**Análise comparativa entre dois Frameworks MVC web para plataforma JAVA:**

**Spring MVC e VRaptor**

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

**Pós-Graduação *Lato Sensu***

**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 apresentado os requisitos dos frameworks, o fluxo do processamento da informação,a importância da utilização da arquitetura MVC ea divisão do código em camadas, a utilização da injeção de dependências, a inclusão de validadores e os principais plug-ins para facilitar ainda mais o desenvolvimento. Além disso,serão abordadosos 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*, comparativo,padrão MVC,*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 resumoé 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 padronizaçãono desenvolvimento web com *Java*.

Atualmente, temos a disposição vários *frameworks*para facilitar a produtividade e a padronizaçãodocódigo. 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*frameworksSpring MVC*e o *VRaptor4*.

E arelevância 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 Javainiciou 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 brasileirosPaulo Silveira e Guilherme Silveira,apresenta-se como uma alternativa eficiente ecom a proposta de sua simplicidade e por trazer a ideologia de ser rápido e de fácil aplicação. E desde esse período, houve novasversões e melhorias, acompanhando as melhores práticas de desenvolvimento no mercado e desempenho; atualmente o *framework* encontra-se na versão *4,*trazendototal integraçãocom *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,*trabalhacom 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 maisbenefíciosem 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 *frameworkVRaptor* 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**

**Spring MVC: Requisitos e Fluxo do Processamento da Informação**

O Spring MVC possui uma divisão bem estruturada de suas camadas, tornando o código de uma forma bem organizada.

O framework é compatível com os principais servidores web Java, como o Apache Tomcat, Jboss, BEA Weblogic ou IBM Websphere. Além disso, possui a integração com frameworks para mapeamento do banco de dados como o Hibernate.

Um dos jeitos mais fáceis de se desenvolver um projeto com o spring mvc é utilizando o Spring Tool Suite que pode ser encontrado no site oficial do framework: <https://spring.io/tools>. Essa ferramenta é um eclipse com plugins de desenvolvimento do spring já instalados e configurados, ou seja, nela podemos criar projetos do tipo spring project e no assistente de criação, podemos especificar que queremos que o projeto seja do tipo Spring MVC. Com isso não precisamos nos preocupar em baixar as bibliotecas manualmente ou saber quais são as dependências do maven que precisam ser adicionadas.

Mas se vocês quiserem utilizar em um projeto maven do eclipse, ele pode ser baixado adicionando-se a seguinte dependência dentro do pom.xml:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.1.4.RELEASE</version>

</dependency>

O fluxo do processo das informações do Spring MVC, segue uma sequência de eventos quando uma requisição é enviada ao framework.

1 - Primeiramente o *DispatcherServlet,* recebe a requisição

O *DispatcherServlet* é um dos principais componentes da estruturação do Spring MVC, pois além de ser um mapeador de requisições, representando um único canal de entrada para todas requisições direcionadas, facilitando o gerenciamento da informação, é responsável por encaminhar para qual *Controller* vai receber e processar a requisição, além de apontar o arquivo de *template* específico no qual será renderizado na camada *View*.

2 - O DispatcherServlet verifica o HandlerMapping e carrega o Controller associado a requisição.

HandlerMapping é uma interface que faz a análise e define um mapeamento da requisição.

3 - O Controller processa a requisição através da chamada aos métodos apropriados do serviço e retorna um objeto ModeAndView para DispatcherServlet.

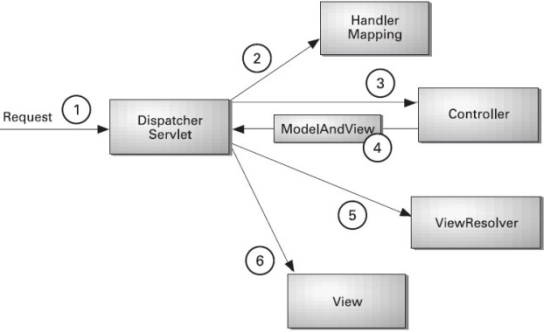
O objeto ModeAndView contém os dados do modelo e o nome da visão.

4 - O DispatcherServlet envia o nome de visão para um ViewResolver para que ele encontre o View que deve ser carregado.

OView Resolver é um gerenciador de visualização, ele procura a página JSP no qual corresponde ao nome da view encaminhada pelo Controller.

