CENTRO UNIVERSITÁRIO BRASILEIRO - UNIBRA
CURSO DE GRADUAÇÃO TECNOLÓGICA EM
REDES DE COMPUTADORES

ARIADNE MILENA DA LUZ DA SILVA

Frameworks PHP
Um comparativo entre CakePHP, CodeIgniter e Laravel

RECIFE/2020

ARIADNE MILENA DA LUZ DA SILVA

Frameworks PHP
Um comparativo entre CakePHP, CodeIgniter e Laravel

Artigo apresentado ao Centro Universitário Brasileiro – UNIBRA, como requisito parcial para obtenção do título de Tecnólogo em Redes de Computadores.

Professor Orientador: Prof. MSc. Renan Costa Alencar

RECIFE/2020

S586f Silva, Ariadne Milena da Luz da
Frameworks PHP: Um comparativo entre CakePHP, Codelgniter e Laravel. / Ariadne Milena da Luz da Silva. - Recife : O Autor, 2020.

18 p.

Orientador(a): Renan Costa Alencar

Trabalho de Conclusão de Curso (Graduação) - Centro Universitário Brasileiro – UNIBRA. Graduação Tecnológica em Redes de Computadores, 2020.

1. PHP. 2. Frameworks. 3. Desenvolvimento Web. I. Centro Universitário Brasileiro - UNIBRA. II. Título

CDU: 004.7

ARIADNE MILENA DA LUZ DA SILVA

Frameworks PHP
Um comparativo entre CakePHP, CodeIgniter e Laravel

Artigo aprovado como requisito parcial para obtenção do título de Tecnólogo em Redes de Computadores, pelo Centro Universitário Brasileiro – UNIBRA, por uma comissão examinadora formada pelos seguintes professores:

_____

Prof.º Prof. MSc. Renan Costa Alencar

Professor(a) Orientador(a)

_____

Prof.º Prof. MSc. Adilson da Silva

Professor(a) Examinador(a)

_____

Prof.º Prof. MSc. Bruno Roberto Silva

Professor(a) Examinador(a)

Recife, ___/___/____

NOTA:_____

Dedico este trabalho aos meus avós, por terem sido tudo para mim,
ao meu querido Paulo, companheiro e amigo de todas as horas
e a todos que me apoiaram nesta jornada.

AGRADECIMENTOS

À Paulo, por todo apoio e companheirismo. Todas as palavras aqui ditas seriam insuficientes para expressar todo o meu carinho e gratidão a você.

À Patrícia, por possibilitar o início da minha vida acadêmica.
À Socorro e família, por todo o acolhimento e carinho.

Aos meus queridos amigos, os quais não serão individualmente mencionados a fim de incluir todos, agradeço por todo o apoio, carinho e diversão que vocês me proporcionaram durante o curso. Que a nossa amizade dure por muitos anos a vir.

“Dificuldades preparam pessoas comuns
para destinos extraordinários”
(C.S Lewis)

SUMÁRIO

Frameworks PHP: Um comparativo entre CakePHP, CodeIgniter e Laravel

Ariadne Milena da Luz da Silva
Renan Costa Alencar[1]

Resumo: Os *frameworks* são uma ferramenta de desenvolvimento que possibilitam o desenvolvimento de uma aplicação de forma mais ágil e flexível ao permitir a reutilização e exibição de código pré desenvolvido. Este trabalho propõe-se a introduzir conceitos importantes para o desenvolvimento com frameworks PHP e fazer um comparativo entre 3 *frameworks* PHP populares, CakePHP, CodeIgniter e Laravel a partir da criação de aplicações back-end para realizar a criação e busca de pedidos e seus produtos no banco de dados. A partir das aplicações utilizadas, foi realizado o levantamento dos tempos para a execução das aplicações em diferentes proporções seguidas da análise das mesmas, tendo o *framework* CodeIgniter o melhor desempenho nos quesitos avaliados. Estudantes e desenvolvedores podem guiar-se a partir do material apresentado neste trabalho introdutório para o aprofundamento em quaisquer dos *frameworks* abordados.

Palavras-chave: PHP. *Frameworks*. Desempenho. Desenvolvimento web.

1 INTRODUÇÃO

Desenvolver uma aplicação comercial é um processo que requer o cumprimento de prazos, planejamento compreensivo de aplicação dos recursos disponibilizados e entendimento das necessidades do cliente, de acordo com Gonçalves, 2020.

Para um desenvolvedor, a escolha e utilização de um *framework* que atenda bem ao projeto desejado resulta em um desenvolvimento mais ágil, segundo Morisio, Romano e Stanelo (2002) e melhor estruturado, principalmente ao trabalhar em equipes de desenvolvimento. A escolha do melhor *framework* para o projeto, porém,

[1] Professor da UNIBRA. Mestre em Inteligência Computacional, Professor Assistante I do curso Superior em Tecnologia de Redes de Computadores.
E-mail para contato: mr.costaalencar@gmail.com

nem sempre é óbvia, demandando uma análise do mercado e das próprias ferramentas, dada a diversidade de opções e tecnologias possíveis.

Para este trabalho foi escolhido o PHP como linguagem de programação, tendo como justificativa a consolidação da linguagem, que está no mercado desde 1995, quando foi disponibilizada para o público, as constantes atualizações que acompanham as tendências da web, sua popularidade no mercado e presença da comunidade desenvolvedora. Os *frameworks* escolhidos para o teste, CakePHP, CodeIgniter e Laravel, foram escolhidos por ocuparem o top 4 dos frameworks da linguagem PHP no site HotFrameworks e pela facilidade de uso.

