UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ÍTALO DE OLIVEIRA COSTA

UMA COMPARAÇÃO ENTRE MICRO FRAMEWORKS WEB PARA O DESENVOLVIMENTO DE APLICAÇÕES BACK-END EM JAVA

QUIXADÁ

2019

ÍTALO DE OLIVEIRA COSTA

UMA COMPARAÇÃO ENTRE MICRO FRAMEWORKS WEB PARA O DESEVOLVIMENTO DE APLICAÇÕES BACK-END EM JAVA

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Orientador: Prof. Júlio Serafim Martins

Coorientador: Prof. Me. Carlos Diego Andrade de Almeida

QUIXADÁ

2019

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

C876c Costa, Ítalo de Oliveira.
Uma comparação entre micro frameworks web para o desenvolvimento de aplicações back-end em Java / Ítalo de Oliveira Costa. – 2019.
66 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2019.
Orientação: Prof. Júlio Serafim Martins.
Coorientação: Prof. Me. Carlos Diego Andrade de Almeida.

1. Comparação binária. 2. Microframework. 3. Java (Linguagem de programação de computador). I. Título.

CDD 005

ÍTALO DE OLIVEIRA COSTA

UMA COMPARAÇÃO ENTRE MICRO FRAMEWORKS WEB PARA O DESENVOLVIMENTO DE APLICAÇÕES BACK-END EM JAVA

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Júlio Serafim Martins (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Me. Carlos Diego Andrade de Almeida (Coorientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Tércio Jorge da Silva
Universidade Federal do Ceará (UFC)

Ao Bom Deus pelo dom da fortaleza.
A minha família e amigos pelo apoio.
A todos aqueles que de uma forma ou de outra se beneficiarão com esse trabalho.

AGRADECIMENTOS

Primeiramente agradeço ao Bom Deus pela força, paciência, perseverança e sabedoria que, de graça me foi concedida. Obrigado pelo sustento e pelas provações superadas, tudo isso só me faz te amar mais. *I live to serve you, God*.

Agradeço também a minha família. A meu pai Hiram, que fez o possível e impossível para que eu conseguisse estudar em Quixadá. Pai, mesmo com todas as imperfeições, sua determinação resoluta foi, é e sempre será minha inspiração.

A minha mãe Regina por todas as batalhas que travou por mim, por todo o sacrifício que fez por mim, por todas as palavras de repreensão e motivação ditas nos momentos mais oportunos. Mãe, a senhora sempre acerta.

A minha irmãzinha Iara, que sendo a mais nova me fez querer crescer e ser seu exemplo me motivou a ser uma pessoa melhor a cada dia. Eu me orgulho muito do você têm se tornado. Agradeço ainda a todos os meus parentes que, quando precisei e sem pedir nada em troca, apenas ajudaram de bom grado na estadia em Quixadá.

Agradeço ao meu amigo conterrâneo, Rafael por me levar para Quixadá e por dar todo o suporte que eu precisava, já que eu me encontrava em "uma terra estrangeira". Agradeço aos meus amigos que residem comigo, Lincoln, Natanael, François e Hilderjares. Que nossa amizade perdure por toda a vida. Agradeço aos meus amigos da UFC pelos bons momentos. Um agradecimento especial ao Rodrigo, ex-morador, que foi e ainda o é, para mim um irmão mais velho. Sucesso na sua vida meu amigo.

Agradeço aos meus orientadores Prof. Júlio Serafim e Prof. Carlos Diego, por todas as reuniões produtivas e as improdutivas também. Por todos os incentivos, conselhos e "carões". Somente com a orientação de vocês esse trabalho foi possível.

Agradeço a UFC, a todos os servidores e professores pelo esforço e dedicação em fazer desse campus uma referência em tecnologia no sertão central.

“... And then you came to me, with nothing but anger and determination!”

(Illidan Stormrage.)

RESUMO

Um micro *framework web back-end* é um *framework* de aplicação com apenas os componentes essenciais para a criação de uma aplicação *web*. Se concentrando em fornecer uma funcionalidade de uma determinada área de forma muito eficiente com menos componentes e menor *overhead* em relação a um *framework* convencional. Assim, este trabalho tem como objetivo apresentar um comparativo entre os micro *frameworks web back-end* em Java, mais populares em número de estrelas nos seus repositórios na plataforma do Github, são eles: Spark, Jooby e Pippo. Neste trabalho foi usado o critério de Popularidade mensurada pelo número de estrelas no repositório do Github. Nessa comparação foram utilizados os seguintes critérios de comparação: documentação, curva de aprendizado, Popularidade, Suporte a IOC, Suporte a arquitetura REST, Validação da entrada, Artigos publicados e Tamanho da comunidade. Para fazer a avaliação foi implementada a aplicação do PetClinic que foi utilizada em diversos outros trabalhos na literatura. A partir dos resultados obtidos da comparação dentre os micro *frameworks web back-end* selecionados foi constatado que o Spark Framework foi o melhor para a implementação da aplicação de teste selecionada, o PetClinic.

Palavras-chave: Comparação binária. Microframework. Java (Linguagem de programação de computador).

ABSTRACT

A backend micro web framework is an application framework with only the essential components for creating a web application. Focusing on providing functionality of a given area very efficiently with fewer components and lower overhead compared to a conventional framework. Thus, this paper aims to present a comparison between the Java backend web micro frameworks, most popular in number of stars in their repositories on the Github platform, they are: Spark, Jooby and Pippo. In this work we used the criterion of Popularity measured by the number of stars in the Github repository. In this comparison we used the following comparison criteria: Documentation, Learning Curve, Popularity, IOC Support, REST Architecture Support, Input Validation, Published Articles, and Community Size. To make the evaluation was implemented the application of PetClinic that was used in several other works in the literature. From the results obtained from the comparison between the selected back-end web micro frameworks it was found that the Spark Framework (which is not Apache Spark) was the best for the implementation of the selected test application, PetClinic.

Keywords: Binary comparison. Microframework Java (Computer Programming Language).

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IOC	Inversão de Controle
REST	*Representational State Transfer*
MVC	Mode-View-Controller
API	*Application Programming Interface*
CRUD	*Create, Read, Update and Apague.*
DI	*Dependency Injection*
HTTP	*Hypertext Transfer Protocol*

SUMÁRIO

1 INTRODUÇÃO

O mercado de *software* tem crescido e se tornado cada vez mais competitivo e exigente. Por consequência dessa acirrada competição entre as empresas, os projetos de *software* necessitam ser desenvolvidos com o máximo de rapidez, qualidade e com o melhor custo benefício possível (ABES, 2018).

Com o crescimento da popularidade da *web* e suas transformações nos anos 2000, esta se tornou, dentre muitas coisas, uma plataforma para desenvolvimento de aplicações (O'REILLY, 2005). Por esses dois fatores a importância de se desenvolver para essa nova plataforma atrelada às novas possibilidades ainda não exploradas, são notáveis. Com isso, as exigências de mercado supracitadas são percebidas também nessa área, o desenvolvimento de *softwares* para a *web*, na parte de *back-end*.

As aplicações *back-end*, ou "do lado do servidor", são aquelas que contêm as regras de negócio do domínio (MDN, 2005). Quando utilizado no seu desenvolvimento uma arquitetura baseada em componentes e uma linguagem de programação orientada a objetos, há uma preocupação especial com reuso desses componentes de software que, por sua vez, devem ser abstrações de alta granularidade a fim de garantir esse reuso. Componentes de persistência e de comunicação HTTP (*Hypertext Transfer Protocol*) são exemplos desse reuso (SOMMERVILLE, 2011).

Preocupando-se com o reuso desses componentes e a manutenibilidade das aplicações *web*, foram criadas estruturas que integram o que supostamente é necessário para o desenvolvimento de uma aplicação desse tipo, os chamados *frameworks* de aplicação (SOMMERVILLE, 2011).

A criação e uso desses *frameworks* têm crescido cada vez mais. Esse crescimento se dá ao fato de que a adoção é justificável não somente em aplicações de grande porte, mas bem como nas de pequeno porte (PETRIJEVCANIN e SUDAREVIC, 2011), uma vez que o uso dessas ferramentas trazem benefícios já apresentados, como por exemplo, a maximização do reuso e a aplicação e suporte ao uso de padrões de projetos. (SOMMERVILLE, 2011)

Devido a esse fato, existe uma grande variedade dessas ferramentas (FRANCO, 2011), que por isso podem causar grandes atrasos na hora da escolha de uma delas para um determinado projeto. Agregado a isso temos também o conceito de *frameworks* ditos como

"*full-stack"*, que possuem muitos componentes, muitos dos quais, em aplicações menores, agregam uma complexidade desnecessária, aumentam ainda mais a complexidade dessa decisão e o atraso na escolha (SOMMERVILLE, 2011).

Com relação a essa possibilidade de agregação de complexidade desnecessária, para atender a demanda de ferramentas com características que evitam tal problema, foram criados os chamados micro *frameworks*, que segundo Josh Lockhart, criador do Slim Framework (NEW MEDIA CAMPAIGNS, 2014) "Um micro framework é uma coleção de necessidades básicas para construir uma aplicação web. Normalmente composto por módulos de requisição e resposta HTTP e para o roteamento". O Slim, escrito em PHP, foi precursor do movimento micro.

Micro *frameworks* podem ser empregados no desenvolvimento de diversos tipos de aplicações, de variados tamanhos. No entanto, eles vêm ganhando maior notoriedade com a ascensão da arquitetura de microsserviços.

A arquitetura de microsserviços é um tipo de arquitetura em que a aplicação é decomposta em funções básicas dispostas em forma de um suite de serviços, cada um com o seu *deploy* independente, possuem o mínimo de gerenciamento centralizado e podem ser feitos em variadas linguagens (THOUGHTWORKS, 2015). O uso desses micro *frameworks* tem se mostrado bastante satisfatório nesse tipo de arquitetura, pricipalmente em aplicações Java (ROSA, 2016).

Existem também micro *frameworks* em outras linguagens, a exemplo do Ruby com o Sinatra (SINATRA, 2007), do Python com o Flask (FLASK, 2010) e do Java como anteriormente citado, com o Spring Boot (PIVOTAL, 2017), Spark (SPARKJAVA, 2011) (que não é o Apache Spark) e o Jooby (JOOBY, 2014).

Existem vários trabalhos acadêmicos sobre comparação de *frameworks web back-end*. Alguns exemplos são: (BAGESTAN, 2016), (FRANCO, 2011) e (COUTO e FOSCHINI, 2015). No entanto, pouco se encontram trabalhos acadêmicos de comparação de micro *frameworks web back-end*.

Este trabalho tem como objetivo principal apresentar um comparativo entre os micro *frameworks web back-end* em Java mais populares na plataforma do Github. Utilizando nessa comparação alguns critérios comuns a essas ferramentas sendo elas, documentação, curva de

aprendizado, Popularidade, Suporte a IOC, Suporte a arquitetura REST, Validação da entrada, Artigos publicados e Tamanho da comunidade, uma vez que, algumas das razões para o uso desse tipo de ferramenta são: a redução da complexidade e o aumento na produtividade (FRANCO, 2011).

Como contribuições principais deste trabalho, podem-se apontar as seguintes: (i) servir de apoio à decisão dos desenvolvedores na escolha dessas ferramentas e (ii) contribuir com o preenchimento da lacuna de trabalhos acadêmicos relacionados a esse domínio, auxiliando a todo e qualquer estudante ou pesquisadores interessados nessa área.

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 está descritos os trabalhos relacionados com o presente trabalho. O Capítulo 3 contém a fundamentação teórica. O Capítulo 4 retrata os procedimentos metodológicos junto com o cronograma de execução. Por fim, O Capítulo 5 mostra os resultados.

2 TRABALHOS RELACIONADOS

Nesta seção são apresentados alguns trabalhos relacionados com o presente trabalho.

2.1 Um Comparativo Entre Frameworks JavaScript Para Desenvolvimento de Aplicações Front-end.

Em (ALMEIDA, 2018) é apresentado um comparativo entre *frameworks* para desenvolvimento de aplicações *web Front-end* em JavaScript. Essa comparação é feita na forma de um *benchmark* entre os *frameworks* selecionados na plataforma do Github.

Os critérios de seleção dos *frameworks* foram Popularidade e Comunidade, o primeiro mensurado pelo número de estrelas no repositório no Github e o segundo pelo número de perguntas no fórum Stack Overflow. Os *framework*s selecionados foram: ReactJS, VueJS e Angular.

Os critérios de comparação escolhidos para o *benchmark* foram: tempo de execução, consumo de memória e CPU. Para a execução do *benchmark* foi proposto um projeto de teste na forma de uma aplicação de CRUD (*create*, *read*, *update* e *delete*) de contatos para cada *framework* selecionado. Para avaliar esses critérios foi proposta uma bateria de casos de teste, listados abaixo:

- Criar 100 contatos
- Editar/Atualizar 100 contatos.
- Apagar 100 contatos.
- Carregar a aplicação com todos os contatos.
- Abrir 10 instâncias da aplicação sem contatos.
- Abrir 10 instâncias da aplicação com 100 contatos.

Os testes foram executados no browser Google Chrome e automatizados na ferramenta Selenium IDE. Na conclusão o autor apresenta gráficos com os resultados dos testes e uma Seção de comentários em relação à documentação e comunidade dos *frameworks* selecionados.

