**Metodologia da Pesquisa Científica e Trabalho de Conclusão de Curso *(*TCC)**

**Pós-Graduação *Lato Sensu***

**Análise comparativa entre dois Frameworks MVC para a plataforma JEE:**

**Spring MVC e VRaptor**

**Cezar Toshiaki Nakase**

[cezar.nk@gmail.com](mailto:cezar.nk@gmail.com)

**Prof. MSc. Claudinei Di Nuno**

[professorclaudinei@uol.com.br](mailto:professorclaudinei@uol.com.br)

Curso de Pós-Graduação *Lato Sensu*em Desenvolvimento Orientado a Objetos com Java

UNESA–Universidade Estácio de Sá

# Resumo

O objetivo deste artigo é de apresentar o comparativo dedois *frameworks Java*voltado ao desenvolvimento web, no qual serão o *Spring MVC* e *VRaptor 4.*Com intuito de auxiliar o desenvolvedor na escolha do *framework*, com base na estrutura e necessidade do seu projeto. Neste artigo será apresentada a criação de um sistema utilizando os dois *frameworks*, apresentando como é feito a sua implantação, a realização de operações básicas de um sistema, como por exemplo, o *CRUD* (*Create, Read, Update e Delete*), validações e reaproveitamento de código. Além disso, serão abordados os pontos fortes e fracos dos *frameworks*, a dificuldade em relação à curva de aprendizagem, analise e continuidade da plataforma e suporte do mantenedor. Com base nas informações apresentadas, o artigo servirá como disseminador dos *frameworks Java* para web e de material de apoio para futuras pesquisas e desenvolvimento.

Palavras-chave: *framework*, desenvolvimento web*Java*, padrão MVC,comparativo,*Vraptor* e *Spring MVC*.

# Introdução

Quando iniciamos o desenvolvimento de um projeto web na linguagem *Java*, uma das questões que muitos programadores têm de defini-la, é com qual ferramenta será utilizada para o desenvolvimento do projeto?

E desenvolver sistemas em *Java* para web, antes das criações dos *framework*s, que em resumo é um conjunto de bibliotecas que reúne inúmeras funcionalidades que ficam à disposição do programador, aumentando sua produtividade; era uma tarefa trabalhosa, pois mesmo o *Java* possuindo os *Servlets* e o JSP para auxiliar nos serviços específicos web, as requisições e tratamentos eram feitos de forma manual, onde cada programador criava sua metodologia de criação, bibliotecas e ferramentas com códigos massivos para sanar suas necessidades, assim não havendo uma padronização no desenvolvimento web com *Java*.

Atualmente, temos a disposição vários *frameworks* para facilitar a produtividade e a padronização do código. Porém, devemos analisar se vale à pena utilizar? Como escolher? Qual ganho e vantagem terão ao escolher uma?

Baseado nesses questionamentos, por boa prática, devemos seguir alguns critérios para a escolha do *framework*. Por exemplo, a sua estrutura, pois verificamos se ele poderá atender as necessidades; o quão reconhecido ele é, pois assim podemos ter mais informações através de comunidades e fóruns a respeito de novas idéias, nova funcionalidade e a qualidade do *framework*; segurança, verificar a capacidade de garantir o funcionamento do sistema, o gerenciamento de riscos e redução de vulnerabilidade; e a documentação, pois sendo bem formulada serão mais fáceis o seu entendimento e a sua utilização.

Seguindo as boas práticas citado anteriormente, este artigo vai apresentar os *frameworks Spring MVC* e o *VRaptor4*.

E a relevância da escolha dos frameworks encontra-se da seguinte forma: *Spring MVC –* atualmente, um dos *frameworks* de *Java* para web mais utilizada, além de ser mais completo em questões de bibliotecas e ferramentas e com muitos módulos que facilitam na configuração do projeto. *VRaptor* 4 – um *framework* desenvolvido por brasileiros, levando o conceito de praticidade na criação do projeto por seguir a convenção de *Convention over Configuration,* conceito de redução de arquivos adicionais de configuração, facilitando e agilizando o desenvolvimento do projeto e a manutenção do código; e por possuir total integração com *Java EE.*

Por conseguinte, o objetivo final em servir como um material de pesquisa para desenvolvedores que buscam alternativas de frameworks *Java* para o desenvolvimento do seu projeto web, apresentando comparativo de ambos os frameworks, visando os prós e contras. Tornando mais claro a diferença em seus pontos específicos, facilitando a visualização para escolha do framework de acordo com o projeto.