A tendência de aplicações com conteúdo dinâmico e a crescente expansão do comércio virtual remetem a necessidade de uma boa comunicação entre a aplicação e o banco de dados para fornecer uma boa experiência ao usuário, tarefa esta que pode ser desempenhada pelos *frameworks*. Por esta razão, foi utilizado o desempenho dos *frameworks* nas requisições ao banco de dados como critério de avaliação, utilizando a métrica de tempo de execução da aplicação teste e espaço ocupado pelo aplicativo.

Este trabalho contribui para todos os profissionais e estudantes na área de desenvolvimento que desejam entender mais sobre *frameworks* e se encontram no processo de escolha para aplicar em um projeto similar a este estudo.

Nas seções seguintes, são abordados conceitos introdutórios importantes, a metodologia do trabalho, bem como o detalhamento do experimento e a discussão dos resultados obtidos.

1.1 Justificativa

Este trabalho tem como objetivo prover conhecimentos acerca dos frameworks mais populares no presente de maneira que estudantes de desenvolvimento e desenvolvedores inexperientes possam escolher com maior facilidade qual framework utilizarão.

A temática para este trabalho foi escolhida a partir da afinidade da autora com a área de desenvolvimento, bem como a identificação da mesma com o público alvo e a popularidade do uso de *frameworks* dentro do contexto de desenvolvimento *web*.

1.2 Objetivos

1.2.1 Objetivo Geral

- Comparar *frameworks* populares de PHP quanto ao desempenho nas requisições ao banco de dados.

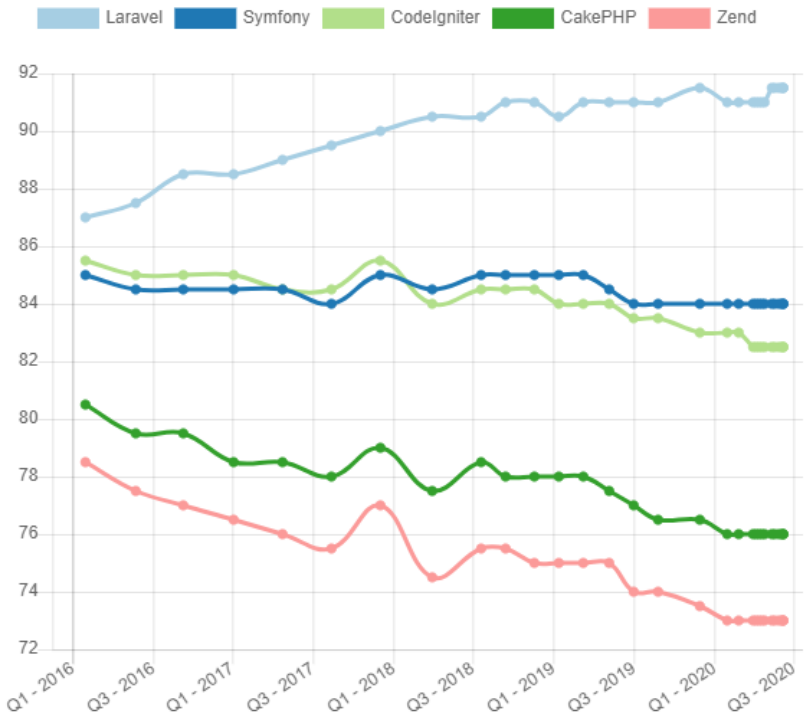
1.2.2 Objetivos Específicos

- Apresentar o conceito de *framework* e os *frameworks* usados para a comparação;
- Apresentar a Linguagem PHP e sua presença no mercado;
- Descrever o experimento e a montagem do mesmo.

2 DELINEAMENTO METODOLÓGICO

Este estudo se baseia numa pesquisa exploratória em conjunto com uma revisão de literatura relacionada ao tema. Os *frameworks* foram decididos com base em sua popularidade de uso (HotFrameworks, 2020) no mercado, de acordo com a facilidade de uso e similaridades em seus objetivos.

Figura 1 - Comparativo de popularidade por período de tempo



Fonte: HotFrameworks (2020)

A pontuação de popularidade de cada *framework* pelo HotFrameworks é definida a partir da média de duas medidas: Pontuação do GitHub na quantidade de estrelas que o repositório do *framework* possui e pontuação do Stack Overflow, baseada no número de perguntas que são marcadas com o nome do *framework*.

O procedimento escolhido para a análise comparativa consiste na criação de uma aplicação back-end que interage com o banco de dados MySQL com dados importados do banco de dados Northwind e faz a criação de pedidos com produtos de forma automática e aleatória por um número determinado de vezes, definidos para este trabalho para uma vez, dez vezes, cem vezes e mil vezes. Após a execução, retorna uma consulta com os pedidos gerados em detalhe.

Este procedimento foi realizado com 3 aplicativos feitos com cada um dos seguintes *frameworks*: CakePHP, CodeIgniter e Laravel. Cada *framework* foi analisado baseado no tempo de processamento do pedido e no espaço ocupado com armazenamento do aplicativo.

3 FUNDAMENTAÇÃO TEÓRICA

O Composer é uma ferramenta para gerenciamentos de dependências PHP que permite que a seleção de *libraries* seja feita manualmente para que, em sequência, instale ou atualize os mesmos localmente, segundo o próprio site. A ferramenta foi amplamente utilizada durante este trabalho para a realização da instalação dos *frameworks* e as dependências necessárias em cada aplicação.

O XAMPP é uma distribuição Apache que contém os componentes MariaDB, PHP e Perl. O propósito do XAMPP é tornar mais fácil ao desenvolvedor a tarefa de levantar um ambiente de desenvolvimento PHP ao prover os recursos que ele necessita. O XAMPP é abrigado no projeto sem fins lucrativos Apache Friends, o qual foi fundado em 2002.