O trabalho de (ALMEIDA, 2018) se assemelha ao presente trabalho por também ser uma comparação entre *frameworks*, porém em detrimento ao trabalho de (ALMEIDA, 2018), neste trabalho é feita a comparação entre micro *frameworks web back-end* escritos em Java, é utilizado apenas um critério de seleção que é o de Popularidade e para comparação são utilizados os critérios de documentação, curva de aprendizado, Popularidade, Suporte a IOC, Suporte a arquitetura REST, Validação da entrada, Artigos publicados e Tamanho da comunidade, além de não ser realizado um *benchmark*.

2.2 Framework comparison method

O trabalho de (GERDESSEN, 2007) foi uma pesquisa financiada pela empresa Everest para saber se era viável ou não migrar os projetos de um framework para outro, no caso o framework utilizado era o Sun BluePrints da Sun Microsystems e desejava-se migrar para o Spring Framework.

Inicialmente em (GERDESSEN, 2007) é questionada a não existência de uma metodologia real para a comparação entre *frameworks*, logo em seguida são propostas três contribuições: A primeira é um modelo teórico para *frameworks* em geral, elaborado a partir de uma revisão da literatura a fim de extrair termos relacionados aos *frameworks* e filtra-los de modo a obter o modelo teórico.

A segunda contribuição é a criação de um método de comparação entre os *frameworks* baseados nesse modelo. São usados os termos que foram citados acima como critérios de comparação. Para o trabalho em questão, foram escolhidos os critérios de modificabilidade e customização.

A terceira contribuição é sugestão de um método para comparar a modificabilidade e a customização com base em diagramas de descrição de recursos, complementados por métricas extraídas do código-fonte de cada *framework*.

A conclusão do autor foi de que valia sim a pena migrar os projetos da Everest do Sun BluePrints para o Spring Framework uma vez eles têm basicamente as mesmas características, entretanto o Spring Framework possui maiores possibilidades de modificabilidade e customização.

O presente trabalho também propõe uma comparação, porém esta é feita entre micro *frameworks web back-end* utilizando diversos critérios de comparação como: documentação,

curva de aprendizado, Popularidade, Suporte a IOC, Suporte a arquitetura REST, Validação da entrada, Artigos publicados e Tamanho da comunidade entre ferramentas desse tipo na linguagem Java, todavia entre ferramentas diferentes e com critérios de seleção.

2.3 Um estudo empírico sobre critérios de seleção de repositório Github

No trabalho de (XAVIER, COELHO e L., 2018) é feita uma revisão sistemática da literatura de artigos publicados em conferências de alto nível da área de Engenharia de Software. Foram revisados manualmente 140 artigos publicados nas conferencias ICSE (*International Conference on Software Engineering*) e FSE (*Symposium on the Foundation of Software Engineering*), com a intenção de responder a duas questões de pesquisa. São elas:

(QP1) Como são selecionados os repositórios Github estudados por artigos de Engenharia Software?

(QP2) Quais a principais características dos *dataset* selecionados?

As contribuições do trabalho de (XAVIER, COELHO e L., 2018) são: (i) mostrar uma análise geral de trabalhos com conjunto de dados coletados do Github, publicados em conferências de alto nível de Engenharia de Software, (ii) prover uma lista de cinco critérios de seleção de *datasets* nos trabalhos analisados, cujo qual são: Popularidade, Atividade, Critérios próprios, Bugs e Idade; e (iii) dar uma breve descrição das principais características destes *datasets*.

Neste trabalho também é feita uma revisão da literatura em busca de critérios de seleção de repositórios de projetos, no entanto este trabalho faz uma comparação entre micro *frameworks web back-end* e é utilizada como único critério de seleção a Popularidade, medida somente pelo número de estrelas no repositório do Github em detrimento da definição dada no trabalho de (XAVIER, COELHO e L., 2018) que inclui também a notoriedade na literatura e número de *forks*.

2.4 Estudo comparativo entre frameworks Java para o desenvolvimento de aplicações web: JSF 2.0, Grails e Spring web MVC.

No trabalho de (FRANCO, 2011) é feita uma comparação dos *frameworks web* para Java mais populares no mercado naquela época. A comparação é feita agrupando as características comuns de cada ferramenta avaliada, como por exemplo, implementação do

padrão MVC, então é realizada uma explanação sobre como cada um dos *frameworks* dá suporte á característica.

Os *frameworks* comparados são: JSF 2.0, Grails e Spring web MVC. Sendo que o JSF 2.0 e Spring web MVC dão suporte ao desenvolvimento somente na linguagem Java e o Grails dá suporte tanto a linguagem Java quanto a linguagem Groovy (uma linguagem alternativa ao Java).

As características selecionadas foram: implementação do padrão MVC, validação, suporte a linguagem Groovy e suporte a AJAX. Também foram expostos dados sobre popularidade, vagas de emprego, livros disponíveis, fóruns de discussão e artigos publicados. Segundo a autora para verificar na prática algumas das características, foi desenvolvido um protótipo de um sistema de controle de finanças pessoais em cada um dos *frameworks*.

O sistema consiste basicamente de um CRUD (*create, read, update* e *delete*) e validações. Como tecnologias de desenvolvimento foram usadas: PostgreSQL e Hibernate para a camada de persistência. A aplicação foi desenvolvida em cada um dos *frameworks* – Para o Grails foi usada à linguagem Groovy.

Como conclusão foi apresentada um compilado das vantagens de cada *framework*, possíveis pontos de dificuldade no desenvolvimento, constatadas no desenvolvimento do protótipo e o *framework* com a melhor curva de aprendizado, o Grails.

O trabalho de (FRANCO, 2011) se assemelha ao presente trabalho na medida em que faz uma comparação entre *web* de desenvolvimento *back-end* em Java, apresentando os pontos em comum de cada ferramenta e como elas cobrem cada ponto.

No entanto o presente trabalho se distingue do trabalho de (FRANCO, 2011) no modo de como foram selecionados os *frameworks* (no caso do deste trabalho, micro *frameworks*). Além de que neste trabalho são comparados os seguintes critérios: documentação, curva de aprendizado, Popularidade, Suporte a IOC, Suporte a arquitetura REST, Validação da entrada, Artigos publicados e Tamanho da comunidade.

No Quadro 1 é disposta uma comparação entre aspectos comuns entre os trabalhos relacionados e o presente trabalho.

Quadro 1 – Comparação entre os trabalhos

Trabalhos	Frameworks	Critérios de seleção	Critérios de comparação
(ALMEIDA, 2018)	ReactJS, VueJS e Angular.	Número de estrelas no repositório do Github e questões no fórum Stack Overflow	Benchmark, medido pelo tempo de resposta e uso de memoria e CPU.
(GERDESSEN, 2007)	Sun BluePrints e Spring Framework	Pré-selecionados pela entidade financiadora	Modificabilidade e customização
(XAVIER, COELHO e L., 2018)	Não se aplica	Número de estrelas no repositório do Github, *forks*, *issues*, *pull requests*.	Não se aplica
(FRANCO, 2011)	JSF 2.0, Grails e Spring web MVC.	Pesquisa de mercado pelos mais usados na época.	Implementação do MVC, Validação, Suporte ao AJAX, Suporte ao Groovy, Popularidade, Vagas de emprego, Livros, Fóruns, Artigos.
Presente Trabalho	Spark, Jooby, Pippo.	Número de estrelas no repositório da plataforma do Github.	Documentação, Curva de aprendizado, Popularidade, Suporte a IOC, Validação, Tamanho da comunidade, Suporte a arquitetura REST.

Fonte: Elaborado pelo Autor.

3 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os principais conceitos deste trabalho, bem como seu impacto sobre as atividades nele desenvolvidas. São descritos os conceitos sobre Desenvolvimento *Web*, Arquitetura *Back-end* e *Front-end*, *Framework web back-end* e Micro *framework web back-end.*

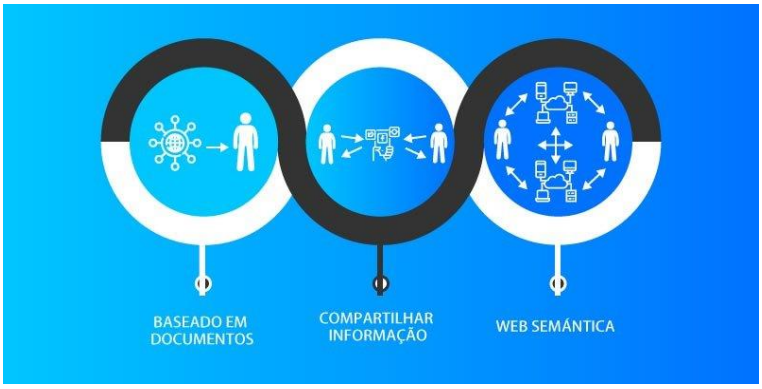
3.1 Desenvolvimento Web

Desde sua criação em 1990 por Tim Berners-Lee a *web* tem como objetivo a ligação de todos os tipos de informações independentemente de onde elas estejam localizadas. Essa primeira versão, que tinha como finalidade ser usada por cientistas para criar e armazenar documentos estáticos foi conhecida como a *web* 1.0 (BERNERS-LEE, 1990).

Em 2004 foi criado e popularizado pela empresa americana O'Reilly Media o termo *web* 2.0 que trata a *web* como uma plataforma de desenvolvimento de aplicações, pois foi nessa época que teve início a criação de *softwares* para essa área. Agora as páginas da *web* tinham conteúdo dinâmico. A regra de ouro dessa nova plataforma era tornar o *software* melhor à medida que os usuários o usam, aproveitando-se da inteligência coletiva (O'REILLY, 2005). Diferentemente do *desktop*, os *softwares* para a *web* deveriam ser o mais modular possível, de modo que fosse fácil adicionar ou remover funcionalidades, reusar módulos e compartilhar recursos (MOREIRA, 2009).

Dada a constante evolução da *web* (PATRIOTA e PIMENTA, 2008), foi criado mais um termo, a *web* 3.0. O responsável por tal nomenclatura é John Markoff, em um artigo publicado na revista The New York Times. Ele diz que todo o conhecimento já acumulado da *web* 2.0 agora seria organizado e usado de forma mais inteligente, de uma maneira que seja possível analisar e ligar dados para obter um novo fluxo de informações. A *web* 3.0 também é conhecida como *Web* Semântica ou *Web* inteligente (MARKOFF, 2006). Na Figura 1 é mostrada uma representação conceitual partindo da *web* 1.0 até a *web* 3.0.

Figura 1 – Representação dos conceitos de web 1.0, 2.0 e 3.0.



Fonte: (IDEAL MARKETING, 2018).

Já a ideia da *web* 4.0 é criada em 2007 por Seth Godin e alguns outros estudiosos como sendo um imenso sistema operacional inteligente e aberto a variados tipos de interação com o usuário (SETH GODIN, 2007). O conceito de *web* 4.0 é ilustrado na Figura 2.

Figura 2 – Representação do conceito de *web* 4.0

Fonte: (IDEAL MARKETING, 2018).

Dado esse contexto é possível perceber a evolução e importância da *web* e o impacto que a mesma tem na vida dos seres humanos. Igualmente, o desenvolvimento de novas aplicações com o máximo de rapidez e qualidade para essa plataforma também é de fundamental relevância (MOREIRA, 2009).

3.2 Back-end e Front-end

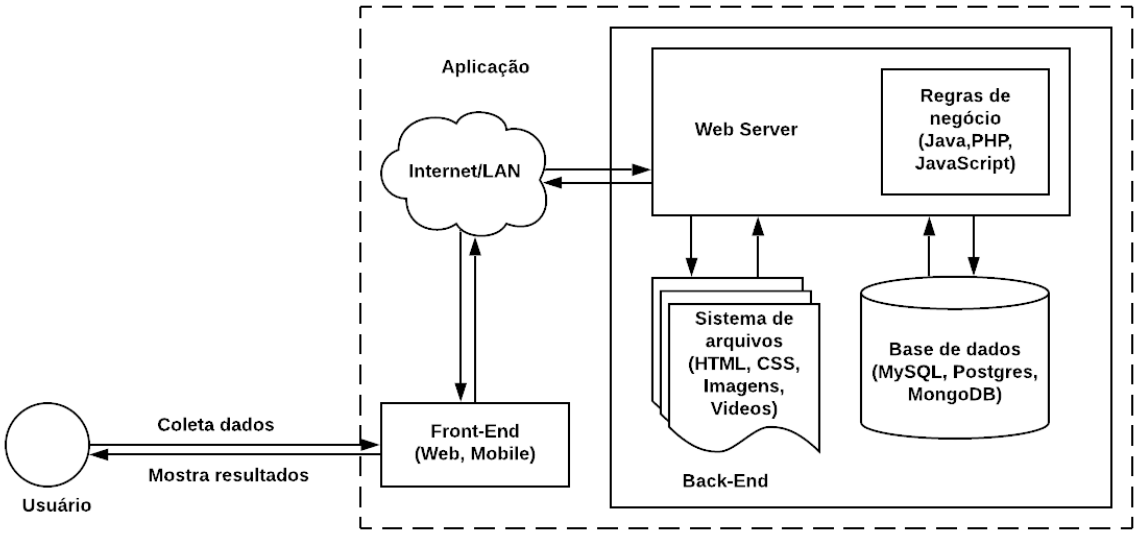
No inicio, a *web* se resumia apenas a paginas estáticas escritas em uma linguagem de marcação desenvolvida especificamente para esse fim, o HTML, de modo que somente existia a camada de apresentação (BERNERS-LEE, CAILLIAU e GROFF, 1992). No decorrer dos anos surgiram novas linguagens que atendiam a algumas necessidades da *web*, como o CSS para estilo e o JavaScript para comportamento que de acordo com (EIS, 2015) são os pilares do *front-end* moderno.

