedores devem escolher com sabedoria qual é a tecnologia que se adequa melhor ao perfil do projeto.

Com base na análise comparativa dos resultados envolvendo os desenvolvimentos SPA tradicional e isomórfico, é possível afirmar que para este cenário onde o banco de dados é idêntico em ambas abordagens de programação, um projeto utilizando o desenvolvimento isomórfico leva vantagem tanto na área de desempenho do servidor quanto em custo.

Por mais que o desenvolvimento isomórfico tenha tido uma vantagem em todas as métricas comparadas, não houve diferenças significantes em relação carga de 1 e 10 usuários enviando solicitações simultâneas para o servidor. Sendo assim, o projetista poderá escolher utilizar a metodologia tradicional quanto a isomórfica.

Utilizando o desenvolvimento isomórfico, é possível otimizar e reduzir o tempo das requisições solicitadas ao servidor e assim, diminuir o tempo em que o usuário aguarda sua página ser carregada ou até mesmo a exibição de uma lista e criação de um produto.

Ademais, a aplicação utilizando este tipo de desenvolvimento torna-se mais escalável, no qual é possível atingir uma quantidade maior de usuários requisitando o servidor com uma menor taxa de erro. Sendo assim, é possível reduzir o custo total do projeto, pois não a mais a necessidade de inserir vários servidores para atender as solicitações enviadas para ele.

Com a realização deste trabalho, prevê-se que será possível auxiliar arquitetos e desenvolvedores a tomar a melhor decisão na utilização de cada metodologia e tecnologia em seus respectivos projetos.

Em relação ao trabalho desenvolvido por Kai Lei, Yining Ma e Zhi Tan, onde eles realizaram uma avaliação comparativa entre as linguagens PHP,

Node.js e Python para servidor, é possível afirmar que este projeto não consegue atingir os mesmos resultados obtidos no trabalho de Kai Lei, Yining Ma e Zhi Tan, devido a metodologia, do cenário e dos recursos computacionais utilizados neste projeto.

4.1 Trabalhos Futuros

Com os resultados adquiridos e avaliados neste trabalho, os próximos passos que poderão complementar este trabalho são:

- Realizar testes de carga e stress nos servidores;
- Comparar os desenvolvimentos utilizando uma base de dados mais robusta/pesada e analisar o seu comportamento;
- Avaliação comparativa entre Node.js e NGINX.

REFERÊNCIAS

WEBFOUNDATION. *History of the Web*. Acessado em Mai 07,2017. Disponível em: <https://webfoundation.org/about/vision/history-of-the-web/>.

W3. *W3C*. Acessado em Mai 07,2017. Disponível em: <https://www.w3.org/>.

Tableless. *O que é client-side e server-side?*. Acessado em Mai 14,2017. Disponível em: <http://tableless.github.io/iniciantes/manual/obasico/o-que-front-back.html/>.

Mikowski, M. S., & Powell, J. C. Single Page Web Applications. Manning Publications. p. 1-5, Julho 2012.

Oh, J., Ahn, W. H., Jeong, S., Lim, J., & Kim, T. Automated Transformation of Template-Based Web Applications into Single-Page Applications. In: Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual. p. 292-302

Emer, J. C. *Fluxo de execução assíncrono em JavaScript – Callbacks*. Acessado em Mai 16,2017. Disponível em: <https://tableless.com.br/fluxo-de-execucao-assincrono-em-javascript-callbacks/>.

Wasson, M. *ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET*. Acessado em Mai 16,2017. Disponível em: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx/>.

JSON. *Introducing JSON*. Acessado em Mai 16,2017. Disponível em: <http://www.json.org/>.

Brehm, S. *Isomorphic JavaScript: The Future of Web Apps*. Acessado em Mai 16,2017. Disponível em: <https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc/>.

Ogasawara, T. Workload Characterization of Server-Side JavaScript. In: Workload Characterization (IISWC), 2014 IEEE International Symposium on. p. 13-21

Google. *MVC Architecture*. Acessado em Mai 17,2017. Disponível em: <https://developer.chrome.com/apps/app_frameworks/>.

Node.js. *About Node.js*. Acessado em Mai 17,2017. Disponível em: <https://nodejs.org/>.