5 - Agora o DispatcherServlet passará o objeto modelo para o View para que o resultado seja renderizado.

6 - A View com os dados vindo do modelo, vai renderizar o resultado para o usuário.

****

**Spring MVC: Controllers**

O Controller é uma classe Java que possui os métodos responsáveis por tratar as requisições, e como o próprio nome já diz, faz parte da camada controladora do modelo MVC. Responsável por intermediar as informações junto a View, receber parâmetros e disponibilizar resultados.

O *Controller* suporta os métodos do tipo GET ou POST usado. O método de serviço definirá os dados do modelo com base na lógica de negócios definida e retornará o nome da visualização ao *DispatcherServlet*   
  
Para dizer que a classe será um Controller, basta fazer a anotação **@*Controller*,** assim a classe já passa a servir com a função de um controlador.   
A anotação **@*RequestMapping*** é usada para mapear uma URL para uma classe inteira ou um método de manipulador específico. E para fazer isso, basta passar os atributos como parâmetro de função dentro da anotação.  
O atributo ***value*** indica a URL para a qual o método do manipulador é mapeado e o atributo ***method*** define o método de serviço para manipular a solicitação.

Veja um exemplo simples:

/\*\* Incluir código Spring MVC – C5 \*/

**Spring MVC: Views**

O Spring MVC se destaca por sua separação das tecnologias de visualização.Suporta muitos tipos de visualizações para diferentes tecnologias de apresentação. Estes incluem - JSPs, HTML, PDF, planilhas do Excel, XML, modelos Velocity, XSLT, JSON, Atom e RSS feeds, JasperReports, etc. Mas, mais comumente, são utilizados modelos JSP escritos com JSTL. Veja um exemplo:

//\*\* incluir código Spring MVC – V1 \*/

Aqui **$ {message}** é o atributo que configuramos dentro do Controller. Você pode ter vários atributos para serem exibidos dentro de sua visão.

**Spring MVC: Injeção Dependências**

O Spring MVC foi desenvolvido ao conceito da utilização de Injeção de Dependência, um padrão onde a classe não precisa se preocupar em como conseguir suas dependências, apenas em trabalhar com elas.  
Com isso, ajuda ao desacoplamento do código, tornando mais fácil ao gerenciamento e a realização de testes no sistema.  
E para que o framework identifique os pontos no qual será injetada, é necessário fazer uma anotação na classe com a expressão @Autowired.  
A anotação pode ser utilizada nos em 3 casos:  
- Nas Propriedades;  
- Nos Construtores;   
- Nos Métodos (normalmente, os setters)  
  
Outro requisito para que uma instância possa ser injetada, é transforma-lo em uma Bean Spring.   
No qual é necessário fazer a anotação @Component ou com qualquer uma de suas especializações:  
  
@Component: Esta anotação faz com que o bean registrado no Spring possa ser utilizado em qualquer bean, seja ele um serviço, um DAO, um controller, etc.  
@Service: Anotação no qual diz que a bean faz parte da camada de serviço.   
@Repository: Anotação no qual diz que a bean faz parte da camada de persistência.

//\*\* incluir código Spring MVC – Di1 \*/

**Spring MVC: Validadores**

A validação da entrada recebida do usuário para manter a integridade dos dados é uma parte importante da lógica do aplicativo. A validação de dados pode ocorrer em diferentes camadas.

O Spring MVC no qual é baseado na plataforma JAVA EE 6, aproveita um dos recursos disponíveis para fazer a validação. O modelo Bean Validation, no qual é suportado por restrições na forma de anotações colocadas em um campo, método ou classe de um componente JavaBeans, como um bean gerenciado e pode ser utilizado em qualquer camada da aplicação.

Com o Bean Validation declaramos através de anotações as regras de validação dentro do nosso modelo:

/\*\* Inserir código Spring MVC Validator – V1 \*/

Com a anotação do Bean Validation na camada de controller. É necessário avisar o Spring MVC que queremos executar a validação. Isso é feito pela anotação Valid que devemos usar na antes do parâmetro da ação:

/\*\* Inserir código Spring MVC Validator – V2 \*/

O Spring MVC pode armazenar o resultado dos erros de validação em um objeto do tipo BindingResult. Este objeto BindingResult se torna um parâmetro da ação. No qual pode ser utilizado para que em vez de ser lançado uma exceção, seja redirecionado para outra página.