O *front-end* pode ser entendido como a camada de apresentação da aplicação ou camada visual, a que contem uma interface amigável com a qual o usuário interage. No caso de aplicações desenvolvidas para a plataforma *web*, trata-se do visual das páginas *web* (ALMEIDA, 2018).

Como já citado na Seção 3.1 deste trabalho, a partir da *web* 2.0 as páginas apresentavam conteúdos dinâmicos e aplicações “no lado do servidor” que geravam e serviam essas a páginas. Tal conceito é denominado de *back-end*.

O *back-end* é um termo utilizado para denominar a camada de negócios de um *software*. No contexto de desenvolvimento *web*, diz respeito à aplicação que está “no lado do servidor” (AMARAL e NERIS, 2015). Cabe a esse “lado” o processamento (aplicando as regras de negócio), recuperação e persistência dos dados (SOMMERVILLE, 2011). Na Figura 3 é apresentada uma representação da estrutura descrita acima.

Figura 3 – Exemplo da arquitetura *front-end* e *back-end*



Fonte: Baseada na imagem do (SAXENA, 2018).

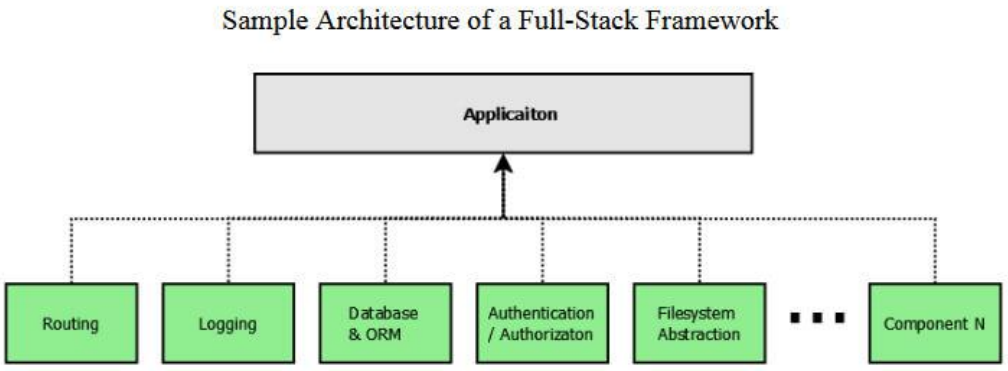
Em comparação com o *front-end* que possui apenas três linguagens, o *back-end* pode ser escrito em toda e qualquer linguagem que tenha suporte para a plataforma *web*, alguns exemplos são: Java (ORACLE CORPORATION, 1991), PHP (LERDORF, 1995) e Python (ROSSUM, 1991).

3.3 Framework web back-end

Renomados autores da literatura de Engenharia de Software como (SCHMIDT e FAYAD, 1997), (MATTSON, 1996), (SOMMERVILLE, 2011) apresentam definições distintas para *Framework*. Por exemplo, para (SCHMIDT e FAYAD, 1997) “Um *framework* é um aplicativo "semi-completo" reutilizável que pode ser especializado para produzir aplicações personalizadas”.

Já para (MATTSON, 1996) um *framework* é “Uma arquitetura (generativa) projetada para reutilização máxima, representada como um conjunto coletivo de classes abstratas e concretas; comportamento potencial encapsulado para especializações sub classificadas”. A Figura 4 demonstra a estrutura de *framework full-stack*.

Figura 4 – Exemplo da arquitetura de um *framework web back-end*.



Fonte: (GIANNINI, 2015).

Em (SOMMERVILLE, 2011) é citada sigla WAF (*Web Application Framework*) que, segundo o autor, agrupam características inerentes a aplicações *web*, como suporte a banco de dados, autenticação, roteamento, disponibilização de páginas *web* dinâmicas etc.

Em síntese para este trabalho podemos definir *framework* como um conjunto de classes e objetos que precisam ser estendidos para criar uma nova aplicação e um WAF como uma especialização para *web* dessa definição. Para este trabalho é usada uma derivação do conceito de *framework*, o micro *framework*, apresentado na Seção seguinte.

3.4 Micro *framework web back-end*

O conceito de micro *framework web* tem suas origens em meados de 2007, como exemplo é possível citar o (SINATRA, 2007), que apesar de esta ser uma DSL (Domian

Specific Language), a sua ideia é a mesma de um micro *framework*: um *framework* de aplicação *web* minimalista, com um núcleo simples, porém extensível.

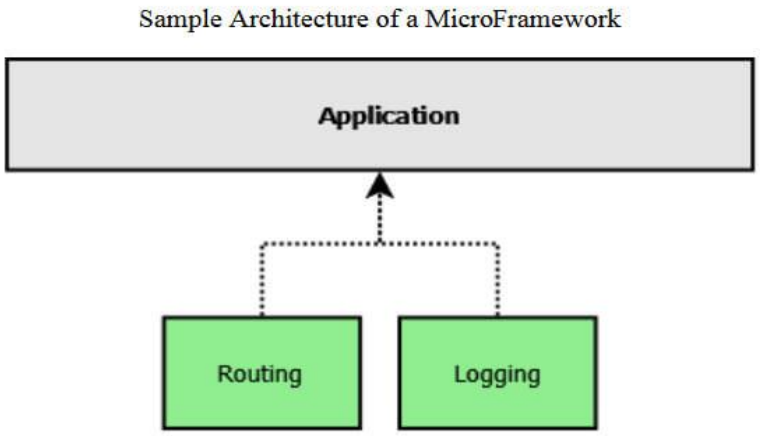
Assim, percebendo a lacuna no mercado de desenvolvimento de *software*, especificamente o da linguagem PHP (PHP: *Hypertext Preprocessor*), Josh Lockhart, na época trabalhando como desenvolvedor de aplicações na New Media Campaigns criou um micro *framework web* para PHP, o Slim Framework (SLIM FRAMEWORK, 2010), considerado o precursor do chamado movimento micro que vinha se expandindo em 2013 (NEW MEDIA CAMPAIGNS, 2014).

Existem poucos autores que dão uma definição clara e aceita do termo micro *framework web*. Em (GIANNINI, 2015, p. 6) fala que "Um micro framework é um framework que tenta fornecer apenas componentes que são absolutamente necessários para um desenvolvedor criar um aplicativo; ou pode se concentrar em fornecer a funcionalidade de uma determinada área de forma muito eficiente".

No trabalho de (SILVA, 2016) é apresentada a seguinte definição: "Micro-frameworks existem como alternativa para quando se deseja a estrutura e velocidade de desenvolvimento fornecida por um framework, porém com menos componentes e menor overhead de um full-stack framework".

É possível sintetizar essas definições da seguinte forma: Um micro *framework web back-end* é uma pequena parte de um *framework* de aplicação *web* de *back-end* convencional. A ideia pode ser comparada com a de uma estrutura em LEGO: dado um conjunto de peças base, pode-se optar por adicionar mais peças de diferentes cores, tamanhos e formas. Analogamente, dado o conjunto de módulos-base para criar a aplicação *web*, podemos adicionar mais módulos com outras funcionalidades a essa estrutura. A Figura 5 exemplifica esse conceito.

Figura 5 – Exemplo da arquitetura de um micro *framework web back-end*.



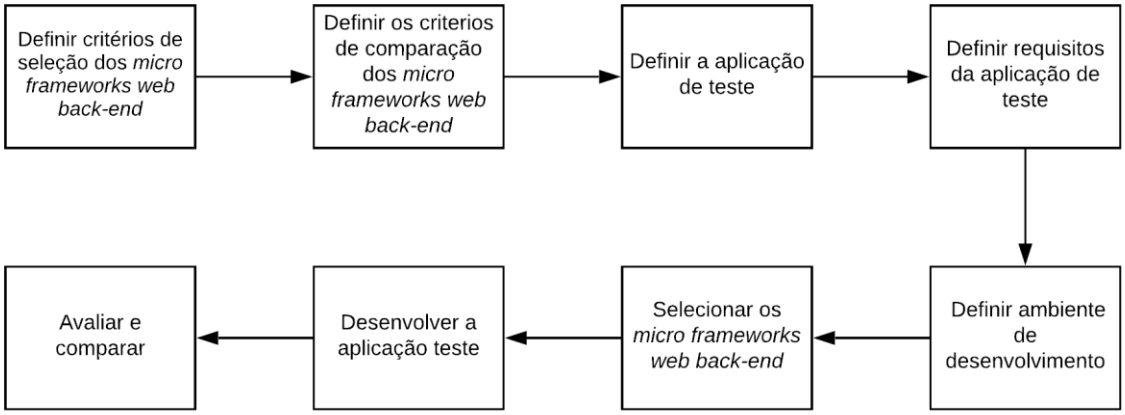
Fonte: (GIANNINI, 2015).

Uma diferença dessa estrutura em relação à dos *frameworks* convencionais se da por causa do seu número de módulos, enquanto os convencionais possuem muitos módulos, dos quais para certas aplicações são desnecessários, os micro *frameworks* contem apenas os módulos essenciais para criar uma aplicação *web* e podem-se adicionar módulos para diferentes funções. (SILVA, 2016).

4 MATERIAIS E MÉTODOS

Nesta seção serão apresentados os procedimentos metodológicos utilizados para a execução deste trabalho de pesquisa. Na Figura 6 é mostrado o passo a passo que é executado no presente trabalho.

Figura 6 – Procedimentos metodológicos



Fonte: Elaborada pelo Autor.

4.1 Definição dos critérios de seleção dos micro frameworks web

A primeira atividade a ser executada neste trabalho, é a definição dos critérios de seleção das ferramentas assim, é realizada uma revisão bibliográfica com ênfase em artigos científicos sobre comparação de *softwares* publicados em no máximo cinco anos, assumindo como base de busca principalmente o Google Scholar, IEEEXplore, ResearchGate e conferências da área de Engenharia de *Software*.

A finalidade dessa busca é encontrar critérios relevantes para a seleção dos micro *frameworks web* de *back-end* a serem comparados. Critérios relevantes no contexto desse trabalho são os critérios com a maior ocorrência nos trabalhos avaliados na revisão bibliográfica, com uma quantidade mínima de um até um máximo de cinco, com base nos resultados do trabalho de (XAVIER, COELHO e L., 2018).

A busca dos micro *frameworks web back-end* usando esses critérios de seleção tem como base de dados à plataforma do Github, dada a sua importância no desenvolvimento de diversos estudos na área de Engenharia de Software, como afirma (XAVIER, COELHO e L., 2018).

Um critério de seleção que pode ser percebido logo de inicio é o de que os micro *frameworks* selecionados devem de ser de código aberto, visto que, o Github é uma plataforma de armazenamento descentralizado de código aberto.

4.2 Definição dos critérios de comparação dos micro frameworks web

Nesta etapa também é realizada uma revisão bibliográfica tomando como prioridade a busca por artigos científicos, monografias, teses e dissertações sobre comparação de *frameworks web back-end*, com o proposito de encontrar critérios significativos para a comparação dos micro *frameworks web* selecionados. Tais critérios são aqueles que têm a maior frequência de uso nos trabalhos revisados, tendo um limite mínimo de seleção de 2 (dois) critérios.

Essa revisão também é feita tendo como base de busca o Google Scholar, IEEEXplore, ResearchGate e conferências da área de Engenharia de *Software*. É necessário atentar-se ao fato de que para um critério encontrado nos trabalhos selecionados seja escolhido é preciso que o mesmo descreva a forma de avaliação e seja relevante no contexto de *frameworks web*, como por exemplo “validação da entrada de dados do usuário”.

4.3 Definição da aplicação de teste

A aplicação de teste escolhida será desenvolvida utilizando os micro *frameworks web back-end* resultantes da seleção feita na Seção 4.1 deste trabalho e servirá, à medida que se fizer necessário, como objeto de estudo na avaliação dos critérios de comparação.

4.4 Definição dos requisitos da aplicação de teste

A aplicação de teste necessariamente tem que contemplar alguns dos critérios de comparação através da definição dos seus requisitos, por exemplo, se um dos critérios for “validação dos dados de entrada” a aplicação precisa ter em pelo menos uma parte da sua implementação algo que demonstre esse critério.

4.5 Definição do ambiente de desenvolvimento

Para pesquisadores que eventualmente usem este trabalho como base não é necessário seguir a risca todas as ferramentas citadas nesta seção. Todas as ferramentas aqui descritas são as usadas no ambiente de desenvolvimento da aplicação de teste. São descritas no Quadro 2.

Quadro 2 – Ferramentas usadas

Nome	Tipo	Versão	Descrição
Java	Linguagem de programação	OpenJDK 8	Linguagem em que os micro *frameworks* foram desenvolvidos.
Maven	Java Build Tool	3.6.0	Sistema de gerenciamento de dependências e automação de tarefas
Eclipse IDE	IDE	Photon	Ferramenta de codificação.
Insomnia	*REST Client*	7.0.5	Ferramenta de teste de requisições HTTP
PostgreSQL	Banco de dados	9.5	Banco de dados relacional.
pgAdmin	Ferramenta gráfica	4.0	Ferramenta gráfica usada para melhor visualizar os dados do banco de dados.

Fonte: Elaborada pelo Autor.