1. **Fundamentação Teórica**

**2.1 Um pouco da história do Java no mundo web**

Desde o avanço da internet, em meados de 1996, quando o Java iniciou seus primeiros passos ao mundo da web através dos recursos do *Servlets*: Oracle (2013) “uma classe de linguagem de programação Java usada para estender os recursos de servidores que hospedam aplicativos acessados ​​por meio de um modelo de programação de solicitação-resposta”; foi um grande avanço, dando vida à comunicação da linguagem Java, Banco de Dados e página web. Porém, no seu início, não havia uma organização definida entre os códigos, se misturavam códigos Java com os de apresentação (*HTML, CSS e JavaScript*) em um único arquivo. Com o surgimento do JSP (*Java Server Page*), onde o intuito era de organizar e definir as especificações que uma página web deveria seguir. IBM - Noel J. Bergman e Abhinav Chopra  (2001) “O JSP combina HTML e XML com *servlet* Java TM (extensão de aplicativo de servidor) e tecnologias *JavaBeans* para criar um ambiente altamente produtivo para desenvolvimento e implantação de sites confiáveis, interativos e independentes de plataforma de alto desempenho”.Resultando em melhora na questão da separação dos códigos.

Porém, mesmo com essas tecnologias criadas,o desenvolvimento web com Java, ainda não seguia uma estrutura organizada, e a necessidade de padronização ou uma arquitetura eficiente era inevitável.

**2.2 Padrões de projeto para ajudar o desenvolvedor**

Visando solucionar conflitos referentes à padronização no desenvolvimento de um projeto, foram criados conceitos onde reunia as melhores práticas formalizadas por programadores para solução de um problema em comum, técnicas adotadas dentro de um contexto para uma melhor organização, no qual segundo [Christopher Alexander](https://pt.wikipedia.org/wiki/Christopher_Alexander) (*A Times Way of Building,* 1979) devia ter as seguintes características: “Encapsulamento, Generalidade, Equilíbrio, Abstração, Abertura e Combinatoriedade”. Padrões de projetos não são específicos a uma empresa ou tecnologia, são soluções reutilizáveis aplicados pela comunidade de desenvolvimento como um todo. Onde define segundo [Christopher Alexander](https://pt.wikipedia.org/wiki/Christopher_Alexander),“cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira”.

Resumindo a sua importância na utilização de um padrão, o ganho na qualidade do código, a reutilização de soluções para um determinado problema e contribuição de especialistas e desenvolvedores para auxilio de manutenção, pois o padrão representa o conhecimento entre os envolvidos.

Diante desse cenário, o padrão de arquitetura MVC (*Model-View-Controller*) se destacanas mais diversas aplicações, e principalmente em aplicações web, por sua organização, onde são divididas em camadas bem estruturadas em relação aos dados, regras de negócios e interface, além de ser um modelo utilizado por muitos *frameworks*.

O MVC criado em 1979 por Trygve Reenskaug, no qual ele justifica sua criação da seguinte forma: “Eu criei o padrão *Model-View-Controller* como uma solução óbvia para o problema geral de dar aos usuários controle sobre suas informações como visto de múltiplas perspectivas”, transcrevendo o modelo mental humano e o modelo digital do computador.

*Model (*Modelo*):* é a representação da estrutura de dados, camada responsável pela manipulação, consulta e persistência das informações no banco de dados.

*View* (Visão): é a camada responsável pela interface apresentado ao usuário, no qual deve gerir requisições e respostas, através de componentes visuais, por exemplo, formulários, tabelas, menus e botões para entrada e saída de dados e que a visão reflita o estado do modelo.

*Controller* (Controladora): responsável pela comunicação ou fluxo de informação entre a camada do modelo e a camada de visão, é nele onde dizemos quais regras de negócio o sistema deve seguir, quais operações devem ser executadas.

