Aline Cristina Pinto
Eduardo Cotta Guimarães França
Luiz Gustavo de Matos Paiva
Pedro Lucas Silva
Ruan Bertuce de Carvalho Souza

Spring MVC

Belo Horizonte

Agosto de 2018

Aline Cristina Pinto
Eduardo Cotta Guimarães França
Luiz Gustavo de Matos Paiva
Pedro Lucas Silva
Ruan Bertuce de Carvalho Souza

Spring MVC

Trabalho acadêmico da disciplina *Linguagem de Programação II* do curso técnico de Informática do Centro Federal de Educação Tecnológica de Minas Gerais.

Belo Horizonte
Agosto de 2018

Talk is cheap. Show me the code.
Linus Torvalds

Resumo

O padrão arquitetural MVC (Model-View-Controller) é um dos mais antigos e mais utilizados atualmente. Neste sentido, o uso de frameworks para auxiliar e respeitar o emprego deste padrão nas aplicações se faz conveniente, visto que eles proveem vários benefícios, como redução nas chances de uma erosão arquitetural e aumento da velocidade e da facilidade no desenvolvimento de uma aplicação. Relativo a isso, este trabalho, primeiramente, busca analisar as peculiaridades e o funcionamento do padrão MVC. Após isso, é realizado um estudo sobre os frameworks MVC e um discernimento sobre seus modelos atuais, utilizando como objeto alguns exemplos de frameworks, com um foco especial no Spring MVC.

Palavras-chave: MVC. Framework. Spring. Baseado em Componentes. Baseado em Ações. Arquitetura de Software. Java Web.

Abstract

The architectural pattern MVC (Model-View-Controller) is one of the oldest and most used nowadays. In that way, the frameworks usage for helping and respecting the use of this pattern in the applications is convenient, since they provide various benefits, such as a reduction in the chances of an architectural erosion and an increase of speed and ease in an application development. Concerning to that, this work, firstly, seeks to analyze the MVC pattern peculiarities and operation. After that, is realized a study about the MVC frameworks and a discernment about its current models, using frameworks examples as object, with a special focus at Spring MVC.

Keywords: MVC. Framework. Spring. Component-Based. Action-Based. Software Architecture. Java Web.

Sumário

1 Introdução

A partir do gigantesco crescimento e popularização do desenvolvimento de sistemas de informação, fez-se necessária a instituição de padrões com o objetivo de guiar o desenvolvimento a um patamar mais sólido, profissional e com foco na qualidade do produto de software. Um desses padrões é o padrão Model-View-Controller, um padrão arquitetural que facilita a interoperabilidade das camadas e sua modularidade.

Tendo em vista uma melhora na produtividade e a utilização de código já produzido, o uso de frameworks que implementam o padrão MVC tornou-se popular, principalmente com o foco nas aplicações WEB, existindo diversos exemplos desses em utilização na área de desenvolvimento de software.

Neste trabalho busca-se apresentar uma visão geral sobre o padrão MVC, o que são frameworks MVC e exemplos importantes como JavaServerFaces, Struts, Play e principalmente Spring MVC, que é o foco do trabalho. Aborda-se as principais características do Spring MVC, assim como seu fluxo de funcionamento, principais componentes e maiores diferenciais em relação a outros frameworks, de modo a apresentá-lo brevemente.

2 Spring MVC

2.1 Histórico

A evolução das aplicações fez necessária a criação de padrões para a implementação facilitada de algumas funcionalidades. Nesse Contexto, em 1979, o cientista norueguês Trygve Reenskaug iniciou o que hoje é conhecido como padrão de projeto MVC, originalmente descrito em seu artigo "Applications Programming in Smalltalk-80: How to use Model-View-Controller" desenvolvido no período em que Reenskaug trabalhava no Smalltalk, na Xerox PARC.

A ideia de Reenskaug gerou um padrão de arquitetura de aplicação baseado na divisão do projeto em três camadas bem definidas e independentes que ajudam na redução de acoplamento e promovem o aumento de coesão nas classes do projeto, facilitando a manutenção do código e sua reutilização em outros projetos.