4.6 Seleção dos micro frameworks web back-end

Com os critérios de seleção já definidos na Seção 4.1 é elaborada uma *string* de busca com base no nesses critérios. Só então realizado uma busca na plataforma do Github por repositórios que atendam a estes critérios de seleção.

Em seguida é feito uma avaliação de todos os repositórios resultantes da pesquisa utilizando essa *String* de busca, para descobrir quais se encaixam na definição de micro *framework web back-end* apresentada na Seção 3.4 da fundamentação teórica deste trabalho.

Os outros passos mais práticos deste trabalho como: desenvolver a aplicação de teste, avaliar e comparar, são apresentados na seção seguinte.

5 RESULTADOS

5.1 Critérios de seleção

Após a revisão da literatura, foi constatado que um critério relevante para seleção de repositórios na plataforma do Github é o de Popularidade mensurada pelo número de estrelas. É possível considerar também o número de *forks*, *issues*, *pull requests* (XAVIER, COELHO e L., 2018) e número de questões no fórum do Stack Overflow, como feito em (ALMEIDA, 2018), porém para este trabalho é usado apenas a Popularidade mensurada pelo número de estrelas.

Com base nos trabalhos citados anteriormente a seleção dos micro *frameworks web back-end* se dão pelos seguintes critérios:

- Deve ser um micro *framework web back-end*.
- Deve ter o número de estrelas no repositório da plataforma do Github igual ou superior a 600.
- Deve ser na escrito na linguagem de programação Java.
- Deve ser de código aberto.
- Deve ter a documentação em inglês.

Neste passo a constatação se o projeto avaliado é ou não um micro *framework web back-end* se dá a partida da definição dada na fundamentação teórica deste trabalho na Seção 3.4.

Em relação ao número estrelas, foram realizadas buscas por números maiores como, 700, 800 e até 1000 estrelas. A razão pela qual o valor de 600 estrelas foi escolhido é porque são retornados 20 repositórios.

5.2 Critérios de comparação

Após uma revisão bibliográfica foram selecionados de alguns critérios de comparação baseados em trabalhos anteriores na literatura, No Quadro 3 são listados esses critérios assim como os seus respectivos trabalhos.

Quadro 3 – Resultado da busca por critérios de comparação.

Critério	**Escala de avaliação**	**Onde foi encontrado**	**Forma de avaliação**
Documentação	Ruim, Regular, Boa e Muito Boa.	(SANTOS, 2007)	Verificar se é completa, clara e objetiva
Curva de aprendizado	Curta, Média, Longa e	(SANTOS, 2007) e	Verificar o grau esforço de

	Muito Longa.	(ALMEIDA, 2018)	aprendizagem.
Popularidade	Baixa, Média e Alta.	(FRANCO, 2011) e (ALMEIDA, 2018)	Medida pela ferramenta do Google Trends e N° de estrelas no Github
Suporte a *IOC*	Se disponível sim ou não	(FRANCO, 2011)	Como é realizado
Suporte a arquitetura *REST*	Se disponível sim ou não	(COUTO e FOSCHINI, 2015)	Como é realizado
Validação da entrada	Se disponível sim ou não	(FRANCO, 2011)	Como é realizado
Artigos publicados	Nenhum, Pouco e Muitos.	(FRANCO, 2011)	Artigos no Baeldung, Medium e Devmedia.
Tamanho da comunidade	Inexistente, Pequena, Média e Grande.	(COUTO e FOSCHINI, 2015)	Número de questões no GUJ e StackOverflow.

Fonte: Elaborado pelo Autor.

5.3 Aplicação de teste escolhida

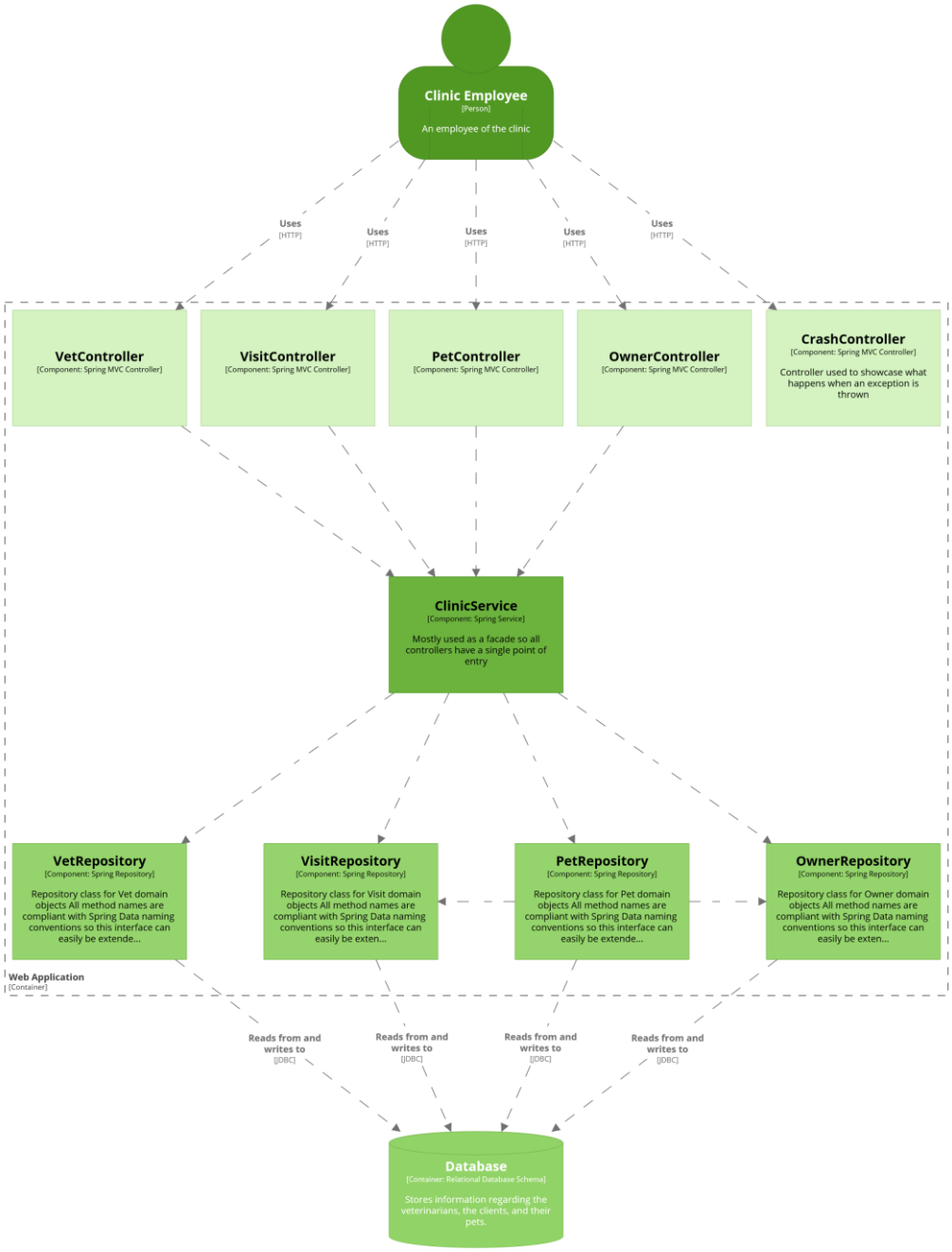
A aplicação escolhida para este trabalho foi a do PetClinic, desenvolvida pela empresa Pivotal, criadora do *framework* Spring Framework. Essa aplicação foi desenvolvida para demostrar as principais funcionalidades do próprio Spring.

As razões por trás dessa escolha são que: ela é uma aplicação com requisitos já bem definidos, é amplamente implementada no mundo Spring e não é preciso preocupar-se com outras coisas a não ser com o seu desenvolvimento.

O PetClinic é uma aplicação de uma clínica veterinária, em que os seus usuários são os próprios funcionários da clínica que, no decorrer de seu trabalho, precisam visualizar e gerenciar informações sobre os veterinários, os clientes e seus animais de estimação.

A aplicação original é pensada para ser *web* no padrão MVC, com a camada de visualização junto da camada de negócios. Para este trabalho isto não é interessante, pois foram desenvolvidas APIs REST utilizando as ferramentas selecionadas. Na Figura 7 é mostrado o diagrama de componentes da aplicação original.

Figura 7 – Diagrama de componentes da aplicação



Fonte: Structurizr

5.4 Requisitos da aplicação de teste

Neste trabalho a aplicação de teste escolhida já possui requisitos bem definidos, dispostos no site da aplicação. Como visualizado no Quadro 4:

Quadro 4 – Requisitos da aplicação de teste

Requisito	Descrição
RQ1	Ver uma lista de veterinários e suas especialidades
RQ2	Ver informações relativas ao proprietário de um animal de estimação
RQ3	Atualizar as informações referentes ao proprietário de um animal de estimação
RQ4	Adicionar um novo proprietário de animal de estimação ao sistema
RQ5	Ver informações relativas a um animal de estimação
RQ6	Atualizar as informações referentes a um animal de estimação
RQ7	Adicionar um novo animal de estimação ao sistema
RQ8	Exibir informações relativas ao histórico de visitação de um animal de estimação
RQ9	Adicionar informações referentes a uma visita ao histórico de visitas do animal de estimação

Fonte: Elaborado pelo Autor.

5.5 Micro frameworks web back-end selecionados

A seleção dos micro *frameworks* foi feita tendo como base os critérios de seleção descritos na Seção 5.1, desses critérios foi elaborada uma *string* de busca para a pesquisa na plataforma do Github, abaixo a *string* de busca:

(*String* de busca) “web framework language: Java stars: 600”

Foram encontrados inicialmente 20 repositórios. Como pode ser observado no Quadro 5 são apresentados repositórios retornados da busca, junto com sua avaliação de acordo com os critérios de seleção e a razão para a sua seleção ou não.

Quadro 5 – Resultado da busca por repositórios no Github.

Nome do projeto	Repositório do Github	Documentação	Selecionado	Razões para a seleção (ou não)
VasSonic	https://github.com/Tencent/VasSonic	No Github	Não	Desenvolvimento hibrido (Android e *web*)
Spark	https://github.com/perwendel/Spark	http://Spark.com/documentation	Sim	Documentação deixa claro que é um micro *framework*
WebMagic	https://github.com/code4craft/webmagic	http://webmagic.io/	Não	É um *crawler*
hsweb framework	https://github.com/hs-web/hsweb-framework	https://docs.hsweb.io/framework/	Não	Contém um *framework* ORM embutido.
Spring MVC showcase	https://github.com/spring-projects/spring-mvc-showcase	https://spring.io/docs	Não	É um *showcase*
Java JFinal	https://github.com/jfinal/jfinal	http://www.jfinal.com/	Não	Contém um *framework* ORM embutido.
WebCollector	https://github.com/CrawlScript/WebCollector	No Github	Não	É um *crawler*
Nutz	https://github.com/nutzam/nutz	https://nutzam.com/	Não	Documentação não está em inglês
AndServer	https://github.com/yanzhenjie/AndServer	https://github.com/yanzhenjie/	Não	Desenvolvimento

		AndServer		hibrido (Android e web)
Ninja Framework	https://github.com/ninjaframework/ninja	http://www.ninjaframework.org/	Não	*Framework full stack*
Vaadin	https://github.com/vaadin/framework	https://vaadin.com/	Não	Contém elementos de UI
Rapidoid	https://github.com/rapidoid/rapidoid	https://www.rapidoid.org/	Não	Documentação não deixa claro
Resty	https://github.com/Dreampie/Resty	https://dreampie.gitbooks.io/resty-chs/content/index.html	Não	Documentação não está em inglês
AlmasB	https://github.com/AlmasB/FXGL	http://almasb.github.io/FXGL/	Não	É uma *game library*
Jooby	https://github.com/jooby-project/jooby	https://jooby.io/v1/	Sim	Documentação deixa claro que é um micro *framework*
Android-Charts	https://github.com/limccn/Android-Charts	https://www.lidaren.com/osf/androidcharts	Não	É para Android
Latke	https://github.com/b3log/latke	https://hacpai.com/tag/latke	Não	Contém um *framework* ORM embutido.
Citrus (webx)	https://github.com/webx/citrus	http://www.openwebx.org/	Não	Não há informações suficientes.
Pippo	https://github.com/pippo-java/pippo	http://www.pippo.ro/	Sim	*Framework full stack*
Spring rest data	https://github.com/spring-projects/spring-data-rest	https://spring.io/projects/spring-data-rest	Não	Usado para teste e documentação

Fonte: Elaborado pelo Autor

Duas observações devem ser levantadas. Primeira: o “Não” marcado na cor laranja quer dizer que aquele projeto não foi selecionado, porém com ressalvas. Na documentação da ferramenta diz: “O Rapidoid é um servidor HTTP extremamente rápido e um contêiner de aplicativos / framework web Java moderno, com um forte foco em alta produtividade e alto desempenho.” Que são característica de um micro *framework*, entretanto em nenhum local consta explicitamente a informação de que ele é um micro *framework* assim como nos outros selecionados, por isso ele foi descartado.

Segunda: o projeto intitulado de Latke não possui mais um repositório no Github até a data de escrita deste trabalho, ou seja, se essa *string* de busca for executada novamente, na data atual de escrita deste trabalho, serão retornados somente 19 repositórios. O mesmo não representa ameaça a validade da pesquisa, pois como consta no Quadro 4, ele foi descartado.