O phpMyAdmin é um front-end web baseado em PHP para o MySQL que provê um painel administrativo com todos os recursos necessários para o gerenciamento do banco de dados podendo acessá-lo remotamente sem a necessidade de estar dentro da máquina servidor. A partir do phpMyAdmin, é possível criar, apagar, importar, exportar bancos de dados inteiros ou tabelas.

O Northwind é o banco de dados de exemplo que acompanha o Microsoft Access, simulando a companhia fictícia Northwind Traders. Este banco de dados dispõe de registros de produtos, clientes, fornecedores, bem como as transações dos

clientes para a empresa quanto da empresa para os fornecedores, provendo um ambiente muito favorável para a execução de *queries* diversas para aprendizado e testes de desenvolvimento (The Access Buddy, 2011). Durante o trabalho, foram utilizadas as tabelas de produtos, detalhes de pedidos e pedidos.

Os códigos das aplicações foram desenvolvidos de forma a apresentar o mesmo conteúdo e com o mínimo de diferenças entre cada um, modificando apenas a sintaxe do código de acordo com o *framework*. Os dados contidos no banco de dados northwind foram restaurados ao original após cada teste antes de prosseguir para o framework seguinte, visando estabilidade nos resultados. Os arquivos e códigos utilizados neste experimento estão disponíveis através do repositório experimento-tcc, disponível através do endereço: https://github.com/ari-mluz/experimento-tcc

A linguagem PHP, *Hypertext Processor*, foi criada em 1994 por Rasmus Lerdorf. Inicialmente, a linguagem era formada apenas por um conjunto de *scripts* com foco na criação de páginas dinâmicas para acesso ao currículo de Rasmus, segundo Dall'Oglio (2009). Ao longo dos anos a linguagem evoluiu, adquiriu novas funções e começou a ocupar seu espaço no mercado ultrapassando 80% de uso na internet em 2016, segundo Prokofyeva & Boltunova (2016).

A atual versão do PHP, o PHP 5 é uma linguagem que permite a Programação Orientada a Objetos (POO), como C++ e Java. Em uma POO, um objeto pode ser compreendido por seu conhecimento, ser manipulado, pensado ou representado, segundo Lima (2012), e uma classe é uma estrutura utilizada para descrever objetos através de atributos, servindo para estruturá-los, segundo Dall'Oglio (2009).

Frameworks são utilizados na linguagem PHP para auxiliar na criação de aplicações na web e são escolhidos por uma série de fatores que podem ou não ter impacto direto no código e no funcionamento do aplicativo. Alguns destes são a documentação, disponibilidade, suporte técnico, suporte a tecnologias necessárias e a eficiência de processamento do *framework* (Zurkiewicz & Milosz, 2015).

CakePHP é um *framework open-source* cujo propósito é o desenvolvimento de aplicações estruturadas na linguagem PHP de forma rápida e flexível, utilizando o padrão MVC. Algumas das vantagens de trabalhar com o *framework*, segundo o próprio site, é a rapidez na construção de protótipos, segurança, nenhuma configuração adicional além da base de dados, licenciamento favorável a instalações comerciais, dentre outros.

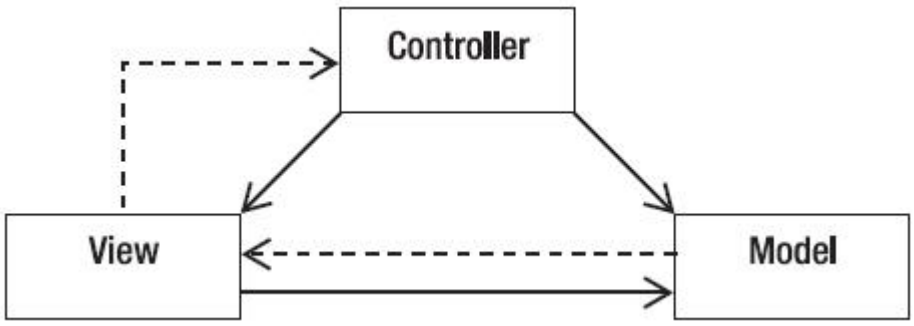
CodeIgniter, segundo o próprio site, é um *framework* de desenvolvimento de aplicações em PHP que permite o padrão MVC e tem como principais propósitos prover um desenvolvimento muito mais rápido quando comparado ao desenvolvimento do zero ao prover *libraries* para as funções mais utilizadas em uma aplicação, enquanto mantendo a lógica e interface simples com o propósito de minimizar a quantidade de código necessário para executar determinada tarefa.

Laravel é um dos frameworks mais populares atualmente, ocupando o primeiro lugar na lista de *frameworks* PHP pelo HotFrameworks. O objetivo do Laravel, assim como os demais *frameworks* citados acima, é o de prover o desenvolvimento de uma aplicação web em PHP mais ágil e flexível utilizando o padrão MVC, porém os diferenciais dos demais *frameworks* deste trabalho são a sua modularização de dependências, mapeamento relacional de objetos e um sistema de autenticação completo (DEV, 2019).

De acordo com Soares (2009), o padrão arquitetural *Model-View-Controller* (MVC) é uma abordagem a qual separa a modelagem, apresentação e fluxo de dados em camadas distintas para modularizar o processo do desenvolvimento e promover a reutilização de código.