2.2 O que é o Padrão Model-View-Controller (MVC)

O padrão de arquitetura MVC tem como objetivo separar a representação dos dados em três camadas muito bem definidas: Modelo (Model), Visão (View) e Controlador (Controller), sendo que cada uma executa apenas o que lhe é definido.

A utilização desse padrão permite isolar as regras de negócios da lógica de apresentação, a interface com o usuário. Isto possibilita a existência de várias interfaces que podem ser modificadas sem que haja a necessidade da alteração das regras de negócios, proporcionando assim muito mais flexibilidade e oportunidades de reuso das classes. Além disso, é possível utilizá-lo em diversos projetos como desktop, web e mobile.

2.2.1 Camadas

Models: Pode ser definida pela camada responsável pela leitura e escrita de dados, além de suas validações. Segundo Reenskaug, em ***MODELS-VIEWS-CONTROLLERS*** (REENSKAUG, 1979, p. 1): "Models represent knowledge. A model could be a single object (rather uninteresting), or it could be some structure of objects. [...] The nodes of a model should therefore represent an identifiable part of the problem."

Views: Pode ser definida pela camada responsável pela exibição dos dados para o usuário. Segundo Reenskaug, em ***MODELS-VIEWS-CONTROLLERS*** (REENSKAUG, 1979, p. 1): "A view is a (visual) representation of its model. It would ordinarily

highlight certain attributes of the model and suppress others. It is thus acting as a presentation filter."

Controllers: Pode ser definida pela camada responsável por receber todas as requisições do usuário e através de seus métodos, chamados actions, controlar qual model usar e qual view mostrar ao usuário. Segundo Reenskaug, em ***MODELS-VIEWS-CONTROLLERS*** (REENSKAUG, 1979, p. 1): "A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen."

2.2.2 Compreendendo o funcionamento do MVC na prática

De forma geral, o funcionamento do MVC pode ser compreendido da seguinte forma: O usuário interage com a interface, que é exibida pela visão (view), mas controlada pelo controlador (Controller) que interpreta as entradas mapeando-as em comandos que são enviados para o modelo (Model) e/ou para a janela de visualização (View). Quando o modelo (Model) recebe os dados do controlador, ele os gerencia respondendo a perguntas sobre o seu estado e respondendo a instruções para que ocorra as mudanças de estado. Por fim, a visão usa o modelo para gerar uma interface com o usuário apropriada. A visão recebe as informações do modelo. O modelo não tem conhecimento direto da visão. Ele apenas responde a requisições por informações de quem quer que seja e requisita por transformações nas informações feitas pelo controlador. Após isso, o controlador, como um gerenciador da interface do usuário, aguarda por mais interações do usuário, onde inicia novamente todo o ciclo.