Assim concluindo, o padrão MVC nos proporciona as seguintes vantagens:

- O MVC por trabalhar em multi-camadas, tende a facilitar o gerenciamento do projeto e código de uma forma mais clara, pois mantém explícito o que cada camada deve executar, tornando o código mais limpo;

- É possível o desenvolvimento das camadas do projeto em paralelo ou escalável, além de maior integração da equipe em questão da divisão de tarefas;

- Possibilidade do reaproveitamento ou re-usabilidade do código de uma forma mais fácil, podendo incluir bibliotecas ou adicionando interfaces no projeto;

- Redução na dificuldade na manutenção do software, pois são realizadas as correções e alterações separadamente, não afetando outras camadas do sistema;

- Diversidade de frameworks ou tecnologias que estão utilizando essa metodologia de padrão de projeto.

**2.3 *Spring* MVC, popular e com muitos recursos**

Após uma breve explicação sobre a história do Java para web, padrões de projetos e modelo MVC, o assunto abordado será sobre o framework *Spring,*mais especificamente o *Spring* MVC, no qual se refere justamente ao módulo web.

**Um pouco da história, estrutura e funcionamento do Spring**

Uma resposta a complexidade das primeiras especificações do J2EE (Java2 Platform Enterprise Edition), surgiu em 2003, criado por Rod Johnson, descrito em seu livro “Expert One-To-One J2EE Design and Development”.

*Spring*é organizado de forma modular, permitindo que você se preocupe apenas com os módulos necessários. Atualmente estão disponíveis 21 módulos, fornecendo tudo o que um desenvolvedor pode precisar para o uso no desenvolvimento de um projeto. WEISSMANN Henrique Lobo (2014, pg. 27) Ressalta a importância dos módulos “É importante termos esta visão panorâmica dos módulos que compõem o framework para que fique claro o quão abrangente ele é: basicamente o *Spring* abrangetodas as necessidades de uma aplicação corporativa”.

*Spring* é baseado na estrutura de POJOS (*Plain Old Java Object*), um objeto Java não limitado por nenhuma restrição, sem a necessidade de implementar ou estender classes pré-especificadas na estrutura. Segue o padrão de IOC (*Inversion of Control –* Inversão de Controle), um fluxo do controle do sistema é invertido, permitindo que indique outro elemento o controle do método dizendo quando deve ser executado. Para KAYAL, Dhrubojyoti (2008, pg. 23) “O contêiner de Inversão de Controle de Mola (IOC) é o coração de todo o framework. Isso ajuda a unir as diferentes partes do aplicativo,formando assim uma arquitetura coerente.” Um dos métodos utilizados pelo IOC é a injeção de dependência, no qual separa um objeto de suas dependências, deixando o foco da classe apenas nos recursos para realizar as tarefas que precisa, e possamos decidir quais dependências serão injetadas durante o tempo de execução. Uma das características mais conhecidas quando programamos com *Spring*.

**Por que *Spring MVC?***

Descrito por HEMRAJANI, Anil da seguinte forma “O Spring Web MVC Framework é uma estrutura robusta, flexível e bem projetada paraaplicativos da web em rápido desenvolvimento usando o padrão de design MVC.alcançado usando este módulo Spring são semelhantes àqueles que você obtém do resto doSpring Framework”.

Por ser um dos mais completos frameworks com diversos componentes de auxílio, além de ser um dos mais utilizado para desenvolvimento web com Java.   
É verdadeiramente abrangente e expansivo que é mais do que capaz de assumir qualquer tarefa ou projeto em potencial no qual você deseja trabalhar.

Devido à modularidade da própria ferramenta, isso permite que você escreva códigos muito limpos e acessíveis. Há uma enorme quantidade de documentação excelente e uma comunidade próspera que o ajudará se você tiver dúvidas ou preocupações sobre como fazer certas coisas ou como certas coisas funcionam ou qualquer coisa desse tipo.

Possui as funcionalidades necessárias para processar as requisições HTTP, gerenciar os componentes para o processamento de dados além de processar e apresentar a resposta da requisição, seguindo o padrão MVC.

Além de possuir uma documentação robusta, uma comunidade ativa onde possui uma página exclusiva no site do *StackOverflow,* apenas com perguntas relativas ao *Spring* MVC e um amplo pacote de ferramentas abrangente para qualquer tipo de projeto que você possa ter.