Como pode ser observado no Quadro 4 foram selecionados três micro *frameworks web back-end*, são eles:

- Spark (SPARKJAVA, 2011)
- Jooby (JOOBY, 2014)
- Pippo (PIPPO, 2014)

5.6 Detalhes sobre os micro frameworks selecionados

Nesta seção são apresentadas particularidades de cada micro *framework* selecionado bem como detalhes de implementação da aplicação de teste em todas as ferramentas.

5.6.1 Spark

O Spark é um micro *framework web* para o desenvolvimento de aplicações Java 8 e Kotlin com mínimo de esforço. É apresentada no próprio site da ferramenta uma pequena introdução a respeito da mesma, sendo ela:

> O Spark é uma DSL de framework da Web Java / Kotlin simples e expressivo criado para desenvolvimento rápido. A intenção do Spark é fornecer uma alternativa para desenvolvedores de Kotlin / Java que desejam desenvolver seus aplicativos da Web o mais expressivo possível e com o mínimo de *boilerplate*. Com uma filosofia clara, o Spark foi projetado não apenas para torná-lo mais produtivo, mas também para melhorar seu código sob a influência da sintaxe elegante, declarativa e expressiva do Spark. (SPARKJAVA, 2011).

O Spark dá suporte a duas linguagens: Java e Kotlin. Na linguagem Java, a versão usada é a 8, pois a ferramenta faz uso de algumas *features* dela como, funções Lambda e a Streams API. Em se tratando de gerenciadores de dependências é dado suporte a duas ferramentas desse tipo: Maven e Gradle (SPARKJAVA, 2011).

Embora o Spark tenha uma estrutura inicial simples e limpa como demonstrado no "*hello world*" da Figura 8, indo além dessa estrutura pode ser necessário o uso de bibliotecas extras para dar suporte a algumas outras necessidades complexas ineretes as aplicações. Para (MALHOTRA, 2019) "O Spark é um *framework* Java maravilhoso e limpo, mas requer muito mais trabalho além de apenas criar uma boa camada RESTful".

Figura 8– Aplicação *"hello world"* no Spark com Java.

```java
import static spark.Spark.*;

public class HelloWorld {
    public static void main(String[] args) {
        get("/hello", (req, res) -> "Hello World");
    }
}
```

Fonte: http://Spark.com/

O modulo *core* do Spark é composto por dois elementos: o Jetty, que é *web server* embarcado e uma ferramenta de *logging*.

5.6.2 Jooby

O Jooby é um *micro-framework* web moderno, dito com bom desempenho (TECHEMPOWER, 2019) e fácil de usar, para Java 8 e Kotlin, tendo como opções de gerenciadores de dependências o Maven e o Gradle. Diferentemente do Spark, além de possuir um melhor desempenho, o Jooby tem em seu *core* um módulo de *Hot-reload* e dá suporte á três *web servers* diferentes, que são eles: Jetty, Netty e Undertow (JOOBY, 2014).

Outra característica dele em detrimento aos demais, é que com o Jooby é possível criar um novo projeto usando um esqueleto oficial de projeto presente no site da ferramenta. Pode-se optar também por usar um projeto "*starter*" na criação de um novo projeto. A Figura 9 mostra um exemplo da aplicação "*hello world*" com Jooby.

Figura 9 - Aplicação "*hello world*" no Jooby com Java.

```java
import org.jooby.Jooby;

public class App extends Jooby { // 1

  {
    // 2
    get("/", () -> "Hello World!");
  }

  public static void main(final String[] args) {
    run(App::new, args); // 3. start the application.
  }

}
```

Fonte: https://jooby.io/v1/quickstart/

5.6.3 Pippo

O Pippo é um micro *framework* de código aberto na linguagem Java, com dependências mínimas e uma rápida curva de aprendizado. O Pippo pode ser usado em aplicações pequenas, médias e em aplicações baseadas na arquitetura de microsserviços. O micro *framework* é baseado no Java Servlet 3.1 e requer o Java 8. O tamanho do núcleo é pequeno (cerca de 140 KB) e segundo o criador é uma pretensão dele mantê-lo assim, o menor/mais simples possível e com decorrer do tempo, adicionar novas funcionalidades em módulos pippo e repositórios/módulos de terceiros (PIPPO, 2014).

Não há uma obrigatoriedade em usar um *web server* embarcado especifico. No Pippo existem algumas opções de implementações nativas da ferramenta prontas para o uso são eles: Jetty, Undertow, Tomcat e TJWS.

Além disso, somente com seu *core*, o Pippo já é totalmente funcional que o torna excelente para dispositivos embarcados (Raspberry Pi, por exemplo). A Figura 10 mostra a aplicação "*hello world*" com o Pippo.

Figura 10 - Aplicação "*hello world*" no Pippo.

```
public class HelloWorld {

    public static void main(String[] args) {
        Pippo.send("Hello World!");
    }

}
```

Fonte: http://www.pippo.ro/doc/hello-world.html

5.7 Detalhes sobre a implementação da aplicação de teste

No desenvolvimento da aplicação de teste em cada um dos micro *frameworks* foi reutilizada a implementação original do Spring PetClinic que pode ser encontrado nesse repositório https://github.com/spring-projects/spring-petclinic. Algumas classes foram totalmente alteradas e outras tiveram pequenas mudanças a fim de remover elementos inerentes ao *framework* Spring Framework.

Das classes domínio, na camada de *model* foram retiradas apenas algumas dependências do Spring e modificados os trechos onde elas eram usadas. Nas camadas de *Controller* e

Repository tudo foi modificado, pois para este trabalho cada micro *framework* precisa implementar os seus próprios *Controllers*.

No *Repository* as classes utilizavam a dependência do Spring Data, então tudo foi mudado para ser utilizado o JPA. Após isso, essa camada foi reutilizada nas implementações do PetClinic nos micro *frameworks* selecionados sem mais modificações.

5.8 Avaliação e comparação dos micro frameworks web back-end

5.8.1 Documentação

A análise da documentação de um software pode esclarecer varias coisas sobre ele, não somente o seu funcionamento. De acordo com (WILLEMMANN e IBARRA, 2007) “um projeto de *software* precisa ter uma boa documentação para poder ser considerado um *framework*”.

Como diz também (SANTOS, 2007) “O custo de difusão do conhecimento na equipe de desenvolvimento depende em parte da complexidade das novas tecnologias a serem adotadas bem como da documentação disponível.” Dada ai a importância da documentação e de sua análise em estudos sobre um determinado *framework*.

Na análise foi considerado apenas a documentação e tutoriais oficiais presentes nos sites de cada micro *framework* avaliado. Como apresentado na Seção 5.2 é utilizado a seguinte escala: Ruim, Regular, Boa e Muito Boa.

5.8.1.1 Spark

No Spark a documentação encontrada no site do micro *framework* contém muitos exemplos de código e todos eles funcionam. Para cada *feature* oferecida pela ferramenta existe um exemplo de código que demonstra a sintaxe de forma fácil e padronizada, mostrando que quando se usa aquele recurso se faz daquela maneira.

O Spark possui bastante tutoriais oficiais no site. Cada tutorial demonstra um “*how to do*” (como fazer, em português) de forma clara e objetiva, que vai, por exemplo, desde executar um simples *deploy* no Heroku até mesmo como usar o Spark em conjunto com o Raspberry Pi, o que mostra que os tutoriais próprios do site cobrem boa parte dos eventuais usos do micro *framework*.

Os tutoriais do site são divididos em: simples/curto e intermediário/longo estes às vezes, cobrem pontos que não são abordados na documentação, o que pode ocasionar algum atraso na hora da busca. Quase tudo pode ser encontrado no próprio site da ferramenta.

No desenvolvimento da aplicação de teste neste micro *framework* puderam ser percebidos todos os pontos positivos apresentados anteriormente. Por possuir uma documentação oficial bem detalhada, clara, objetiva, com bons exemplos de código, tutoriais diversos que cobrem desde o básico ao avançado e isso tudo no próprio site, sendo dito como oficial aliado ainda a outras fontes de pesquisa contidas na internet, a documentação do Spark é classificada como: Boa.

5.8.1.2 Jooby

Em relação o Jooby, o mesmo possui duas versões ativas que podem ser usadas sem qualquer problema. Para este trabalho foi usado à versão v1 do micro *framework* por ser um pouco mais madura e estável que a v2.

A documentação oficial contém para cada funcionalidade disponibilizada da ferramenta um trecho de código que demonstra seu uso, entretanto em detrimento ao Spark o Jooby não possui tutoriais que demonstram sua aplicação em outros possíveis casos.

Ao invés disso ele apresenta o que ele chama de "*Starter projects*" (contidos em um repositório do Github) que são projetos parcialmente configurados e desenvolvidos usados como base para a criação de aplicações, como por exemplo, para a criação de um projeto em Kotlin que usa o Gradle ele disponibiliza o "*Starter Project*" "jooby-project/kotlin-gradle-starter" no repositório do Github.

O desenvolvimento da aplicação de teste utilizando o Jooby foi complicado em alguns aspectos. É bastante fácil entender o funcionamento da ferramenta, mas à falta de exemplos na documentação oficial ocasionou um grande atraso na codificação da aplicação de teste.

Um exemplo factível de dificuldade percebida durante o desenvolvimento é que o uso de *Filters* não é suportado quando se trabalha com o padrão MVC e isso não está escrito em nenhum local da documentação. Foi necessária uma busca minuciosa nas *issues* fechadas do micro *framework* para constatar isso.

Outra dificuldade ocasionada também pela ausência de exemplos de código na documentação oficial é que, para o desenvolvimento de aplicações utilizando o padrão arquitetural MVC pouco ou nenhum exemplo de código confiável é disposto no site.

Levando em consideração todos os fatores citados acima, é possível perceber que ainda existe uma pequena deficiência em relação à documentação do micro *framework* Jooby, apesar de existir bons exemplos para a construção de APIs de *script*. A classificação da documentação é: Regular.

5.8.1.3 Pippo

Relação à documentação do micro *framework* Pippo, assim como todos os outros este também possui exemplos para cada funcionalidade, porém em comparação aos outros a documentação mostrou-se muito mais simplificada e objetiva.

Não há nem tutoriais nem *starters* na documentação oficial, ao invés disso existe os demos, que são, como sugere o nome, aplicaçãoes de demonstração, como por exemplo uma aplicação de CRUD. No desenvolvimento da aplicação de teste utilizando esta ferramenta, não houve maiores dificuldades além da adaptação a sintaxe da ferramenta.

Por todos os fatos explanados acima, aliado ao fato que a documentação supriu inteiramente todas as duvidas surgidas no desenvolvimento da aplicação de teste a documentação do micro *framework* Pippo é classificada como: Muito Boa.

Assim, no Quadro 6 é sumarizado os resultados da avalição da documentação.

Quadro 6 – Resultados da avaliação da documentação dos micro *frameworks*.

Micro *framework*	**Avaliação da documentação**
Spark	Boa
Jooby	Regular
Pippo	Muito Boa

Fonte: Elaborado pelo Autor.

5.8.2 Curva de aprendizado

A curva de aprendizado é avaliada com base no trabalho de (SANTOS, 2007) e (ALMEIDA, 2018) que afirmam que boa documentação e conhecimento prévio do desenvolvedor influenciam diretamente nesse fator.

5.8.2.1 Spark

No Spark a curva de aprendizado é avaliada considerando alguns fatores, como por exemplo:

- Documentação dita como Muito Boa
- Não existem *annotations* próprias do micro *framework.*
- Fácil de usar, sintaxe simples, configuração mais simples ainda.
- Não há a necessidade de ter conhecimentos específicos para usa-lo, basta saber a sintaxe do Java 8.

Existe também o uso de expressões lambda na construção de uma aplicação usando o padrão MVC. Para desenvolvedores que não estão familiarizados com esse recurso isso pode vir a se tornar um ponto de dificuldade. A Figura 11 demonstra o uso de Lambdas no Spark.

Figura 11 – Trecho de código da aplicação de teste com o Spark

```
12  public class OwnerController {
13          public static Route addOwnerEndPoint = (Request req, Response res) -> {
```

Fonte: Elaborado pelo Autor.

Para o desenvolvimento no padrão MVC a sintaxe é parecida com a uma *function expresssion* do JavaScript, onde o valor de uma função lambda é armazenado em uma variável, nesse caso em um objeto do tipo *Route*. Esses pontos puderam ser percebidos no desenvolvimento da aplicação de teste. Por isso a curva de aprendizado do Spark é classificada como: Curta.

5.8.2.2 Jooby

Os fatores levados em consideração no Jooby foram os mesmos do Spark com exceção da não existência de *annotations*, o micro *framework* Jooby possui sim *annotations*, muitas delas, o que torna ele em relação aos demais bem mais verbosos. A Figura 12 mostra o uso dessas *annotations*.

Figura 12 – Trecho de código da aplicação de teste com o Jooby

```
33  @Path("/owner")
34  @Produces("application/json")
35  @Consumes("application/json")
36  public class OwnerController {
37
38          @GET
39          @Path("/all")
40          public List<Owner> getAllOwners() {
41                  return ownerRepo.getAllOwners();
42          }
```

Fonte: Elaborado pelo auto.

Por isso a curva de aprendizado do Jooby é classificada como: Média.

5.8.2.3 Pippo

Na avaliação da curva de aprendizado do Pippo foram considerados os mesmos pontos do Jooby, ou seja, o Pippo também contém *annotations*, só que diferentemente do Jooby ele faz uso também de herança para diminuir o número de *annotations*. As figuras 13 e 14 mostram fragmentos do código da aplicação de teste que exemplificam isso.