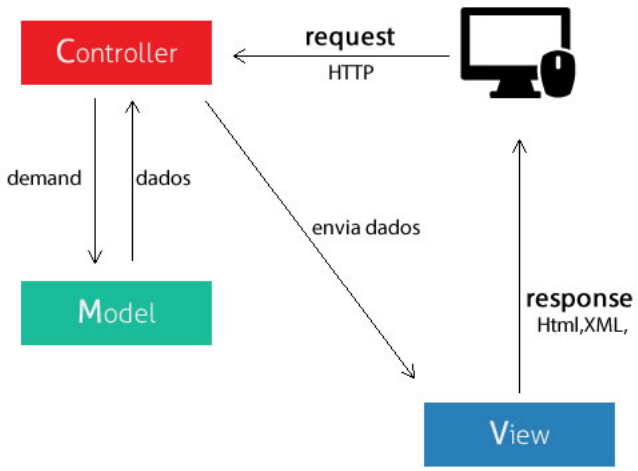

Figura 1 – Exemplificação do funcionamento do Padrão MVC

2.3 Frameworks MVC

2.3.1 Conceito

Tendo ciência do que é o padrão MVC, para entender o que é um framework desse modelo resta saber dois conceitos: domínio de aplicação e framework. Domínio de aplicação são partes de diferentes aplicações que se determinam a fazer algo em comum. Já framework é um conjunto de soluções em código que tem o propósito de resolver problemas específicos de um domínio de aplicação, os quais são por ser desta maneira recorrentes e assim ser reutilizável. Utilizando esses conceitos, pode-se definir que framework MVC é um framework que resolve problemas específicos do MVC, o qual pode ser considerado um domínio de aplicação.

2.3.2 Modelos

Existem dois: um baseado em componentes (Component-Based) e outro baseado em ações (Action-Based). O baseado em componentes busca trazer uma maior abstração no desenvolvimento WEB permitindo trabalhar através de componentes visuais de alto nível no que se refere ao seu objetivo. Ele inicia seu processamento através da camada de visão, a qual obtêm os dados do controlador e solicita os dados ao Managed Bean, que através das suas classes busca os objetos solicitados invocando a execução as regras de negócio das camadas de modelo. Os dados recuperados pelo Managed Bean são passados para a camada de visão que devolve a resposta para o usuário. Já o baseado em ações busca trazer um maior grau de liberdade para otimizar da aplicação. Esse modelo proporciona um melhor aproveitamento da Web e outras ferramentas que trabalham com o protocolo HTTP e garante disponibilidade e escalabilidade de uma forma mais fácil. Nele o processamento começa sempre da camada de controle, que invoca o modelo para a execução das regras de negócio e devolve a resposta para a camada de visão a qual por sua vez devolve a resposta para o usuário.

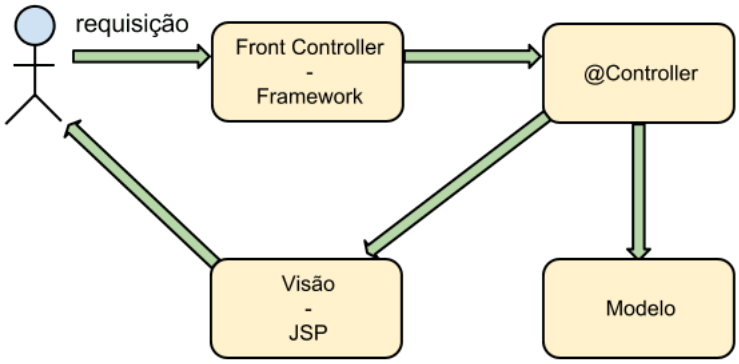

Figura 2 – Funcionamento de um framework MVC Action Based

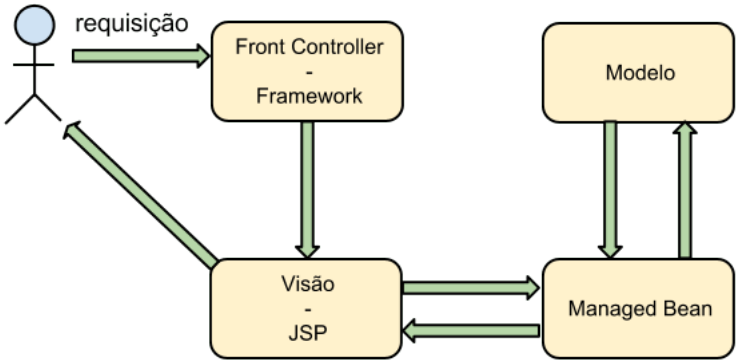

Figura 3 – Framework MVC Component Based

2.3.3 Vantagens de utilização

Como todo framework, as vantagens de utilizar frameworks MVC se encontram nas facilidades que eles trazem durante a implementação da aplicação. Algumas delas são: aproveitamento de código implementado, redução de custos de implementação e estrutura bem definida e comum a várias aplicações.