A camada *model* do modelo MVC é a responsável pela lógica da aplicação, realizando o processamento, leitura, escrita e validação dos dados que passam por ela, enquanto a camada *view* faz apenas a exibição dos dados ao usuário após estes serem criados ou validados pelo *model* e encaminhados pelo *controller*, responsável por receber e encaminhar as informações da *view* para o *model*, como representado na Figura 2.

Figura 2 - Esquema do fluxo de dados de uma aplicação MVC



Fonte: DevMedia (2013)

4 RESULTADOS

A máquina escolhida para os testes utiliza o sistema operacional Windows 10 e conta com o PHP em sua versão 7.4.7, Composer na versão 1.10.6 para criação e gerenciamento das dependências de cada framework e o XAMPP. As configurações de hardware da máquina consistem em um processador Intel Core i7-8750H @ 2.20GHz 2.21GHz, 16GB de RAM e 2GB VRAM.

O ambiente local utilizado fez uso do programa XAMPP para execução do servidor Apache e banco de dados MySQL na interface phpMyAdmin, a qual hospeda a base de dados northwind utilizada em todas as aplicações. O banco de dados northwind foi obtido através do repositório MyWind (GitHub, 2014?), o qual disponibiliza a base convertida para o formato aceito pelo MySQL.

Os resultados deste experimento foram obtidos ao executar as aplicações 10 vezes cada, medindo o tempo de execução com a função *microtime(true)* do PHP dentro do próprio código, com o objetivo de cobrir variações no tempo de execução de cada aplicação. As tabelas a seguir registram e medem os valores obtidos para que possa ser feita uma comparação e a discussão dos dados nas seções seguintes.

4.1 CakePHP

O tamanho total dos arquivos do repositório da aplicação CakePHP foi de 67.9MB. O tempo médio para uma *query* individual foi de 34,79 em um total de 11.110 *queries* executadas.

A estrutura utilizada para fazer *queries* no banco de dados pode ser encontrada na Figura 3, onde primeiramente uma variável é criada para conter uma tabela e posteriormente outras variáveis a referenciam para fazer buscas àquela específica tabela no banco de dados.

Figura 3 - Recorte da Estrutura de *Query* em CakePHP

```
$customers = TableRegistry::getTableLocator()->get('Customers');

$query_customers = $customers->find('all',array(
    array('conditions'=>array('id'=>$customer_id)),
));
```

Fonte: A autora (2020)

A estrutura utilizada para fazer *Insert* no banco de dados pode ser encontrada na Figura 4, onde uma variável referência a variável onde o banco de dados se encontra. Posteriormente, esta mesma variável gera os campos a serem inseridos antes de preenchê-los e salvá-los no banco de dados.

Figura 4 - Recorte da Estrutura de *Insert* em CakePHP

```
$orders_insert = $orders->query();

$orders_insert->insert(['employee_id', 'order_date', 'shipped_date', 'shipper_id',
'customer_id', 'ship_name', 'ship_address', 'ship_city', 'ship_state_province',
'ship_zip_postal_code', 'ship_country_region', 'shipping_fee', 'taxes', 'payment_type',
'paid_date', 'notes', 'tax_rate', 'tax_status_id', 'status_id'])
    ->values([
        'id' => $id,
        'employee_id' => $employee_id,
        'order_date'  => $order_date,
        'shipped_date'  => $shipped_date,
        'shipper_id'  => $shipper_id,
        'customer_id' => $customer_id,
        'ship_name'  => $ship_name,
        'ship_address'  => $ship_address,
        'ship_city'  => $ship_city,
        'ship_state_province'  => $ship_state_province,
        'ship_zip_postal_code'  => $ship_zip_postal_code,
        'ship_country_region'  => $ship_country_region,
        'shipping_fee'  => $shipping_fee,
        'taxes'  => $taxes,
        'payment_type'  => $payment_type,
        'paid_date'  => $paid_date,
        'notes'  => $notes,
        'tax_rate'  => $tax_rate,
        'tax_status_id'  => $tax_status_id,
        'status_id'  => $status_id
    ])
    ->execute();
```

Fonte: A autora (2020)

Na execução de uma *query* individual, o tempo mínimo de execução observado foi de 52,1 ms e o tempo máximo foi de 87,89 ms. Os tempos individuais gravados na rodada estão visíveis na Tabela 1. A tempo médio de execução da *query* individual foi de 64,13 ms, com uma variância de 146,21 ms^2 e um desvio padrão de 11,47 ms, como pode ser observado na Tabela 2.

Tabela 1 - Relatório de execuções e tempos (CakePHP)

	1 Query		10 Queries		100 Queries		1000 Queries	
	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)
Execuções	53,41	53,41	371,20	37,12	3.733,18	37,33	34.699,00	34,70
	74,02	74,02	381,98	38,20	3.373,08	33,73	34.261,75	34,26
	56,81	56,81	311,05	31,11	3.227,17	32,27	35.774,17	35,77
	87,89	87,89	301,02	30,10	2.025,41	20,25	33.835,85	33,84
	70,27	70,27	249,03	24,90	2.103,58	21,04	33.937,75	33,94
	66,49	66,49	355,23	35,52	3.090,63	30,91	36.627,00	36,63
	54,37	54,37	445,63	44,56	3.039,25	30,39	38.273,90	38,27
	53,01	53,01	376,99	37,70	4.683,07	46,83	35.233,81	35,23
	72,97	72,97	505,71	50,57	4.479,86	44,80	34.190,06	34,19
	52,10	52,10	346,49	34,65	3.140,20	31,40	32.551,99	32,55

Fonte: A autora (2020)

Tabela 2 - Valores estatísticos das operações em ms (CakePHP)