Figura 13 – Trecho de código de um *Controller* da aplicação de teste com Pippo.

```
16  @Path("/owner")
17  public class OwnerController extends Controller{
```

Fonte: Elaborado pelo Autor.

Figura 14 – Fragmento de código de um método da aplicação de teste com Pippo.

```
26          @GET("/all")
27          @Produces(Produces.JSON)
28          public List<Owner> getAllOwners() {
```

Fonte: Elaborado pelo Autor.

Por isso a curva de aprendizado do Pippo é classificada como: Curta

No Quadro 7 são sumarizadas essas avaliações.

Quadro 7 – Resultado da avaliação da curva de aprendizado.

Micro *framework*	**Curva de aprendizado**
Spark	Curta
Jooby	Média
Pippo	Curta

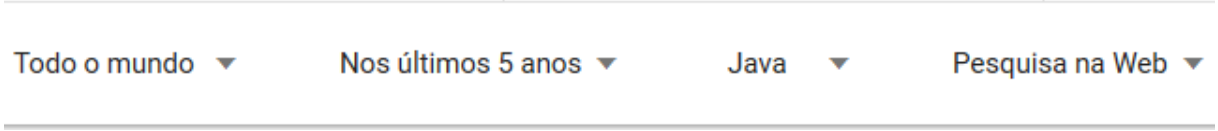
Fonte: Elaborado pelo Autor.

5.8.3 Popularidade

Como determinado na Seção 5.2 a popularidade é medida usando o Google Trends, uma ferramenta da Google que verifica a frequência das pesquisas a determinados termos de busca. Foi usado também o número de estrelas no repositório do projeto na plataforma do Github. Foi necessário personalizar os parâmetros da busca no próprio Google Trends para que este retornasse resultados consistentes para a pesquisa, mostrados na Figura 15. Abaixo a lista de modificações.

1) Mudar o alcance para “Todo o Mundo”
2) Mudar o tempo para “Nos últimos 5 anos”
3) Selecionar a grande área como sendo “Java”

Figura 15 – Parâmetros de busca do Google Trends para esta pesquisa



Fonte: Google Trends

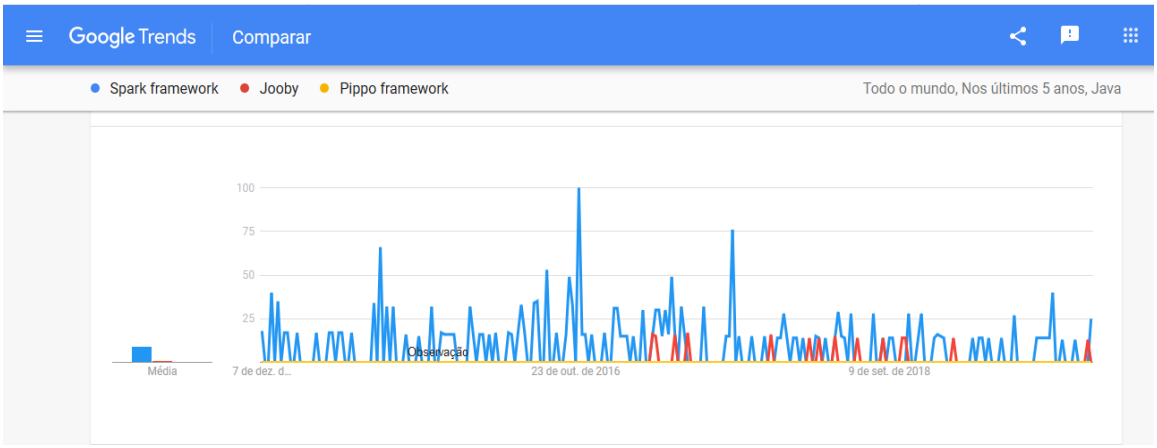
Para cada micro *framework* foi elaborado uma *string* de busca tentando ser o mais preciso possível em relação aos resultados. No Quadro 8 é mostrado cada termo de busca para cada micro *framework* e na Figura 16, o resultado da comparação do Google Trends.

Quadro 8 – Micro *frameworks* e os termos de busca usados.

Micro *framework*	**Termo de busca**
Spark	“Spark”
Jooby	“Jooby”
Pippo	“Pippo framework”

Fonte: Elaborado pelo Autor.

Figura 16 - Resultados da comparação com o Google Trends



Fonte: Google Trends.

Também foi avaliado o número de estrelas nos repositórios de cada micro *framework* na plataforma do Github. No Quadro 9 são apresentado estes os dados.

Quadro 9 – Micro *framework* e o número de estrelas em cada repositório.

Micro *framework*	Número de estrelas no repositório
Spark	8.6k
Jooby	1.1k
Pippo	718

Fonte: Elaborado pelo Autor.

Com base na Figura 16 e no Quadro 9, o micro *framework web back-end* Spark foi considerado o mais popular. O micro framework web back-end Jooby aparece em segundo. Por fim o micro *framework web back-end* Pippo foi o menos popular. Como pode ser observado na Figura 16 micro *framework web back-end* Pippo não obteve resultados expressivos na pesquisa do Google Trends. Um dos motivos para isso pode ser atribuído ao fato de que ele é relativamente novo. No Quadro 10 é ilustrado esse resultado.

Quadro 10 – Resultado da avaliação da Popularidade.

Micro *framework*	Popularidade
Spark	Alta
Jooby	Média
Pippo	Baixa

Fonte: Elaborado pelo Autor.

5.8.4 Suporte a Inversão de controle

A inversão de controle (IOC) pode ser entendida como a inversão do fluxo de controle da aplicação que a utiliza. As responsabilidades de iniciar a execução da aplicação, instanciar as dependências dos objetos e finalizar a execução aplicação são retiradas do poder do programador e passadas para o controle de um terceiro, seja ele um módulo, uma classe ou o

mais comum, um *container*. Esse fenômeno é conhecido também como “Principio de Hollywood – Não ligue para nós, ligaremos para você.” (FOWLER, 2005).

5.8.4.1 Spark

O Spark não oferece suporte nativo à inversão de controle. Ao invés disso é recomendado utilizar dependências estáticas instanciadas na classe principal que deverão usadas ao decorrer do desenvolvimento da aplicação. A Figura 17 monstra isso.

Figura 17 – Exemplo da “IOC” do Spark.

```
public class Application {

    // Declare dependencies
    public static BookDao bookDao;
    public static UserDao userDao;

    public static void main(String[] args) {

        // Instantiate your dependencies
        bookDao = new BookDao();
        userDao = new UserDao();
```

Fonte: http://Spark.com/tutorials/application-structure

A justificativa dada pelo próprio criador é a de que, com a inversão de controle, geralmente feita através da injeção de dependências não se obtém nenhum beneficio real, fazendo apenas com que se tenha uma repetição desnecessária de código. Em relação aos testes de unidade (que é o argumento principal para o uso de injeção de dependências) o criador diz que para o teste de controladores, testes de aceitação são superiores aos de unidade uma vez que eles testam a aplicação no estado exato em que ela será implantada.

No desenvolvimento da aplicação de teste com o Spark foi utilizada a abordagem recomendada na documentação oficial, ou seja, a apresentada nos parágrafos anteriores, porque não havia necessidade de se fazer testes unitários na aplicação.

5.8.4.2 Jooby

No Jooby o suporte a inversão de controle é feito através da injeção de dependências com o uso da *annotation* "@Inject" do modulo interno que utiliza o *framework* Guice. Não é possível usar outro *framework*. A Figura 18 exemplifica isso.

Figura 18 – Exemplo de IOC do Jooby

```
@Path("/")
public class MyRoutes {

  @Inject
  public MyRoutes(DepA a, DepB) {
    ...
  }
```

Fonte: https://jooby.io/v1/doc/#routes-mvc-routes

No desenvolvimento da aplicação de teste com o Jooby, no tocante a inversão de controle foi utilizada a mesma estratégia usada no Spark, ou seja, dependências estáticas, pois se fosse usado o modulo próprio do Jooby as aplicações ficariam muito diferentes e muito provavelmente isso influenciaria nos resultados das avaliações sobre essas aplicações.

5.8.4.3 Pippo

Já no Pippo, o suporte a inversão de controle também é feita com injeção de dependências. Para isso ele possui três módulos com implementações nativas que usam containers de DI para executar esse processo, são eles: Spring, Guice e CDI (Contexts & Dependency Injection for Java). Para o desenvolvimento da aplicação de teste, na parte de inversão de controle, novamente e pelo mesmo motivo foi utilizada a abordagem do Spark. No Quadro 11 são sumarizados os resultados desse critério de comparação

Quadro 11 – Resultados da avaliação do suporte a inversão de controle

Micro *framework*	**Suporte a IOC**
Spark	Não disponível
Jooby	Disponível
Pippo	Disponível

Fonte: Elaborado pelo Autor.

5.8.5 Suporte a arquitetura REST

Atualmente tem se tornado cada vez mais difícil desenvolver uma aplicação sem considerar pelo menos uma vez a sua integração com outras plataformas e aplicações. Para isso existem inúmeras formas consolidadas de se fazer tal integração, porém dentre todas elas

uma vem ganhando destaque, a abordagem da arquitetura REST. Uma definição é apresenta a seguir.

> REST é o acrônimo de REpresentational State Transfer (Transferência de Estado Representativo) e pode ser definido como um conjunto de boas práticas de uso do protocolo HTTP para expor webservices, ou seja, serviços responsáveis por executar determinada lógica ou regra de negócio, geralmente utilizados para integração entre aplicações. (COUTO e FOSCHINI, 2015).

5.8.5.1 Spark

O Spark dá suporte à arquitetura REST de forma muito simples e fácil. Para tal é necessário que o projeto possua como umas de suas dependências uma biblioteca de manipulação de JSON (JavaScript Object Notation), por exemplo, o GSON do Google, em cada método do *Controller* declare explicitamente que o conteúdo da resposta será JSON, informando também o *statusCode* da requisição e finalmente retornando um objeto para ser serializado. A Figura 19 demonstra um exemplo desse uso.

Figura 19 – Exemplo de uso do REST com Spark.

```
public static Route searchOwnerEndPoint = (Request req, Response res){
        res.type("application/json");
        if (ownerRepo.findOwnerByName(req.params(":name")).size() == 0) {
                res.status(404);
                return new MessageJson("Owner not found");
        }
        res.status(200);
        return ownerRepo.findOwnerByName(req.params(":name"));
};
```

Fonte: Elaborado pelo Autor.

5.8.5.2 Jooby

O Jooby também dá suporte à arquitetura REST, porém ele possui módulos nativos de bibliotecas de manipulação de JSON especificas, são elas: Jackson, Gson e Yasson. Para que as requisições e respostas sejam realizadas usando JSON é necessário usar as *annotations* “@Produces(‘application/json’)” e “@Consumes(‘application/json’)” (isso pode ser visualizado na Figura 9) em cada *Controller* e ao decorrer da codificação dos métodos, jogar exceções do tipo “Err(statusCode, message)” no caso de erros, informando o *statusCode* e a mensagem de erro. A Figura 20 demonstra isso.

Figura 20 - Exemplo de uso do REST com Jooby.

```
44          @POST
45          @Path("/save")
46          public Owner addOwner (@Body Owner o) {
47                  if(o != null) {
48                          try {
49                                  ownerRepo.saveOwner(o);
50                          } catch (GenericException e) {
51                                  e.printStackTrace();
52                                  throw new Err(Status.BAD_REQUEST, e.getMessage());
53                          }
54                          return o;
55                  }
56                  throw new Err(Status.SERVER_ERROR, "Something wrong happened!!");
57          }
```

Fonte: Elaborado pelo Autor.

5.8.5.3 Pippo

No Pippo, no que diz respeito à manipulação dos tipos de conteúdos existem classes nativas da ferramenta que tratam disso, as chamadas *content type engines*. Para JSON o Pippo possui três: “GsonEngine”, “FastJsonEngine” e “JacksonJsonEngine”. Cada uma usando uma biblioteca de manipulação de JSON diferente.

É possível também criar um *content type engine* customizado, bastando apenas implementar a interface “ContentTypeEngine”. Para registrá-lo existem duas formas: declarar na classe da aplicação que irá usá-la através do método “registerContentTypeEngine” ou criar um inicializador.

Para que seja efetivamente retornado o JSON é obrigatório o uso da *annotation* “@Produces (Produces. JSON)” em cada método que precisar retornar um JSON. A manipulação do *statusCode* é feita através do método “getResponse. status (statusCode)”. A Figura 20 ilustra essa funcionalidade.

Figura 21 – Exemplo de uso do REST com o Pippo.

```
88          @GET("/find/{name}")
89          @Produces(Produces.JSON)
90          public List<Owner> findOwner(@Param("name") String name) {
91                  try {
92                          getResponse().status(200);
93                          return ownerRepo.findOwnerByName(name);
94                  } catch (IndexOutOfBoundsException e) {
95                          e.printStackTrace();
96                          getResponse().status(500);
97                          return null;
98                  }
99          }
100 }
```

Fonte: Elaborado pelo Autor.

No Quadro 12 são sumarizados os resultados desse critério de comparação.

Quadro 12 – Resultados da avaliação do suporte a arquitetura REST.

Micro *framework*	**Suporte a arquitetura REST**
Spark	Disponível
Jooby	Disponível
Pippo	Disponível

Fonte: Elaborado pelo Autor.