Koeppe, F., Schneider, J. Do you get what you pay for? Using Proof-of-Work Functions to Verify Performance Assertions in the Cloud. In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. p. 687-692

Benchmark.js. Benchmark.js. Acessado em Mai 18,2017. Disponível em: <https://benchmarkjs.com/>.

Chromium. Octane 2.0. Acessado em Mai 18,2017. Disponível em: <https://chromium.github.io/octane/>.

Google. The Benchmark. Acessado em Mai 18,2017. Disponível em: <https://developers.google.com/octane/benchmark/>.

JetStream. In Depth Analysis. Acessado em Mai 18,2017. Disponível em: <http://browserbench.org/JetStream/in-depth.html/>.

Mozilla. JavaScript. Acessado em Mai 18,2017. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/>.

Mozilla. What is JavaScript?. Acessado em Jun 05,2017. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript/>.

jQuery. What is jQuery?. Acessado em Mai 18,2017. Disponível em: <https://jquery.com/>.

Bootstrap. About. Acessado em Mai 18,2017. Disponível em: <http://getbootstrap.com/about/>.

Express.js. Express: Fast, unopinionated, minimalist web framework for Node.js. Acessado em Mai 19,2017. Disponível em: <https://expressjs.com/>.

PHP. What is PHP?. Acessado em Mai 19,2017. Disponível em: <http://php.net/manual/en/intro-whatis.php/>.

Mysql. About Mysql. Acessado em Mai 19,2017. Disponível em: <https://www.mysql.com/about/>.

Nodebr. O que é Node.js?. Acessado em Jun 05,2017. Disponível em: <http://nodebr.com/o-que-e-node-js/>.

Lopes, C. O que é Node.js e saiba os primeiros passos. Acessado em Jun 05,2017. Disponível em: <https://tableless.com.br/o-que-nodejs-primeiros-passos-com-node-js/>.

Google. V8. Acessado em Jun 05,2017. Disponível em: <https://developers.google.com/v8/intro/>.

JMeter. Apache JMeter. Acessado em Out 30,2017. Disponível em: <http://jmeter.apache.org/>.

Integração, PAPPE. Manual de Utilização da Ferramenta JMeter. Goiânia, 2013. 29p. Manual Técnico. Instituto de Informática, Universidade Federal de Goiás.

WAMP. WAMP SERVER. Acessado em Nov 11,2017. Disponível em: <http://www.wampserver.com/en/>.

Pereira, C. R. Node.js: Aplicações web real-time com Node.js. 1ª Edição. Casa do Código: Casa do Código, 2017.

DevMedia. Desmistificando o AJAX – Parte II. Acessado em Nov 12,2017. Disponível em: <https://www.devmedia.com.br/desmistificando-o-ajax-parte-ii/9499/>.

DevMedia. Trabalhando com eventos em JavaScript. Acessado em Nov 12,2017. Disponível em: <https://www.devmedia.com.br/trabalhando-com-eventos-em-javascript/28521/>.

NGINX. Welcome to NGINX Wiki. Acessado em Nov 12,2017. Disponível em: <https://www.nginx.com/resources/wiki/>.

Lei, K., Ma, Y., Tan, Z. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python and Node.js. In: 2014 IEEE 17th International Conference on Computational Science and Engineering. p. 661 -668