/\*\* Inserir código Spring MVC Validator – V3 \*/

/\*\* Inserir código Spring MVC Validator – V4 \*/

Para a exibição das mensagens de validação na camada da View, é utilizado uma tag especial que o Spring MVC oferece. A tag se chama **form:errors**:

<form:errorspath="tarefa.descricao" />

O atributo path indica com que atributo essa mensagem está relacionada.

/\*\* Inserir código Spring MVC Validator – V5 \*/

**Spring MVC: Plugins**

O Spring MVC possui diversos módulos disponíveis para acoplamento no seu projeto, para facilitar e agilizar ainda mais no desenvolvimento para serviços específicos.

• Spring Security útil para inclusão de funcionalidades de autenticação e autorização.

• Spring Data para aplicações que usam novas tecnologias de armazenamento de dados e serviços na nuvem.

• Spring Social para fácil integração com redes sociais.

• Além de outros. Disponível no site oficial do framework <http://spring.io/projects>

**VRaptor: Requisitos e Fluxo do Processamento da Informação**

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

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:

VraptorBlank 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.

Os Servidores suportados e já testados pela própria framework, são Wildfly 8, Tomcat 7 e Jetty 8.

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

Assim como o Spring MVC, o VRaptor é baseado o seu fluxo de processamento da informação em Action Based, que recebe diretamente as requisições HTTP. Tornando o modelo flexível, deixando a livre opção de escolha do tipo de view para gerar uma requisição HTTP compatível.

O VRaptor possui tem o benefício de encapsular as principais classes de Servlets, por exempo as classes HttpServletRequest, HttpServletResponse e Session, obtendo assim o ganho de poder tratar a regra de negócio por Controllers.

O VRaptor foca em simplicidade, baseado em **convenção sobre configuração,** com intuito de facilitar a padronização do código e evita as diversas configurações em arquivos XML vistas em outros frameworks.

- Quando uma requisição chega, ela é prontamente atendida pelo VRaptor.

- O framework então através da anotação do Controller, 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.

// incluir imagem

**Segue algumas das anotações utilizadas para o gerenciamento da informação do VRaptor:**

@Controller – Anotação responsável por identificar os elementos que estarão disponíveis como controladores de requisição

@Component – Anotação para indicar que aquela classe usará o padrão de Injeção de Dependências para executar tarefas.

@Path – Anotação utilizado para informar a URL que será atendido a requisição

**VRaptor: Controllers**

Para criar o Controller no VRaptor, que são classes que vão executar as tarefas de requisição, basta apenas adicionar a anotação @Controller na classe desejada, para que seus métodos publico estejam disponíveis. A partir daí o framework necessita que sejam seguidas algumas regras de convenções de URLs e JSPs, para funcionar corretamente.

As classes controladoras devem possuir o nome com a terminação *Controller*, e anotadas com @*Controller*.

O nome do arquivo JSP precisa ser igual o nome do método no controlador.

A URL de acesso deve seguir o seguinte formato: domínio/contexto/controlador/método

Exemplo: localhost:8080/meuProjeto/produto/cadastrar

/\*\* Inserir código VRaptorController– C1 \*/

Seguindo corretamente essas convenções, todos os métodos públicos dos *controllers* serão mapeados, passando a executar as requisições.

Outra vantagem que o VRaptor utiliza do conceito de anotações em classes, é a possibilidade de declarar o tipo de requisição que será executado. Bastando apenas, assim no *controller*, anotar a classe com os seguintes tipos:

@Post – Anotação para dizer que o método atendera apenas requisição do tipo Post

@Get – Anotação para dizer que o método atendera apenas requisição do tipo Get

@Put – Anotação utilizado para realizar alterações em parte ou no objeto inteira

@Delete - Anotação utilizado para realizar a exclusão de algum objeto ou elemento

/\*\* Inserir código VRaptorController– C2 \*/

**VRaptor: Views**

O VRaptor tem a característica de possuir baixo acoplamento da camada visão com seu controlador, devido ao modelo baseado no MVC de ações. Ele torna-se flexível a escolha da tecnologia da visualização da interface.

Pode ser utilizado diversos templates, por exemplo: JSP, Velocity ou Freemaker para criação de paginas dinâmicas, entretanto as ações e os elementos visuais ainda sim, devem ser criados manualmente ou utilizando bibliotecas externas como por exemplo: Bootstrap, JQuery, AngularJS, etc.