5.8.6 Validação

A validação dos dados da entrada de um sistema deve ser considerada pelo desenvolvedor como uma parte muito importante no desenvolvimento de um *software*, já que ela garante alguns benefícios consideráveis, como por exemplo, consistência dos dados no banco de dados, boa qualidade do produto de software desenvolvido e fluxo continuo e fluido de navegação na aplicação, uma vez que os dados transitados na aplicação são válidos. (BONFIM, RAMOS, *et al.*, 2010).

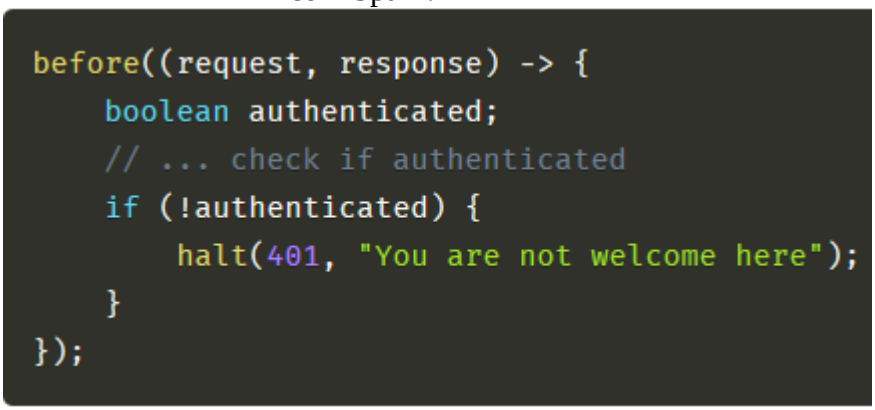
5.8.6.1 Spark

No Spark a validação da entrada de dados do usuário pode ser feita através de um recurso nativo da própria ferramenta denominado *Filter*. Existem dois tipos de *Filters* os *before* (antes) e os *after* (depois), nesse caso o “antes” e o “depois” são em relação às requisições HTTP.

É possível optar entre um filtro para todas as requisições ou um filtro para rotas especificas. Existe ainda uma sutil diferença de sintaxe no desenvolvimento de APIs de script

e utilizando o padrão MVC. Nas Figuras 22 e 23 são mostrados exemplos mais detalhados para APIs de script.

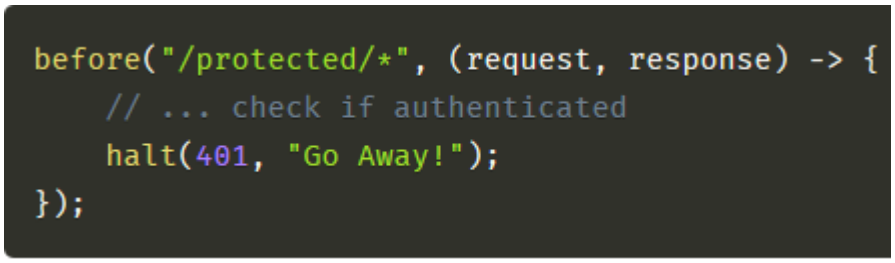
Figura 22 – Exemplo de *Filter* para todas as requisições em uma API de script com Spark.



```
before((request, response) -> {
    boolean authenticated;
    // ... check if authenticated
    if (!authenticated) {
        halt(401, "You are not welcome here");
    }
});
```

Fonte: http://Spark.com/

Figura 23 – Exemplo de *Filter* para uma rota em uma API de script com Spark.



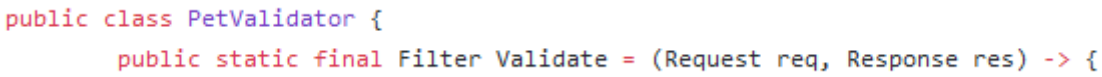
```
before("/protected/*", (request, response) -> {
    // ... check if authenticated
    halt(401, "Go Away!");
});
```

Fonte: http://Spark.com/

Os exemplos de código das Figuras 22 e 23 contém um elemento chamado *halt ()*, sua função é parar imediatamente a requisição retornando para o cliente o *status code* e uma mensagem, ele geralmente é usado em conjunto com um *Filter*.

Em se tratando do desenvolvimento utilizando o padrão MVC o código é escrito seguindo a mesma sintaxe vista anteriormente na Figura 8, a de função lambda. A Figura 24 contém um exemplo de código retirado da aplicação de teste, somente para ilustrar o que já foi falado.

Figura 24 – Exemplo de *Filter* no padrão MVC com Spark.



```
public class PetValidator {
        public static final Filter Validate = (Request req, Response res) -> {
```

Fonte: Elaborado pelo Autor

5.8.6.2 Jooby

Para o Jooby a validação dos dados de entrada do usuário também é feita utilizando *Filters* próprios, com basicamente os mesmos elementos usados no Spark (*berfore* e *after*), porém há uma pequena diferença na sintaxe, a presença do elemento "*chain"*. A Figura 25 demonstra isso.

Figura 25 – Exemplo de um *Filter* no Jooby

```
{
  use("*", "*", (req, rsp, chain) -> {
    before(req, rsp);
    chain.next(req, rsp);
  });
}
```

Fonte: https://jooby.io/v1/doc/#routes-request-handling

O Jooby não oferece suporte para *Filters* no formato convencional das aplicações que utilizam o padrão MVC, formato esse visto na Figura 23.

5.8.6.3 Pippo

No Pippo, assim como nos outros micro *framework* avaliados, são utilizados *Filters* para a validação de entradas. Um *filter* no Pippo é essencialmente um *Route* que não retorna uma resposta e sim dá seguimento na cadeia de execução da aplicação. A sintaxe para *before* e *after filters* muda um pouco em relação aos demais. As Figuras 26 e 27 ilustram os exemplos de *before* e *after* respectivamente.

Figura 26 – Exemplo de código com um *before filter* no Pippo.

```
// before filter that create a database instance
ANY("/.*", routeContext -> {
    routeContext.setLocal("database", getDatabase());
    routeContext.next();
});
```

Fonte: http://www.pippo.ro/doc/filters.html

Figura 27 - Exemplo de código com um *after filter* no Pippo.

```
// after filter that release the database instance
ANY("/.*", routeContext -> {
    Database database = routeContextremoveLocal("database");
    database.release();
}).runAsFinally();
```

Fonte: http://www.pippo.ro/doc/filters.html

No Quadro 13 são sumarizados os resultados desse critério de comparação.

Quadro 13 – Resultados da avaliação do suporte a validação da entrada.

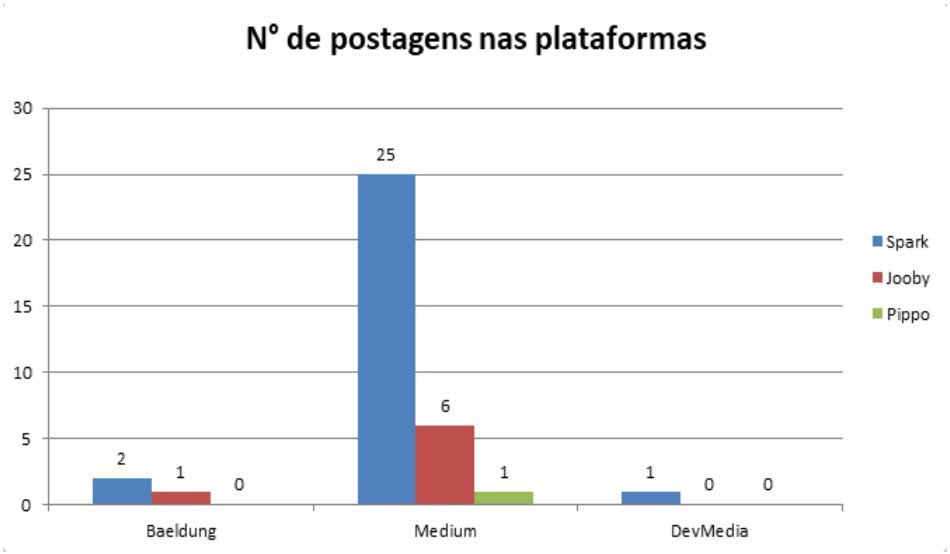
Micro *framework*	**Suporte a validação da entrada**
Spark	Disponível
Jooby	Disponível
Pippo	Disponível

Fonte: Elaborado pelo Autor.

5.8.7 Artigos publicados

Em artigos publicados foram avaliados o número de artigos nos sites do Baeldung, Medium e DevMedia para cada micro *framework* selecionado. Estes sites foram escolhidos, respectivamente, por ser um fórum bastante popular do ecossistema Java, por ser uma plataforma de publicação de blog que vem tendo um destaque e último por ser o maior site para desenvolvedores de língua portuguesa. Foi buscado pelo nome de cada ferramenta. A Figura 28 mostra o resultado.

Figura 28 – Resultado da busca por artigos



Fonte: Elaborado pelo Autor.

Na Figura 28 é possível visualizar que dentre todos os micro *frameworks web back-end*, o Spark obteve a maior quantidade de *posts* na plataforma do Medium, mas não teve tanta expressividade na plataforma do Baeldung, o que pode se considerar estranho, uma vez que essa plataforma tem como foco a linguagem Java. Em segundo vem o Jooby, que também não obteve destaque no Baeldung, mas sim no Medium. Por ultimo o Pippo, em terceiro com apenas um *post* no Medium. No Quadro 14 é mostrado esse o resultado.

Quadro 14 – Resultado da avaliação da quantidade de artigos publicados.

Micro *framework*	**Artigos publicados**
Spark	Muitos
Jooby	Poucos
Pippo	Poucos

Fonte: Elaborado pelo Autor.

5.8.8 Tamanho da comunidade

A comunidade desempenha um papel de fundamental importância na manutenção e evolução de um produto de software. Uma comunidade grande e ativa oferece um *feedback* valioso para a garantia de que cada evolução desse software seja alinhada com as expectativas dos seus usuários. (COUTO e FOSCHINI, 2015).

Na avaliação do tamanho da comunidade foi realizada para cada micro *framework*, uma busca por questões sobre os mesmos nos foruns do GUJ e Stack Overflow. O GUJ é um fórum brasileiro sobre a tecnologia Java, já o Stack Overflow é um fórum internacional sobre programação. O Quadro 15 contém os termos de busca para cada micro *framework* em cada fórum.

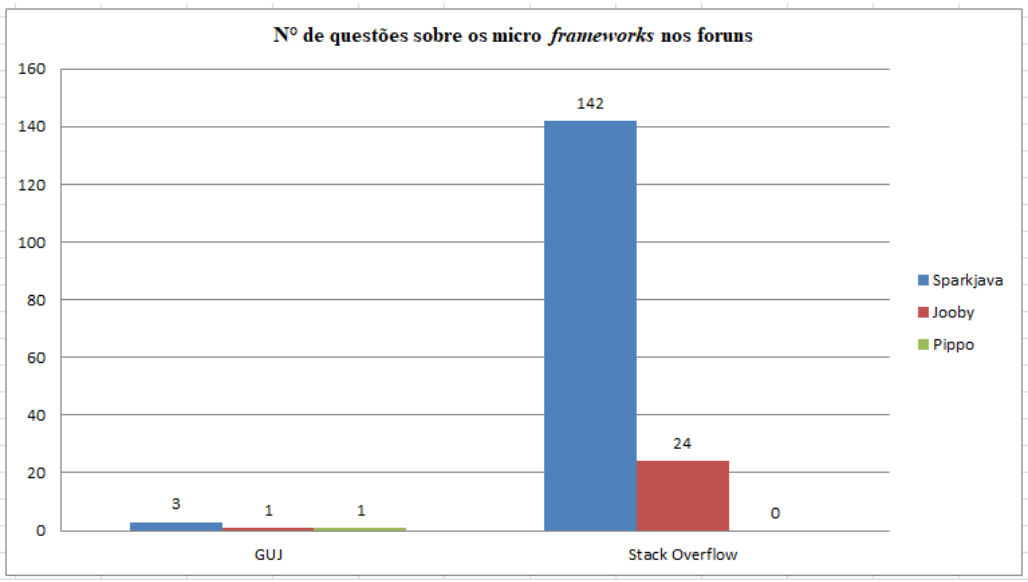
Quadro 15 – Termos de busca para questões nos foruns.

Micro *frameworks*	**GUJ**	**Stack Overflow**
Spark	Spark	Spark [Spark-java] -apache -[apache-Spark]
Jooby	jooby	jooby [jooby]
Pippo	pippo	pippo [pippo]

Fonte: Elaborado pelo Autor.

A Figura 29 mostra os resultados dessas buscas utilzando os termos de busca no GUJ e no Stack Overflow.

Figura 29 – Resultados das pesquisas pelos termos em cada fórum.



Fonte: Elaborado pelo Autor

Na Figura 29 é visto que dentre os três micro *frameworks web back-end* o Spark é o que tem a maior comunidade, ficando em segundo o Jooby e em ultimo o Pippo. Pode-se dizer que esse resultado negativo do Pippo é possivelmente causado por ele ser relativamente novo no mercado. Outro ponto a se observar é que essas ferramentas possuem uma grande comunidade fora do Brasil. No Quadro 16 é mostrado o resultado da avaliação desse critério.

Quadro 16 – Resultado da avaliação do tamanho da comunidade.