**2.4 *VRaptor* para agilizar o desenvolvimento**

Diferente de outros frameworks, *VRaptor*, criado em 2004, pelos brasileiros Paulo Silveira e Guilherme Silveira,apresenta-se como uma alternativa eficiente e com a proposta de sua simplicidade e por trazer a ideologia de ser rápido e de fácil aplicação. E desde esse período, houve novas versões e melhorias, acompanhando as melhores práticas de desenvolvimento no mercado e desempenho; atualmente o *framework* encontra-se na versão *4,* trazendo total integração com *Java* EE 7, mas sempre mantendo sua proposta. *VRaptor.org (2018) “* O VRaptor 4 traz alta produtividade para um desenvolvimento Java Web rápido e fácil com CDI”.

O que seria o CDI e qual vantagem da utilização desse componente ao *VRaptor*?

CDI - *Contexts and Dependency Injection* (Injeção de Dependência e Contextos), em sua versão 1.0 introduzida junto a plataforma *Java* EE 6 eatualmente na versão 2.0 continuada no *Java* EE 8; possui um conjunto de serviços com intuito de melhorar a estrutura do código visando a produtividade, fornece uma arquitetura uniforme para injeção de dependência e o gerenciamento do ciclo de vida de beans. De acordo com o site oficial do CDI fica claro o objetivo e ganhos no caso de sua utilização. *Cdi-spec.org* (2018):

- Um ciclo de vida bem definido para objetos com estado ligados a contextos de ciclo de vida, em que o conjunto de contextos é extensível.

- Um mecanismo de injeção de dependência sofisticado e seguro de tipos, incluindo a capacidade de selecionar dependências no tempo de desenvolvimento ou de implantação, sem configuração detalhada.

- Suporte para modularidade *Java* EE e a arquitetura de componente *Java* EE - a estrutura modular de um aplicativo *Java* EE é levada em conta ao resolver dependências entre componente Java EE.

Além disso, o *VRaptor,*trabalha com o conceito de estrutura MVC e integra com as arquiteturas atuais como o REST *– Representational State Transfer* (Transferência de Estado Representacional) e *ActionBased,* trazendo consigo mais benefícios em sua utilização;

Um exemplo, ao utilizar a arquitetura REST, Arnon Rotem-Gal-Oz (2012, pg. 234), faz uma observação positiva em sua utilização: “O termo implica integração fácil e rápida, frequentemente usando APIs e fontes de dados para produzir resultados enriquecidos que não era necessariamente a razão original para produzir os dados da fonte bruta. As principais características do *mashup* são combinação, visualização e agregação. É importante tornar os dados existentes mais úteis, além disso, para uso pessoal e profissional. ”

Em questão da arquitetura *ActionBased*, que suportam as requisições de entrada através de controladores de ações. Paulo Silveira, Guilherme Silveira, Sergio Lopes, Guilherme Moreira, Nico Stepat, Fabio Kung (2011, pg. 152 e 153) orienta da seguinte forma “Podemos preferir trabalhar orientados a requisições e respostas com características stateless para melhor aproveitar a Web e outras ferramentas que giram em torno do HTTP, além de garantir escalabilidade e disponibilidade mais facilmente. Nesse tipo de situação, usar um framework action-based costuma se encaixar melhor como solução”.

O *framework VRaptor* possui a característica de flexibilidade, permitindo a possibilidade de sobrescrever praticamente quase todos os seus comportamentos, sem a necessidade das configurações em XML.Assim, o desenvolvedor ganha autonomia para fazer ajustes e configurações específicas de acordo com seu projeto. Essa facilidade é descrita por Lucas Cavalcanti (2014, pg. 3), mostrando a importância da seguinte forma “mesmo os problemas mais complexos e necessidades mais específicas dos projetos conseguiram ser resolvidos sobrescrevendo o comportamento do *VRaptor* usando os meios normais da sua API, ou sobrescrevendo um de seus componentes”.

Finalizando, o *VRaptor* tem se empenhado em atrair e conquistar desenvolvedores, através dos conceitos citado acima, mantendo sempre seu foco de extensibilidade e desenvolvimento de aplicações o mais fácil e produtivo.