1 Query		10 Queries		100 Queries		1000 Queries	
Média	64,13	Média	364,43	Média	3289,54	Média	34938,53
Variância	146,21	Variância	5295,44	Variância	743.945,72	Variância	2.638.862,06
Desvio Padrão	11,47	Desvio Padrão	69,04	Desvio Padrão	818,26	Desvio Padrão	1541,10

Fonte: A autora (2020)

Na execução de dez *queries* conjuntas, o tempo mínimo de execução foi de 301,02 ms e o tempo máximo de execução foi de 505,71 ms, com tempos mínimo e máximo de execução de uma *query* individual de 30,1 ms e 50,57 ms, respectivamente, segundo a Tabela 1. O tempo médio de execução das *queries* foi de 364,43 ms com variância de 5295,44 ms² e um desvio padrão de 69,04 ms segundo a Tabela 2. O tempo médio de execução de uma *query* individual é de 36,44 ms, com uma variância de 52,95 ms² e desvio padrão de 6,9 ms, segundo a Tabela 3.

Tabela 3 - Valores estatísticos individuais da *query* em ms (CakePHP)

1 Query		10 Queries		100 Queries		1000 Queries	
Média	64,13	Média	36,44	Média	32,90	Média	34,94
Variância	146,21	Variância	52,95	Variância	74,39	Variância	2,64
Desvio Padrão	11,47	Desvio Padrão	6,90	Desvio Padrão	8,18	Desvio Padrão	1,54

Fonte: A autora (2020)

Na execução de cem *queries* conjuntas, o tempo mínimo de execução foi de 2.025,41 ms e o tempo máximo de execução foi de 4.683,07 ms, com tempos mínimo e máximo de execução de uma *query* individual de 20,25 ms e 46,83 ms,

respectivamente, segundo a Tabela 1. O tempo médio de execução das *queries* foi de 3.289,54 ms com variância de 743.945,72 ms² e um desvio padrão de 818,26 ms, segundo a Tabela 2. O tempo médio de execução de uma *query* individual é de 32,9 ms, com uma variância de 74,39 ms² e desvio padrão de 8,18 ms, segundo a Tabela 3.

Na execução de mil *queries* conjuntas, o tempo mínimo de execução foi de 32.551,99 ms e o tempo máximo de execução foi de 38.273,9 ms, com tempos mínimo e máximo de execução de uma *query* individual de 32,55 ms e 38,27 ms, respectivamente, segundo a Tabela 1. O tempo médio de execução das *queries* foi de 34.938,53 ms com variância de 2.638.862,06 ms² e um desvio padrão de 1.541,1 ms, segundo a Tabela 2. O tempo médio de execução de uma *query* individual é de 34,94 ms, com uma variância de 2,64 ms² e desvio padrão de 1,54 ms, segundo a Tabela 3.

4.2 CodeIgniter

O tamanho total dos arquivos do repositório da aplicação CodeIgniter foi de 11,3MB. O tempo médio para uma *query* individual foi de 18,54 em um total de 11.110 *queries* executadas.

A estrutura utilizada para fazer *queries* no banco de dados pode ser encontrada na Figura 5, onde primeiramente uma variável é criada fazendo uma busca em em SQL ao servidor SQL antes de converter este resultado em um *Array* para que ele possa posteriormente ser exibido.

Figura 5 - Recorte da Estrutura de *Query* em CodeIgniter

```
$query = $db->query(
    'SELECT last_name, first_name, address, city, state_province, zip_postal_code, country_region
    FROM customers WHERE id = '.$customer_id.'');
$result = $query->getResultArray();
```

Fonte: A autora (2020)

A estrutura utilizada para fazer *Insert* no banco de dados pode ser encontrada na Figura 6, onde um *Array* é criado com campos com nome de campos iguais aos nomes dos campos do banco de dados. Antes de usar a variável do servidor para diretamente inserir os dados na tabela apropriada.

Figura 6 - Recorte da Estrutura de *Insert* em CodeIgniter

```
$data = [
    'id' => $id,
    'employee_id' => $employee_id,
    'order_date'  => $order_date,
    'shipped_date'  => $shipped_date,
    'shipper_id'  => $shipper_id,
    'customer_id' => $customer_id,
    'ship_name'  => $ship_name,
    'ship_address'  => $ship_address,
    'ship_city'  => $ship_city,
    'ship_state_province'  => $ship_state_province,
    'ship_zip_postal_code'  => $ship_zip_postal_code,
    'ship_country_region'  => $ship_country_region,
    'shipping_fee'  => $shipping_fee,
    'taxes'  => $taxes,
    'payment_type'  => $payment_type,
    'paid_date'  => $paid_date,
    'notes'  => $notes,
    'tax_rate'  => $tax_rate,
    'tax_status_id'  => $tax_status_id,
    'status_id'  => $status_id
];
$db->table('orders')->insert($data);
```

Fonte: A autora (2020)

Na execução de uma *query* individual, o tempo mínimo de execução observado foi de 28,99 ms e o tempo máximo foi de 43,61 ms. Os tempos individuais gravados na rodada estão visíveis na Tabela 4. A tempo médio de execução da *query* individual foi de 35,44 ms, com uma variância de 29,71 ms² e um desvio padrão de 5,17 ms, como pode ser observado na Tabela 5.