Micro *framework*	**Tamanho da comunidade**
Spark	Média
Jooby	Pequena
Pippo	Inexistente

Fonte: Elaborado pelo Autor

6 AMEAÇAS A VALIDADE

6.1 Influência sobre os resultados do Spark

Nos resultados de buscas sobre o Spark, pode ser que tenham sido colhidas informações sobre o Apache Spark, que uma ferramenta de Big Data. Para tentar mitigar essa ameaça foram definidas *strings* de busca precisas.

6.2 Nível de experiência do desenvolvedor

O nível de experiência em desenvolvimento do pesquisador representa uma possível ameaça à validade da pesquisa, pois diferentes níveis de experiência com desenvolvimento podem gerar diferentes resultados.

6.3 Repositório de código base

Os micro *frameworks web back-end* foram selecionados utilizando a plataforma do Github. Porém, se fosse utilizado outra plataforma de armazenamento distribuído de código, possivelmente resultaria na seleção de outros micro *frameworks web back-end*. Para mitigar essa ameaça é utilizado uma plataforma que foi empregada em muitos trabalhos na literatura, como por exemplo (XAVIER, COELHO e L., 2018), (ALMEIDA, 2018) e (COUTO e FOSCHINI, 2015).

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho teve como objetivo principal apresentar um estudo comparativo entre os micro *framework web back-end* em Java mais populares da plataforma do Github, usando para alcançar esse objetivo critérios de comparação relevantes extraidos de trabalhos cientificos. No Quadro 17 é mostrado os resultados.

Quadro 17 – Comparação dos micro *frameworks*.

Critérios	**Spark**	**Jooby**	**Pippo**
Documentação	Boa	Regular	Muito Boa
Curva de aprendizado	Curta	Média	Curta
Popularidade	Alta	Média	Baixa
Suporte a IOC	Não	Sim	Sim
Suporte a REST	Sim	Sim	Sim
Validação da entrada	Sim	Sim	Sim
Artigos publicados	Muitos	Poucos	Poucos
Tamanho da comunidade	Média	Pequena	Inexistente

Fonte: Elaborado pelo Autor.

É esperado que, com a avaliação dos micro *frameworks* nesses critérios, mesmo que estes sejam subjetivos, seja possível auxiliar principalmente os desenvolvedores na escolha dessas tecnologias para um determinado projeto e também prover para quaisquer pesquisador ou grupo de pesquisadores, uma base na área de estudo de micro *framework web back-end*.

A utilização de qualquer um dos micro *frameworks* estudados neste trabalho garantem a obtenção de boa parte dos beneficios que um *framework* de aplicação trás, em qualquer área, seja ela *web*, *mobile*, *desktop* e até mesmo em dispositivos com sistemas embarcados.

Segundo a avaliação do pesquisador o micro *framework web back-end* considerado o melhor dentre os três selecionados é o Spark, pelos seguintes motivos:

- Sua documentação é completa, correta (tem exemplos de código que funcionam) e demonstra variadas formas de uso da ferramenta.
- A curva de aprendizado foi curta, dado o fato que é necessario ter conhecimento apenas nas *features* do Java 8, como por exemplo, funções Lambda.
- No Suporte a IOC apesar de não possuir, o Spark sugere uma alternativa a esse conceito com as dependências estáticas.

Dado que o presente trabalho tem como objeto de estudo micro *frameworks*, onde umas da caracteristicas principais é minimalismo, foi elaborado uma tabela com a demonstração da diferença entre a quantidade de linhas de código das classe da camada de *Controller* da aplicação de teste bem como o tamanho em KB, para ilustrar qual das ferramentas obteve a melhor colocação no quesito supracidato. A Tabela 1 mostra este resultado.

Tabela 1 – Diferença de linhas de código e tamanho em KB das implementações.

Micro *frameworks*	**N° linhas**	**Tamanho em KB**
Spark	149	1.40
Jooby	258	2.49
Pippo	226	1.60

Fonte: Elaborado pelo Autor.

Trabalhos futuros identificados a partir das descobertas deste trabalho sugerem: i) Analisar outros critérios de comparação, como desempenho por exemplo. ii)Analisar mais micro *frameworks* como o Rapidoid e iii) Analisar micro *frameworks* implementados em outras linguagens como por exemplo Python com o Flask.

REFERÊNCIAS

ABES. **Mercado Brasileiro de Software. Panorâma e Tendências**. São Paulo: [s.n.], v. I, 2018.

ALMEIDA, F. E. V. D. **Um comparativo entre frameworks JavaScript para desenvolvimento de aplicações front-end**. Trabalho de Conclusão de Curso bacharelado em Engenharia de Software, Quixadá, p. 45, 2018.

AMARAL, R. A. D.; NERIS, V. P. D. A. **Análise comparativa entre frameworks de front-end para aplicações web ricas visando reaproveitamento do back-end**. Tecnologias, Infraestrutura e Software (TIS), São Carlos, v. IV, n. 1, p. 88-96, Janeiro 2015.

BAGESTAN, F. H**. Estudo comparativo de frameworks PHP, com enfoque no Codeigniter**. Monografia de Graduação, Sistemas para Internet do Instituto Federal Sul-rio-grandense, Campus Passo Fundo**.** PASSO FUNDO, v. I, n. 1, 2016.

BERNERS-LEE, T. Information Management: A Proposal. **CERN**, v. I, p. 1-21, [S.l], Maio 1990.

BERNERS-LEE, T.; CAILLIAU, R.; GROFF, J.-F. The world-wid web. **Computer Networks and ISDN Systems**, Geneva, I, 1992. 454-459.

BONFIM, T. et al. **Padrão para Validação de Entrada de Dados de Usuários**. Universidade Federal do Vale do São Francisco (UNIVASF) , Juazeiro-BA, p. 10, 2010.

COUTO, R. S.; FOSCHINI, I. J. Análise comparativa entre dois Frameworks MVC para a Plataforma Java EE: JSF e VRaptor. **Revista T.I.S (Tecnologias, Infraestrutura e Software)**, São Carlos, v. IV, p. 11, jul. 2015.

EIS, D. **Guia Front-End - O Caminho das Pedras para ser um dev Front-End**. [S.l.]: Casa do Código, v. I, 2015.

FLASK. index. **Flask**, [S.l], 2010. Disponivel em: https://flask.palletsprojects.com/en/1.1.x/. Acesso em: 25 Ago. 2019.

FOWLER, M. **InversionOfControl**. [S.l], 2005. Disponivel em: https://martinfowler.com/bliki/InversionOfControl.html. Acesso em: 07 Dez. 2019.

FRANCO, R. S. T. **Estudo comparativo entre frameworks java para desenvolvimento de aplicações web:** JSF 2.0, Grails e Spring web MVC. (Monografia de especialização) - Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná .Curitiba, p. 90, 2011.

GERDESSEN, A. **Framework comparison method**. Tese de mestrado em engenharia de software pela universidade de Amsterdam, Amsterdam, p. 78, 13 ago. 2007.

GIANNINI, N. J. **Vulnerable Web Application Framework**. University of Rhode Island - DigitalCommons@URI - Open Access Master's Theses, Kingston, v. I, n. 629, p. 1-79, 2015.

IDEAL MARKETING. **Web 4.0**. Ideal Blog, [S.l,], 2018. Disponivel em: https://www.idealmarketing.com.br/blog/web-4-0/. Acesso em: 21 maio 2019.

JOOBY. **index**. Jooby.org. . [S.l], 2014. Disponivel em: https://jooby.org/. Acesso em: 03 maio 2019.

LERDORF, R. **into-whatis**. Php, [S.l], 1995. Disponivel em: https://www.php.net/manual/pt_BR/intro-whatis.php. Acesso em: 27 maio 2019.

MALHOTRA, R. Rapid Java Persistence and Microservices. In: MALHOTRA, R. **Rapid Java Persistence and Microservices**. Berkeley: Apress, v. I, 2019. Cap. 2, p. 9-25.

MARKOFF, J. **Entrepreneurs See a Web Guided by Common Sense**. The New York Times, San Francisco, Nov 2006. Disponivel em: https://www.nytimes.com/2006/11/12/business/12web.html. Acesso em: 21 maio 2019.

MATTSON, M. **Object-Oriented Frameworks. A survey of methodological issues**. Tese (Licenciatura em Ciência da Computação) Departamento de Ciência da Computação, Universidade de Lund, Sweden, p. 128, 1996.

MDN. **Introduction**. MDN web docs, [S.l], 2005. Disponivel em: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction. Acesso em: 14 maio 2019.

MOREIRA, D. D. R. **Um estudo da tecnologia Web 2.0**. (Monografia) - Curso de Bacharelado em Ciência da Computação. Universidade Federal de Goiás Campus Catalão. Catalão, v. I, n. 1, p. 1-101, 2009.

NEW MEDIA CAMPAIGNS. **An-interview-with-the-founder-of-slim-php-framework-our-josh-lockhart**. New Media Campaigns, [S.l], 2014. Disponivel em: https://www.newmediacampaigns.com/blog/an-interview-with-the-founder-of-slim-php-framework-our-josh-lockhart. Acesso em: 02 maio 2019.

ORACLE CORPORATION. **java**. Java, [S.l], 1991. Disponivel em: https://www.java.com/pt_BR/. Acesso em: 27 maio 2019.

O'REILLY, T. **what is web 2.0**. 0reilly, [S.l], 2005. Disponivel em: https://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html. Acesso em: 12 maio 2019.

PATRIOTA, K. R. M.; PIMENTA, R. D. D. H. Da Mídia 2.0 para a Mídia 3.0 perspectivas da próxima onda na Web. **XXXI Congresso Brasileiro de Ciências da Comunicação**. Natal-RN: [s.n.]. 2008. p. 14.

PETRIJEVCANIN , V.; SUDAREVIC, B. Use of Web Application Frameworks in the Development of Small Applications. **Proceedings of the 34th International Convention MIPRO**. Opatija, Croatia.: [s.n.]. 2011. p. 5.

PIPPO. **Home**. Pippo Micro Java Web Framework, [S.l], 2014. Disponivel em: http://www.pippo.ro/. Acesso em: 02 Dez 2019.

PIVOTAL. **Spring Boot**. Spring Boot, [S.l], 2017. Disponivel em: https://spring.io/projects/spring-boot. Acesso em: 23 Nov 2019.

ROSA, T. P. **Um método para o desenvolvimento de software baseado em microsserviços**. Trabalho de Conclusão de Curso bacharelado em Engenharia de Software, Quixadá, p. 66, 2016.

ROSSUM, G. V. **index**. Python, [S.l], 1991. Disponivel em: https://www.python.org/. Acesso em: 27 maio 2019.

SANTOS, T. R. D. **Análise e Comparação de Frameworks para o Desenvolvimento Web em Java**. Monografia de Graduação, Universidade Federal de Santa Catarina, Departamento de Informática e Estatística, Curso de Sistemas de Informação., Florianópoles, p. 60, 2007.

SAXENA, R. **App Architecture – Understanding Frontend, Backend and Web Servers**. City Kitty Design, [S.l], 2018. Disponivel em: http://citykittydesign.com/app-architecture-understanding-frontend-backend-and-web-servers/. Acesso em: 29 maio 2019.

SCHMIDT, D.; FAYAD, M. Object-Oriented Application Frameworks. **Communications of the ACM**, p. 7, set. 1997.

SETH GODIN. **web4**. sethsblog, [S.l], 2007. Disponivel em: https://seths.blog/2007/01/web4/. Acesso em: 23 maio 2019.

SILVA, M. F. **Utilização dos modelos full-stack framework e micro-framework para o desenvolvimento de aplicações web escaláveis em linguagem php**. Monografia de Graduação, Faculdade de Ciência da Computação das Faculdades Integradas de Caratinga, Caratinga, v. I, n. 1, p. 1-77, Dezembro 2016.

SINATRA. **intro**. Sinatra website, [S.l], 2007. Disponivel em: http://sinatrarb.com/intro.html. Acesso em: 02 maio 2019.

SLIM FRAMEWORK. **index**. Slim, [S.l], 2010. Disponivel em: https://www.slimframework.com/. Acesso em: 30 maio 2019.

SOMMERVILLE, I. **Engenharia de Software**. 9°. ed. [S.l.]: Pearson, v. I, 2011.

SPARKJAVA. **index**. Spark, [S.l], 2011. Disponivel em: http://Spark.com/. Acesso em: 03 maio 2019.

STRUCTURIZR. **Spring PetClinic**. Structurizr. [S.l], 2010. Disponivel em: https://structurizr.com/share/1/diagrams#components. Acesso em: 01 Dez 2019.

TECHEMPOWER. **Round 18**. TechEmpower, [S.l], 2019. Disponivel em: https://www.techempower.com/benchmarks/#section=data-r18&hw=ph&test=plaintext&a=2. Acesso em: 07 Dez 2019.

WILLEMMANN, D. P.; IBARRA, G. B. **Framework Java de Apoio ao Desenvolvimento de Aplicações Web com Banco de Dados, utilizando Struts, Tiles e Hibernate**. Monografia de Graduação, Universidade Federal de Santa Catarina, Centro Tecnológico, Departamento de Infromática e Estatística. Florianópolis, v. I, n. 1, p. 1-155, 2007.

XAVIER, L.; COELHO, J.; L., L. S. Um Estudo Empírico sobre Critérios de Seleção de Repositórios GitHub. **Conferência Brasileira de Software (CBSOFT) - VI Workshop de Visualização, Manutenção e Evolução de Software (VEM)**. São Carlos: [s.n.]. 2018. p. 8.