Anexo A

CÓDIGO FONTE DESENVOLVIMENTO TRADICIONAL

Controller – Product.php

```php
<?php
namespace App\Http\Controllers;

use App\Models;
use Illuminate\Support\Facades\DB;

class Product extends BaseController
{


    public function index()
    {
         try {
             return response(Models\Product::query()->get(), 200);
         } catch (\Exception $error) {
             return response(['error' => trans('error.unknown')], 500);
         }
    }

    public function show($id)
    {
         try {
             $product = Models\Product::find($id);
             if (empty($product)) {
                 return response(['error' => trans('error.not-found')], 404);
             }

             return $this->dataFormat($product);

         } catch (\Exception $error) {
             return response(['error' => trans('error.unknown')], 500);
         }
    }

    public function store()
    {
       try {
             return response(Models\Product::create(request()->all()), 201);
       } catch (\Exception $error) {
          return response([ 'error' => trans('error.unknown') ], 500);
       }
```

```
    }

    public function update($id)
    {
        try {
            $request = request()->all();

            if(empty($request)){
                throw new \Exception('Something went wrong!');
            }

            return response([ 'data' => Models\Product::where('id', $id)-
>update($request) ], 201);

        } catch (\Exception $error) {
            return response([
                'error'    => trans('error.validation'),
                'messages' => $error->validator->errors()
            ], 400);

        } catch (\Exception $error) {
            return response([ 'error' => trans('error.unknown') ], 500);
        }
    }

    public function destroy($id)
    {
        try {
            $product = Models\Product::find($id);

            if (empty($product)) {
                return response(['error' => trans('error.product-not-
found')], 404);
            }

            if(!$product->delete()){
                throw new \Exception('Something went wrong!');
            }

            return response([], 204);

        } catch (\Exception $error) {
            return response(['error' => trans('error.unknown')], 500);
        }
    }
}
```

Model – Product.php

```
<?php
namespace App\Models;

class Product extends BaseModel
{
    protected $fillable = ['name', 'description', 'sku', 'bars_code',
'price'];
}
```

Routes – api.php

```
<?php

$api = app('Dingo\Api\Routing\Router');

$api->version('v1', function () use ($api) {

   $api->group([
      'namespace' => 'App\Http\Controllers'
   ],function() use ($api) {

         $api->resource('products', 'Product', ['only' => [
             'index', 'show', 'store', 'update', 'destroy'
         ]]);
   });
});
```

DB Connection – .env

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db
DB_USERNAME=root
DB_PASSWORD=
```

Anexo B

Código Fonte Desenvolvimento Isomórfico

Routes – products.js

```
var express = require('express');
var router = express.Router();
var products=require('../models/products');

router.get('/',function(req,res,next){

     products.getAllProducts(function(err,rows){
        if(err)
        {
            res.json(err);
        }
        else
        {
            res.json(rows);
        }
    });
});

router.post('/',function(req,res,next){

    products.addProducts(req.body,function(err,count){
        if(err)
        {
            res.json(err);
        }
        else{
            res.json(req.body);
        }
    });
});

 router.post('/:id',function(req,res,next){
  products.deleteAll(req.body,function(err,count){
    if(err)
    {
      res.json(err);
    }
    else
    {
      res.json(count);
    }
```

```
  });
});

router.delete('/:id',function(req,res,next){

        products.deleteProducts(req.params.id,function(err,count){

            if(err)
            {
                res.json(err);
            }
            else
            {
                res.json(count);
            }

        });
});

router.put('/:id',function(req,res,next){

    products.updateProducts(req.params.id,req.body,function(err,rows){

        if(err)
        {
            res.json(err);
        }
        else
        {
            res.json(rows);
        }
    });
});

module.exports=router;
```

Model – products.js

```
var db=require('../dbconnection');

var products={

getAllProducts:function(callback){

return db.query("select * from products",callback);

},
getProductsById:function(id,callback){

    return db.query("select * from products where id=?",[id],callback);
},
addProducts:function(products,callback){
return db.query("insert into products (name, description, sku, bars_code,
price)
values(?,?,?,?,?)",[products.name,products.description,products.sku,products.
bars_code,products.price],callback);
//return db.query("insert into products(Id,Title,Status)
```

```
values(?,?,?)",[products1.Id,products1.Title,products1.Status],callback);
},
deleteProducts:function(id,callback){
    return db.query("delete from products where id=?",[id],callback);
},
updateProducts:function(id,products,callback){
    return  db.query("update products set name=?, description=?, sku=?,
bars_code=?, price=? where
id=?",[products.name,products.description,products.sku,products.bars_code,pro
ducts.price,id],callback);
},
deleteAll:function(item,callback){

var delarr=[];
   for(i=0;i<item.length;i++){

       delarr[i]=item[i].Id;
   }
   return db.query("delete from products where id in (?)",[delarr],callback);
}
};
module.exports=products;
```

DB Connection – dbconnection.js

```
var mysql=require('mysql');
var connection=mysql.createPool({
    host:'localhost',
    user:'root',
    password:'',
    database:'db'
});
module.exports=connection;
```