A convenção padrão utilizada na view, é no qual os arquivos JSP devem estar dentro da pasta /WEB-INF/jsp com o nome referente ao controlador correspondente, excluindo a terminação Controller e seguindo o estilo lowerCamelCase,

O VRaptor disponibiliza um objeto para trabalhar com alguns recursos relacionados a View, este objeto chamado de Result, pode ser injetado através do construtor

- O Result pode redirecionar o fluxo para outra lógica de outro controlador

Result.redirect()

- O Result pode modificar a *view* padrão, retornando JSON, XML, Status HTTP, ao invés de JSP. Este tipo de solução é comum ao disponibilizar serviços web para integração entre sistemas.

Result.use(Results.json()).serialize();

- O Result pode adicionar objetos no request, tornando-os acessíveis na JSP:

Result.include(“mensagem”, “Senha alterado com sucesso”)

- O Result pode redirecionar o fluxo caso ocorra uma Exception:

Result.on(GenericAccessDeniedException.class).redirectTo(AccessDeniedController.class).principal();

- Conversão automática de tipos;

Para registrar objetos a serem acessados na view, usamos o método include:

/\*\* Inserir código VRaptorViews– V1 \*/

/\*\* Inserir código VRaptorViews– V2 \*/

/\*\* Inserir código VRaptorViews– V3 \*/

**VRaptor: Injeção de Dependências**

O VRaptor se beneficia de todas funcionalidades e as boas práticas fornecidas pelo CDI do Java EE 7, pois todos os componentes, que são instancias de classes necessários para execução de tarefas, o ciclo de vida de seus componentes e a possibilidade de armazenar o estado de interação do usuário, são todos gerenciados através dele.

Isso torna o framework mais desacoplado e extensível, além da integração com recursos nativos do servidor de aplicação com as demais especificações contidas na plataforma.

Para implementação, é necessário declarar um construtor padrão para que o CDI possa gerenciar a classe, ao componente gerenciável fazer a anotação @Named e para identificar o ponto no qual será injetavel com a anotação @Inject.

/\*\* Inserir código VRaptor DI – D1 \*/

Um exemplo do VRaptor gerenciando o ciclo de um componente através do escopo:

/\*\* Inserir código VRaptor DI – D2 \*/

**VRaptor - Validadores**

VRaptor utiliza como principal método de validação a especificação do BeanValidation, fornecidos também pelo Java EE 7, através dele podemos validar todos os modelos baseado em anotações.

Entretanto, pode ser utilizado os métodos de validação do próprio framework, através do método add() para retorno de uma mensagem simples ou internacionalizada, e o método addIf() para exibir a mensagem na condição do método ser verdadeiro ou ensure() para exibição d mensagem sob a condição falsa, todos métodos pertencentes da classe Validator do VRaptor.

Exemplo de validação com validator do VRaptor:

Exemplo da validação com BeanValidation com regra de redirecionamento no caso de uma restrição de validação.

/\*\* Inserir código VRaptor Validador – V1 \*/

**VRaptor: Plugins**

O VRaptor posssui diversos plugins disponibilizados em seu site oficial, no qual muitos foram criados pela Caelum ou pela própria comunidade, pois muitos desses plugins foram criados através de terceiros, por desenvolvedores que possui um nível mais avançado e reconhecimento na comunidade, pois o framework tem essa facilidade de criar componentes reusáveis e de fácil aplicação, com intuito de sempre tentar resolver um problema em comum.

Para utilizar esses plugins, é necessário apenas adicionar o arquivo jar no seu projeto, pois enquanto o plugin possuir o arquivo bean.xml, o CDI vai fazer o gerenciamento e a disponibilização das classes a serem injetadas.

Segue alguns plugins que já possuem uma versão compatível com versão do framework:  
[vraptor-time-converters](https://github.com/caelum/vraptor-time-converters) – para converter com data e hora   
[vraptor-simplemail](https://github.com/caelum/vraptor-simplemail) –plugin para facilitar o envio de emails  
[vraptor-quartzjob](https://github.com/caelum/vraptor-quartzjob) – realizar agendamento de tarefas  
[vraptor-jpa](https://github.com/caelum/vraptor-jpa) e vraptor-hibernate – produtores e controle de transação