1. **Materiais e Métodos**

**VRaptor na prática**

Pré requisito JDK 7 e o CDI 1.1 ou superiores, criando projetos Download disponível pelo <https://bintray.com/caelum/VRaptor4/br.com.caelum.vraptor/> possui 2 opções:

- Vraptor Blank Project é um projeto preparado com mínimo necessário para rodar o VRaptor, usando o Maven para gerenciar as dependências

- Na página de download possui também o zip de distribuição, que contém a distribuição completa da última versão do VRaptor. Nesse zip podemos encontrar o jar do VRaptor, suas dependências (pasta lib), seu javadoc (pasta apidoc) e código fonte (pasta src). Assim já é possível linkar esses artefatos na sua IDE (Eclipse, Netbeans, etc.) e facilitar o desenvolvimento.

- Servidores suportados Wildfly 8, Tomcat 7 e Jetty 8.

- Para manipulação dos dados junto ao banco de dados, recomendável a utilização do Hibernate.

O Hibernate é um framework **ORM - Object Relational Mapping**. É uma ferramenta que nos ajuda a persistir objetos Java em um banco de dados relacional. O trabalho do desenvolvedor é definir como os objetos são mapeados nas tabelas do banco e o Hibernate faz todo o acesso ao banco, gerando inclusive os comandos SQL necessário.

O VRaptor é um Framework MVC para desenvolvimento rápido de aplicações WEB que faz uso das anotações e conceitos de inversão de controles e injeção de dependência. Outros conceitos como o de Convenção do Invés de Configuração e Active Record tornam o desenvolvimento bastante produtivo sem perder flexibilidade.

VRaptor trabalha na camada de Controller, ele é quem controla as entradas e dispara as requisições internas para os controllers e suas views.

Quando uma requisição chega, ela é prontamente atendida pelo VRaptor. O framework então decide qual action chamar. A action executa, e ao final, diz ao framework qual JSP exibir. O VRaptor por fim, pega a JSP, a processa, e envia para o usuário final, finalizando a requisição.

**- Controllers e Convenções**

Tudo que se precisa fazer para criar um controller do VRaptor é adicionar a anotação @Controller. A partir daí o framework já utiliza suas convenções de URLs e JSPs, exigindo o mínimo de configurações. Veja um exemplo:

@Controller

public class UsuarioController {

public void lista() {

...

}

}

A convenção de URLs do VRaptor é *nomeDoController/nomeDoMetodo*; portanto o método lista() seria mapeado para a URL */usuario/lista*. Repare que o sufixo *Controller* não é considerado na rota.

Outra convenção importante é da página JSP pra a qual o VRaptor vai fazer o dispatch. Ela é bem parecida com a convenção de URLs.O VRaptor procura a JSP no diretório:

WEB-INF/jsp/nomeDoController/nomeDoMetodo.jsp,

que neste caso será:

WEB-INF/jsp/usuario/lista.jsp.

Todos os métodos públicos dos controllers serão mapeados seguindo essas convenções, e passam a atender requests independente do verbo HTTP utilizado.

Pode-se explicitamente declarar qual tipo de requisição cada método vai atender, por exemplo utilizando a anotação @Get. Há ainda as anotações @Post, @Put e @Delete. Se preferir não utilizar a convenção de URLs, você pode passar a URL como *value* dessas anotações de verbo, ou utilizar a anotação @Path se o verbo for indiferente:

@Get("alguma/outra/url")

public void lista(){ ... }

**- Views**

O VRaptor possui grande flexibilidade na escolha de tecnologias da camada de visão, devido ao baixo acoplamento com o controlador proporcionado pelo modelo MVC baseado em ações. É possível utilizar vários templates para geração de páginas dinâmicas, como por exemplo JavaServer Pages (JSP), Velocity, Freemarker, entre outros. Entretanto, os

componentes visuais devem ser desenvolvidos manualmente, utilizando bibliotecas externas como o JQuery UI, Bootstrap, ExtJS, AngularJS e etc.

A camada de visão não precisa ser, necessariamente, gerada dinamicamente no

servidor através de uma ferramenta de template. No caso do cliente ser outra aplicação ou simplesmente uma página HTML estática, a comunicação pode ser feita através da

serialização de objetos em determinado formato, como XML ou JSON. Este tipo de solução é comum ao disponibilizar serviços web para integração entre sistemas.