Tabela 4 - Relatório de execuções e tempos (CodeIgniter)

	1 Query		10 Queries		100 Queries		1000 Queries	
Execuções	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)
	34,77	34,77	173,09	17,31	1.344,07	13,44	16.567,47	16,57
	27,25	27,25	139,64	13,96	1.592,54	15,93	16.372,64	16,37
	28,99	28,99	156,32	15,63	1.369,55	13,70	17.714,95	17,71
	43,61	43,61	159,91	15,99	1.433,39	14,33	18.462,33	18,46
	37,97	37,97	139,72	13,97	1.483,14	14,83	18.610,20	18,61
	41,34	41,34	174,58	17,46	1.620,96	16,21	19.579,28	19,58
	30,89	30,89	151,15	15,12	1.479,66	14,80	19.864,42	19,86
	36,29	36,29	176,78	17,68	1.808,89	18,09	20.041,22	20,04
	40,26	40,26	138,76	13,88	1.564,54	15,65	21.005,59	21,01
	33,06	33,06	178,45	17,85	1.530,17	15,30	20.613,68	20,61

Fonte: A autora (2020)

Tabela 5 - Valores estatísticos das operações em ms (CodeIgniter)

1 Query		10 Queries		100 Queries		1000 Queries	
Média	35,44	Média	158,84	Média	1522,69	Média	18883,18
Variância	29,71	Variância	262,36	Variância	18.333,45	Variância	2.618.824,03
Desvio Padrão	5,17	Desvio Padrão	15,37	Desvio Padrão	128,45	Desvio Padrão	1535,23

Fonte: A autora (2020)

Na execução de dez *queries* conjuntas, o tempo mínimo de execução foi de 138,76 ms e o tempo máximo de execução foi de 178,45 ms, com tempos mínimo e máximo de execução de uma query individual de 13,87 ms e 17,84 ms, respectivamente, segundo a Tabela 4. O tempo médio de execução das *queries* foi de 158,84 ms com variância de 262,36 ms² e um desvio padrão de 15,37 ms, segundo a Tabela 5. O tempo médio de execução de uma *query* individual é de 35,44 ms, com uma variância de 29,71 ms² e desvio padrão de 5,17 ms, segundo a Tabela 6.

Tabela 6 - Valores estatísticos individuais da *query* em ms (CodeIgniter)

1 Query		10 Queries		100 Queries		1000 Queries	
Média	35,44	Média	15,88	Média	15,23	Média	18,88
Variância	29,71	Variância	2,62	Variância	1,83	Variância	2,62
Desvio Padrão	5,17	Desvio Padrão	1,54	Desvio Padrão	1,28	Desvio Padrão	1,54

Fonte: A autora (2020)

Na execução de cem *queries* conjuntas, o tempo mínimo de execução foi de 1.344,07 ms e o tempo máximo de execução foi de 1.808,89 ms, com tempos mínimo e máximo de execução de uma *query* individual de 13,44 ms e 18,08 ms,

respectivamente, segundo a Tabela 4. O tempo médio de execução das *queries* foi de 1.522,69 ms com variância de 18.333,45 ms² e um desvio padrão de 128,45 ms, segundo a Tabela 5. O tempo médio de execução de uma *query* individual é de 15,23 ms, com uma variância de 1,83 ms² e desvio padrão de 1,28 ms, segundo a Tabela 6.

Na execução de mil *queries* conjuntas, o tempo mínimo de execução foi de 16.372,64 ms e o tempo máximo de execução foi de 21.005,59 ms, com tempos mínimo e máximo de execução de uma query individual de 16,37 ms e 21 ms, respectivamente, segundo a Tabela 4. O tempo médio de execução das queries foi de 18.883,18 ms com variância de 2.618.824,03 ms² e um desvio padrão de 1.535,23, ms segundo a Tabela 5. O tempo médio de execução de uma query individual é de 18,88 ms, com uma variância de 2,62 ms² e desvio padrão de 1,54 ms, segundo a Tabela 6.

4.3 Laravel

O tamanho total dos arquivos do repositório da aplicação Laravel foi de 48,9MB. O tempo médio para uma *query* individual foi de 22,89 em um total de 11.110 *queries* executadas.

A estrutura utilizada para fazer *queries* no banco de dados pode ser encontrada na Figura 7, onde uma variável é criada é uma função é utilizada para acessar uma tabela do banco de dados e selecionar linhas que sigam as especificações necessárias.

Figura 7 - Recorte da Estrutura de *Query* em Laravel

```
$query_customerbuild = DB::table('customers')
    ->where('id', '=', ''.$customer_id.'')
    ->get();
```

Fonte: A autora (2020)

A estrutura utilizada para fazer *Insert* no banco de dados pode ser encontrada na Figura 8, uma função seleciona uma tabela do banco de dados onde um *Array* contendo campos com nome idêntico ao dos campos do banco de dados, e preencher tais campos para que eles sejam inseridos no banco de dados.

Figura 8 - Recorte da Estrutura de *Insert* em Laravel

```
DB::table('orders')->insert(
    [
        'id' => $id,
        'employee_id' => $employee_id,
        'order_date'  => $order_date,
        'shipped_date'  => $shipped_date,
        'shipper_id'  => $shipper_id,
        'customer_id' => $customer_id,
        'ship_name'  => $ship_name,
        'ship_address'  => $ship_address,
        'ship_city'  => $ship_city,
        'ship_state_province'  => $ship_state_province,
        'ship_zip_postal_code'  => $ship_zip_postal_code,
        'ship_country_region'  => $ship_country_region,
        'shipping_fee'  => $shipping_fee,
        'taxes'  => $taxes,
        'payment_type'  => $payment_type,
        'paid_date'  => $paid_date,
        'notes'  => $notes,
        'tax_rate'  => $tax_rate,
        'tax_status_id'  => $tax_status_id,
        'status_id'  => $status_id
    ]
);
```

Fonte: A autora (2020)

Na execução de uma *query* individual, o tempo mínimo de execução observado foi de 21,27 ms e o tempo máximo foi de 47,34 ms. Os tempos individuais gravados na rodada estão visíveis na Tabela 7. A tempo médio de execução da *query* individual foi de 32,92 ms, com uma variância de 67,83 ms² e um desvio padrão de 7,81 ms, como pode ser observado na Tabela 8.