**Spring MVC vs. VRaptor**

**- Curva de Aprendizagem**

**- Comunidade e Aceitação do Mercado**

**- Documentação**

1. **Conclusão**

**Considerações Finais**

**Levar em consideração os seguintes itens, para a escolha de uma framework:**

**Popularidade**

**Filosofia**

**Sustentabilidade**

**Técnica**

**Segurança**

**Documentação**

**Licença**

**Recurso no Mercado**

**Experimentar, Teste**

**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.**

**O Spring Web MVC Framework é uma estrutura robusta, flexível e bem projetada para**

**aplicativos da web em rápido desenvolvimento usando o padrão de design MVC.**

**alcançado usando este módulo Spring são semelhantes àqueles que você obtém do resto do**

**Spring Framework. Vamos rever alguns deles. Eu vou demonstrar alguns desses benefícios**

**mais adiante neste capítulo.**

**Teste mais fácil - Esse é um tema comum que você encontrará em todas as classes do Spring.**

**O fato de que a maioria das classes do Spring é projetada como JavaBeans permite que você**

**injetar dados de teste usando os métodos setter dessas classes. Primavera também fornece simulado**

**classes para simular objetos Java HTTP (HttpServletRequest, por exemplo), que**

**torna o teste de unidade da camada da Web muito mais simples.**

**Vincule-se diretamente a objetos de negócios - o Spring MVC não exige sua empresa**

**classes (model) para estender quaisquer classes especiais; isso permite que você reutilize seus negócios**

**objetos, vinculando-os diretamente aos campos de formulários HTML. Na verdade, o seu**

**Conceitos do Spring Web MVC 129**

**classes do controlador são as únicas que são necessárias para estender as classes do Spring (ou**

**implementar uma interface do controlador Spring).**

**Clara separação de papéis - o Spring MVC separa muito bem os papéis desempenhados pelo**

**vários componentes que compõem este framework web. Por exemplo, quando discutimos**

**conceitos como controladores, objetos de comando e validadores, você começará**

**para ver como cada componente desempenha um papel distinto.**

**Controladores adaptáveis ​​- se o seu aplicativo não exigir um formulário HTML, você**

**pode escrever uma versão mais simples de um controlador de mola que precisa de todo o extra**

**componentes necessários para controladores de formulário. De fato, a Spring fornece vários tipos de**

**controladores, cada um servindo a um propósito diferente. Por exemplo, existem controladores sem forma,**

**Controladores de formulários simples, controladores de formulários com aparência de mago, visualizações sem controladores,**

**e até mesmo controladores pré-empacotados que permitem escrever visualizações sem**

**seu próprio controlador personalizado.**

**Biblioteca de tags simples, mas eficiente: a biblioteca de tags da Spring é pequena, direta, mas**

**poderoso. Por exemplo, Spring usa a linguagem de expressão JSP (EL) para argumentos**

**para a tag <spring: bind>.**

**Fluxo da Web - Este módulo é um subprojeto e não é fornecido com o núcleo Spring**

**distribuição. Ele é construído sobre o Spring MVC e adiciona a capacidade de**

**escrever aplicativos da web em forma de mago que abrangem várias solicitações HTTP (uma**

**carrinho de compras online, por exemplo).**

**Visualize tecnologias e estruturas da Web - embora estejamos usando a JSP como nossa visão**

**tecnologia, o Spring também suporta outras tecnologias de visualização, como o Apache**

**Velocity (jakarta.apache.org/velocity/) e FreeMarker (freemarker.org). Este é um**

**conceito poderoso porque mudar de JSP para Velocity é uma questão de configuração.**

**Além disso, o Spring fornece suporte de integração para o Apache Struts (struts.**

**apache.org), Apache Tapestry (jakarta.apache.org/tapestry) e OpenSymphony’s**

**WebWork (opensymphony.com/webwork/).**

**Ambiente mais leve - Como mencionei no capítulo anterior, Primavera**

**permite criar aplicativos prontos para empresas usando POJOs; o ambiente**

**a configuração pode ser mais simples e menos dispendiosa porque você pode desenvolver e implantar**

**seu aplicativo usando um contêiner de servlet mais leve.**

Spring MVC é um dos módulos que compõem o Spring Framework utilizado para construir aplicações web. Além de um Framework ele conta com as boas práticas de projeto para desenvolvimento de software web utilizando a plataforma Java EE.

O padrão MVC de projetos web tem como base romper a comunicação direta das camadas de visualização com as camadas de modelo ou negócios. Essas duas camadas permanecem isoladas uma da outra, se comunicando apenas por uma camada intermediária, chamada de controle. Dessa maneira, o acoplamento diminui consideravelmente, melhorando a qualidade e a manutenção da aplicação.