**- REST**

**- Injeção de Dependências**

**- Interceptadores**

**- Validadores**

**- Plugins**

**Spring MVC e VRaptor**

**- Curva de Aprendizagem**

**- Tamanho da Comunidade**

**- Aceitação do Mercado**

**- Documentação**

1. **Conclusão**

**Considerações Finais**

**Utilizar o VRaptor para criar aplicações web é bastante simples e torna a curva de aprendizado muito pequena. Este é um dos principais fatores que estão levando diversas equipes a utilizá-lo.**

**O que mais atrai no framework é a facilidade para construir Actions fazendo uso das anotações. Outro recurso implementado é o conceito de Convenção ao Invés de Configuração**

**Fornecer uma alternativa mais simples e eficiente para desenvolver aplicações MVC utilizando recursos como: anotações, injeção de dependências e conceitos de "Convenção ao Invés de Configuração".**

**Desenvolvido em território nacional o framework vem ganhando bastante espaço no mercado de aplicações que utilizam o padrão MVC.**

**Referências Bibliográficas**

BERGMAN Noel J, CHOPRA Abhinav. **Introdução às páginas do JavaServer**. Publicado em 28 de Ago. 2001. Disponível em <https://www.ibm.com/developerworks/java/tutorials/j-introjsp/j-introjsp.html>. Acesso em 02 de Set. 2018.

CAELUM, Vraptor 4. **Repositório de download e instruções de instalação da framework**. Disponível em <https://github.com/caelum/vraptor4>. *2018 Github, Inc.* Acesso em 27 de Ago. 2018.

CALVALCANTI Lucas, ***Vraptor*– Desenvolvimento Ágil para Web com Java**, publicado em 16 de Abr. 2014, Ed. Casa do Código.

CHRISTOPHER[Alexander, **An Introduction for Object-Oriented Designers**](http://g.oswego.edu/dl/ca/ca/ca.html). Disponível em <http://g.oswego.edu/dl/ca/ca/ca.html>, Acessado em 07 de Set. 2018

FRANZINI, Fernando, **O que aprendi com livro VRaptor – Desenvovlimento Ágil para Web com *Java***. Publicado em 11 de Dez. 2013. Disponível em<https://imasters.com.br/back-end/o-que-aprendi-com-o-livro-vraptor-desenvolvimento-agil-para-web-com-java>

GUERRA, E., ***Design Patterns* com Java: Projeto orientado a objetos guiado por padrões**. Publicado em 2012. Ed. Casa do Código.

HEMRAJANI, Anil, **Agile JAVA Development with Spring, Hibernate and Eclipse**. Publicado em 2006. Editora Paperback

KAYAL, Dhrubojyoti, **Pro JAVA Spring Patterns – Best Practices and Design Strategies Implementing JAVA EE Patterns with the Sprign Framework**, Publicado em 2008. Editora Apress.

MATTEI, Marcelo. **Boas práticas no desenvolvimento de websites.** Publicado em 19 de Nov. 2007. Disponível em <https://webinsider.com.br/boas-praticas-no-desenvolvimento-de-websites/>. Acesso em 27 de Ago. 2018.

ORACLE, **O que é um *servlet*?** Disponível em <https://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html>. Acesso em 02 de Set. 2018.

REENSKAUG, T., **Models - Views - Controllers**. Disponível em <https://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf>. Acessado em 07 de Set. 2018.

ROTEM GAL OZ, Arnon, ***Soa Patterns***. Publicado em 23 de Set. 2012. Ed. *Manning.*

# SILVEIRA Paulo e Guilherme, Lopes Sergio,Moreira Guilherme, Steppat Nico, Kung Fabio, Introdução À Arquitetura e Design de Software - Uma Visão Sobre a Plataforma Java, publicado em 9 de Dez. 2011,  Ed. *Elsevier Ltda.*

SPRING. **Documentação oficial do framework**. Disponível em <https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html>.*2018 Pivotal Software.* Acesso em 27 de Ago. 2018.

SPRING-PROJECT, Spring Framework.**Repositório de projetos, instruções e downloads referente a framework**. Disponível em <https://github.com/spring-projects/spring-framework>. *2018 Github, Inc*. Acesso em 27 de Ago. 2018.

VRAPTOR. **Documentação oficial do framework**. Disponível em <http://www.vraptor.org/pt/docs>. Acesso em 27 de Ago. 2018.

WEISSMANN Henrique Lobo, **Vire o jogo com *Spring Framework***. Publicado em 16 de Abr. 2014. Ed. Casa do Código.