Tabela 7 - Relatório de execuções e tempos (Laravel)

	1 Query		10 Queries		100 Queries		1000 Queries	
	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)	Tempo total (ms)	Tempo por query (ms)
Execuções	47,34	47,34	157,45	15,75	2.124,98	21,25	22.916,66	22,92
	32,29	32,29	221,25	22,13	1.574,90	15,75	20.894,95	20,89
	30,52	30,52	237,83	23,78	1.965,90	19,66	20.046,55	20,05
	30,77	30,77	183,80	18,38	2.080,63	20,81	22.015,63	22,02
	34,70	34,70	216,20	21,62	1.853,13	18,53	19.868,26	19,87
	42,26	42,26	197,67	19,77	2.012,78	20,13	24.150,08	24,15
	38,36	38,36	168,13	16,81	1.822,01	18,22	28.099,06	28,10
	30,07	30,07	230,73	23,07	2.118,85	21,19	26.544,54	26,54
	21,61	21,61	189,69	18,97	2.110,80	21,11	24.776,27	24,78
	21,27	21,27	202,64	20,26	1.807,79	18,08	23.138,16	23,14

Fonte: A autora (2020)

Tabela 8 - Valores estatísticos das operações em ms (Laravel)

1 Query		10 Queries		100 Queries		1000 Queries	
Média	32,92	Média	200,54	Média	1947,18	Média	23245,02
Variância	67,83	Variância	699,32	Variância	32.539,17	Variância	7.377.620,36
Desvio Padrão	7,81	Desvio Padrão	25,09	Desvio Padrão	171,13	Desvio Padrão	2576,79

Fonte: A autora (2020)

Na execução de dez *queries* conjuntas, o tempo mínimo de execução foi de 157,45 ms e o tempo máximo de execução foi de 237,83 ms, com tempos mínimo e máximo de execução de uma *query* individual de 15,75 ms e 23,78 ms, respectivamente, segundo a Tabela 7. O tempo médio de execução das *queries* foi de 200,54 ms com variância de 699,32 ms² e um desvio padrão de 25,09 ms, segundo a Tabela 8. O tempo médio de execução de uma *query* individual é de 20,05 ms, com uma variância de 6,99 ms² e desvio padrão de 2,51 ms, segundo a Tabela 9.

Tabela 9 - Valores estatísticos individuais da *query* em ms (Laravel)

1 Query		10 Queries		100 Queries		1000 Queries	
Média	32,92	Média	20,05	Média	19,47	Média	23,25
Variância	67,83	Variância	6,99	Variância	3,25	Variância	7,38
Desvio Padrão	7,81	Desvio Padrão	2,51	Desvio Padrão	1,71	Desvio Padrão	2,58

Fonte: A autora (2020)

Na execução de cem *queries* conjuntas, o tempo mínimo de execução foi de 1.574,9 ms e o tempo máximo de execução foi de 2.124,98 ms, com tempos mínimo e máximo de execução de uma *query* individual de 15,75 ms e 21,25 ms,

respectivamente, segundo a Tabela 7. O tempo médio de execução das *queries* foi de 1.947,18 ms com variância de 32.539,17 ms² e um desvio padrão de 171,13 ms, segundo a Tabela 8. O tempo médio de execução de uma *query* individual é de 19,47 ms, com uma variância de 3,25 ms² e desvio padrão de 1,71 ms, segundo a Tabela 9.

Na execução de mil *queries* conjuntas, o tempo mínimo de execução foi de 19.868,26 ms e o tempo máximo de execução foi de 28.099,06 ms, com tempos mínimo e máximo de execução de uma *query* individual de 19,87 ms e 28,1 ms, respectivamente, segundo a Tabela 7. O tempo médio de execução das *queries* foi de 23.245,02 ms com variância de 7.377.620,36 ms² e um desvio padrão de 2.576,79 ms, segundo a Tabela 8. O tempo médio de execução de uma *query* individual é de 23,25 ms, com uma variância de 7,38 ms² e desvio padrão de 2,58 ms, segundo a Tabela 9.

5 DISCUSSÃO

A partir da análise dos resultados das amostras de cada framework para a execução de mil queries, é possível definir o melhor framework nos quesitos estabilidade e velocidade de processamento ao utilizar, respectivamente, a média e o desvio padrão. No quesito tamanho, serão utilizados os tamanhos ocupados em- disco por cada aplicação.

CodeIgniter demonstrou ser o framework com maior estabilidade, possuindo menor desvio padrão em todas as escalas testadas, ficando apenas bem próximo na escala de mil queries do CakePHP, onde o desvio padrão da velocidade de processamento do CodeIgniter foi 1.535,23 ms, enquanto do CakePHP foi 1.541,1 ms.

Com relação a velocidade de processamento, antes de qualquer outro dado relacionado, é necessário afirmar que, em todos os frameworks, uma query e inserção individuais gastaram, em média, um tempo consideravelmente maior que o tempo médio de execução de uma query e inserção individual quando múltiplas queries eram feitas em seguida, chegando a demorar 91% de tempo adicional para executar as mesmas ações.

A média geral de velocidade de processamento do CodeIgniter foi cerca de 20% melhor que o segundo colocado, Laravel, com um tempo de execução de cerca de 18,54 ms, porém com relação queries individuais, Laravel demonstrou ser 7% mais rápido que o CodeIgniter. Em todas as outras escalas, o CodeIgniter demonstrou ser mais rápido que o Laravel e o CakePHP por no mínimo 18%.