O Spring MVC ajuda a construir aplicações web flexíveis e com baixo acoplamento. O padrão de design Modelo-visão-controlador ajuda na separação da lógica de negócio, lógica de apresentação e lógica de navegação. Os Modelos são responsáveis por encapsular os dados da aplicação. As Visões renderizam as repostas ao usuário com a ajuda do objeto modelo. Os Controladores são responsáveis por receber as requisições do usuário e executar os serviços.

Todo mundo que utiliza algum [framework MVC](http://pt.wikipedia.org/wiki/MVC) moderno já se deparou em algum momento com o conceito de [Inversão de Controle](http://pt.wikipedia.org/wiki/Invers%C3%A3o_de_controle) (IoC). Com esse conceito, a classe não mais se preocupa em como conseguir suas dependências, mas sim em apenas trabalhar com elas. O principal jeito de conseguir isso é através da [Injeção de Dependências](http://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_depend%C3%AAncia) (DI). Essa técnica, presente em diversos frameworks MVC, ajuda muito no desacoplamento e nos [testes do nosso sistema](http://blog.caelum.com.br/facilitando-a-manutencao-dos-testes-ao-diminuir-o-acoplamento-com-o-codigo/).

Dois dos frameworks onde encontramos isso são o [Spring MVC](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html) e o [VRaptor](http://www.vraptor.org/).  
No exemplo abaixo, temos um Controller em cada um desses Frameworks. Esse Controller depende de um DAO, que por sua vez tem como dependência uma Connection.

**DI manter o baixo acoplamento entre as classes** de um mesmo projeto.

Com certeza, essa característica é uma **grande vantagem** para a arquitetura do seu sistema. Sendo assim, esse é um conceito que merece a atenção da sua parte.

**Vantagens VRaptor**

Alta produtividade,

Baixa curva de aprendizagem

Testabilidade

Economia de tempo no projeto e consequentemente dinheiro

Flexibilidade

Documentação em português

**Desvantagens**

Não possui componentes próprios como o JSF

Maior dependência do HTML, CSS e Javascript

Pouco utilizado em nosso mercado

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.

Spring

* Conhecido principalmente pelo seu mecanismo de injeção de dependências.
* Baseado em ações.
* Total controle do que acontece no front-end
* Facilidade de integração com bibliotecas no front-end
* Comunidade muito ativa
* Existe uma gama de componentes na família spring SData, SSecurity, SAOP, entre outros.
* Configuração muito chata, até pouco tempo atrás era baseada em xml. Atualmente muitas coisas, se não tudo, pode ser feito via Annotations.
* Por ter muito pontos de extensão, ao meu ver, a curva de aprendizado é um pouco maior do que o JSF.

A simplicidade de configurar, pouca intrusão do framework, diminuição do acoplamento ao utilizá-lo, a modularização dos projetos, a ótima integração entre os projetos spring e a fácil customização do framework. É relevante utilizar nos projetos novos por ser confiável, leve e consolidado. Alem de possuir o Spring Boot, que aumenta muito a produtividade do desenvolvedor e diminui a curva de aprendizado do ambiente Spring.

O que me levou a adotar o spring foi a grande quantidade de documentação e ajuda disponível. Há muitos bons livros e o framework é amplamente usado e divulgado.   
Creio que a relevância do spring está em duas coisas: Na sua comunidade e na forma como

o framework torna mais simples conceitos complexos, como injeção de dependências e programação orientada a aspectos.

Há uma grande variedade de frameworks para o desenvolvimento Web em Java, o que torna muito difícil a sua avaliação. O levantamento de critérios auxilia a escolha de um framework para uma determinada situação, pois permite a tabulação das características de cada artefato estudado, facilitando assim a análise.

É difícil levantar critérios objetivos na comparação de tecnologias. Critérios como velocidade de desenvolvimento ou linhas de código necessárias para desenvolver uma aplicação não seriam avaliadas adequadamente apenas com a construção de protótipos.

O estabelecimento de critérios, embora subjetivos, deve auxiliar futuras avaliações de frameworks, permitindo que o analista investigue diretamente a classificação do artefato nos critérios pré-estabelecidos.

A elaboração de um quadro comparativo facilita as análises por sintetizar boa parte do estudo elaborado. Ele pode ser facilmente atualizado, no caso de uma nova versão do framework, ou expandido, no caso da avaliação de um novo framework.

No caso dos frameworks avaliado, não é possível apontar qual seria o ideal para qualquer situação. No entanto, o estudo feito deve auxiliar na analise numa situação específica, uma vez que os dados estão tabulados e seu embasamento está contido no trabalho.

A arquitetura Spring MVC, embora simples, possui muitas camadas e abstrações que podem ser difíceis de depurar se surgirem problemas. Também é altamente dependente do núcleo da primavera. É uma estrutura antiga e madura que possui inúmeras maneiras de estendê-la e configurá-la - e isso, na verdade, torna-a bastante complexa.

### Spring

### A atualização e a manutenção do código podem ser uma tarefa difícil se você for iniciante

A atualização do seu projeto é gerenciável se você já estiver familiarizado com o framework e o projeto em si, mas se você está mergulhando nele, pode ser um pouco complicado e difícil.

### Falta no desenvolvimento da interface do usuário

Embora realmente muito bom e com um conjunto de recursos muito completo e rico para desenvolver e manter código no lado do servidor, ele ainda não fornece nenhuma estrutura rica para construir boas

VRaptor

, já que poderia ter dificuldades e exigir o entendimento de linguagens como CSS, HTML e JavaScript. A documentação pode ser melhor e mais completa com uma série de exemplos e oportunidades para aplicar esse framework. Uma desvantagem do framework não é compatível com a especificação do JSF

Se você precisa de uma estrutura fácil e funcional para criar programas de rede, o vraptor é o caminho certo. De e-commerces a aplicações de grande escala. Sem dúvida, é fácil de usar e pode ser muito divertido criar um utilitário com este framework. Eu estou fazendo uma aplicação inteligente para usar em educação a distância. Aplicativos da Web e móveis criaram rapidamente o uso do Raptor.

Vraptor

Embora existam muitos bons recursos e idéias, a comunidade nunca realmente construiu sobre ela de uma forma que se poderia realmente confiar. Como eu era um usuário feroz da terceira versão e comecei outro projeto na quarta versão, fiquei completamente desapontado com as grandes falhas e falta de interesse de outras partes

 É muito fácil de usar e tem uma boa documentação, mas carece de colaboradores mais ativos para melhorar e consertar as coisas

Spring

Constantemente sendo atualizado com novas funcionalidades e extremamente fácil de usar. Eles são definitivamente uma ótima ferramenta se você quiser desviar um pouco de outros produtos.

Curva de aprendizado íngreme, mas quando você trabalha com o produto algumas vezes, é muito fácil se adaptar e melhorar.

A curva de aprendizado pode ser difícil, pois requer muito conhecimento ao customizar componentes.

Documentação, problemas que resolvem, fácil de adotar, estrutura flexível e completa.

A primavera é complicada porque tenta fazer muitas coisas ao mesmo tempo. Ele passou por muitas revisões, tem muitas dependências externas e possui uma sintaxe muito detalhada.

A segurança pode ser um desafio para a configuração porque é altamente configurável ... tantas peças podem ser sobrescritas. Pode ser difícil descobrir o que precisa ser "ligado" ou configurado para obter a configuração específica de que você precisa.

O Spring é simples de usar, fácil de configurar e altamente configurável. Injeção torna o acesso a outros objetos / referências super fácil. A maioria dos componentes é substituível e fácil de configurar. A instalação de novos projetos é rápida e indolor. O Spring funciona bem com ORMs como o Hibernate. Eu não usaria outro IDE / framework para desenvolvimento de API. A implantação é fácil e rápida.

Spring

 Spring funciona como um todo, que é sua maior advertência. Há muitos tutoriais on-line e páginas de documentação, bem como o aplicativo de amostra Petclinic que você pode despir e usar como base, mas, no geral, o Spring é para construir aplicativos sérios com fundamentos sólidos, interfaces de usuário avançadas e uma API RESTful. o que torna completamente desnecessário para qualquer coisa significativamente mais simples que isso.

O SpringMVC, e junto com o Spring, está constantemente mudando e melhorando. A questão é que seus desenvolvedores terão que acompanhar constantemente a tecnologia para melhorar o aplicativo à medida que o Java cresce, os navegadores da Web mudam e outras melhorias acontecem no espaço da Web.

**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.