O CakePHP teve períodos de execução consideravelmente mais longos que os outros frameworks analisados, isto se deve ao fato de que ele possui uma etapa adicional para interação com o banco de dados seja para leitura ou escrita.

O framework mais leve foi o CodeIgniter, sendo este um ponto forte declarado pelos próprios desenvolvedores do framework em seu site oficial. Possuindo apenas 11,3 *megabytes,* o CodeIgniter é mais que 4 vezes menor que o Laravel e em torno de 6 vezes menor que o CakePHP. Isto se dá conta pelo fato dele possuir um kit mais simples que os outros frameworks apresentam.

Laravel, mesmo sendo um framework modular, não recebeu adição de nenhum dos seus diversos módulos adicionais durante o experimento, e, mesmo assim, ainda ocupou um pouco mais que 4 vezes o espaço do CodeIgniter.

6 CONSIDERAÇÕES FINAIS

Considerando os resultados do experimento, por conta de sua velocidade e simplicidade, o CodeIgniter é o mais apropriado para projetos de pequeno porte os quais não venham a necessitar das diversas ferramentas incluídas no CakePHP ou nos módulos do Laravel.

É preciso lembrar que o escopo do estudo foi estritamente limitado ao uso destes frameworks em um ambiente simples, estudos adicionais são necessários tanto para testar um uso mais diversos das funcionalidades dos frameworks quanto como aplicações criadas neles funcionam com o aumento da complexidade dos programas, o que não faz parte do escopo deste trabalho.

Em trabalhos futuros, é importante que sejam abordados outros frameworks como Symfony e Zend, que no presente fazem parte do top 5 do site HotFrameworks, bem como outros frameworks que terão popularidade no futuro. O experimento também deve ser diversificado para levar em conta a complexidade do framework, a dificuldade da manutenção do código e as funcionalidades de front-end, já que estes também são fatores importantes na escolha de um framework por um estudante de desenvolvimento ou desenvolvedor inexperiente.

REFERÊNCIAS

Prokofyeva, Natalya & Boltunova, Victoria. (2017). **Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems.** Procedia Computer Science. 104.51-56.10.1016/j.procs.2017.01.059.

PHPMYADMIN. [S.I.][2017?]. **Bringing MySQL to the web.** Disponível em: https://www.phpmyadmin.net/about/ Acesso em: Junho, 2020.

CAKEPHP. [S.I.][2020?]. **CakePHP.** Disponível em: https://cakephp.org Acesso em: Junho, 2020.

CAKEPHP. [S.I.][2020?]. **CakePHP num piscar de olhos.** Disponível em: https://book.cakephp.org/4/pt/intro.html Acesso em: Junho, 2020.

SOARES, W. (2009) **Crie um Framework para Sistemas Web com PHP 5 e AJAX**. São Paulo: Editora Érica.

GONÇALVES, D. Cronapp. mai/2020. **Desenvolvimento de software: Tendências do momento e do futuro.** Disponível em: https://blog.cronapp.io/desenvolvimento-de-software-as-tendencias-do-momento-e-do-futuro/ Acesso em: Junho, 2020.

LUZ, A. GitHub. 2020. **experimento-tcc.** Disponível em: https://github.com/ari-mluz/experimento-tcc Acesso em: Junho, 2020.

COMPOSER. [S.I.][2018?]. **Introduction - Composer.** Disponível em: https://getcomposer.org/doc/00-intro.md Acesso em: Junho, 2020.

MEDEIROS, H. 2013. **Introdução ao Padrão MVC**. Disponível em: https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308. Acesso em: Maio, 2020.

PHP. **Manual do PHP**. Disponível em: http://www.php.net/manual/pt_BR/. Acesso em: Maio, 2020.

GITHUB. [S.I.][2014?] **MyWind**. Disponível em: https://github.com/dalers/mywind Acesso em: Junho, 2020.

THEACCESSBUDDY. [S.I.] jul/2011 **Northwind Database Explained**. Disponível em: https://theaccessbuddy.wordpress.com/2011/07/03/northwind-database-explained Acesso em: Junho, 2020.

DALL'OGLIO, P. (2009) **PHP: Programando com orientação a objetos**. 2. ed. São Paulo, Editora Novatec.

M. Morisio, D. Romano and I. Stamelos, **Quality, productivity, and learning in framework-based development:** an exploratory case study, in IEEE Transactions on Software Engineering, vol. 28, no. 9, pp. 876-888, Sept. 2002.

Zurkiewicz, Adrian & Miłosz, Marek. (2015). **SELECTING A PHP FRAMEWORK FOR A WEB APPLICATION PROJECT** — THE METHOD AND CASE STUDY. 10.13140/RG.2.1.1321.2567.

APACHE FRIENDS. [S.I.] [2018?] **Sobre.** Disponível em: https://www.apachefriends.org/pt_br/about.html Acesso em: Junho, 2020.

LIMA, A. S. (2012) **UML 2.3: Do requisito à Solução**. São Paulo: Editora Érica.

HOTFRAMEWORKS. **Web framework rankings** Disponível em: https://hotframeworks.com/languages/php Acesso em: Abril, 2020.

CODEIGNITER. mai/2020. **Welcome to CodeIgniter4** Disponível em: https://codeigniter.com/user_guide/intro/index.html Acesso em: Junho, 2020.

DEV. [S.I.] dez/2019 **What is Laravel? Explain it like I'm five.** Disponível em: https://dev.to/creativetim_official/what-is-laravel-explain-it-like-i-m-five-19eb Acesso em: Junho, 2020.

.