2.4 Exemplos de Frameworks MVC

2.4.1 JavaServer Faces

O JavaServer Faces, também conhecido como JSF, é um framework MVC baseado em componentes (Component-based) que faz parte da Java EE. A proposta do JSF é abstrair a complexidade relativa a alguns recursos web, como HTML, CSS, Javascript e até mesmo o próprio protocolo HTTP (CAELUM, 2014), auxiliando na construção de aplicações Java para web pois fornece uma maior separação entre a interface e a lógica.

Para tal, o JSF disponibiliza ao programador uma série de ferramentas e componentes localizados em bibliotecas de tags customizadas (taglibs). Estes componentes vão desde caixas de entrada de texto simples à tabelas dinâmicas que conseguem, a partir de um código de visão limpo de códigos lógicos, chamar métodos da lógica da aplicação. Os métodos chamados pelos componentes visuais do JSF se localizam nos Managed Beans, que são classes POJO, e eles também possuem um código limpo, recebendo e retornando elementos Java comuns (como Strings, Lists, entre outros), e não elementos HTML.

Esta tecnologia possui vários pontos positivos, entre eles: alta popularidade e uso no mercado por ser um padrão definido pelo JCP, rapidez na construção das aplicações provida pela grande abstração dos elementos web, componentes que podem ser reutilizados ao longo do desenvolvimento da aplicação e código mais legível em relação aos códigos Java misturados com HTML no JSP. Por outro lado, a curva de aprendizagem do JSF é

tida como um de seus principais pontos negativos, além da grande opacidade na execução, que gera dificuldades na realização de debugs e otimizações.

2.4.2 Struts

O Framework Struts é baseado em ações (Action Based), então, diferente do JSF, possui um acoplamento bem menor com a visão. Este framework ajuda na separação da aplicação entre as três camadas do MVC, mantendo o controle de fluxo de decisões fora da camada de apresentação. Já pelo nome é perceptível seu propósito: fornecer toda uma estrutura inicial para a aplicação que o utiliza.

O Struts possui vários elementos, como taglibs, Controller Components e arquivos de configuração para conectar as camadas da aplicação. As tags do Struts buscam facilitar a referência a dados dinâmicos de uma página, controlando seu fluxo e mostrando os dados. (APACHE, 2018). Os Controller Components são um conjunto de componentes programáveis que permitem aos desenvolvedores definirem exatamente como a aplicação deverá interagir com o usuário, além de simplificar o uso de Servlets.

A partir desta visão é possível perceber que o Struts consegue prover uma organização central na construção de aplicações baseadas em MVC, separando claramente a lógica e a apresentação, além de permitir uma fácil integração com componentes externos. Contudo, esta tecnologia é pouco flexível, impondo a seus usuários um formato de se pensar e codificar. Além disso a curva de aprendizagem do Struts também é elevada. (CHAUBEY, 2012).

2.4.3 Play Framework

O Play é um framework de código livre escrito na linguagem Scala e também utilizável em outras linguagens como Java. Esta tecnologia fornece uma convenção que, ao ser seguida, poupa o desenvolvedor de criar configurações adicionais. "Play is a modern web framework that combines the performance and reliability of Java and Scala, the power of reactive programming, and the productivity of full hot reload support". (BRIKMAN, 2013).

A performance do Play deve-se muito ao Netty, seu servidor integrado, e ao Akka. Esta integração garante um melhor gerenciamento de threads ao utilizar a técnica de “Non-blocking I/O”, melhorando o desempenho e o design da aplicação. O Netty e outros componentes facilitam o desenvolvimento do projeto, visto que as configurações relacionadas a servidores HTTP serão reduzidas, transformando o uso deste protocolo em algo "simples, flexível e poderoso". (BOAGLIO, 2014). O tratamento de erros também é positivo no Play, que garante uma integração com outros frameworks de testes.

Esta tecnologia não é tão utilizada quanto o JSF ou Spring MVC, mas é empregada

por várias empresas grandes, como LinkedIn, Electronic Arts e Walmart. Possui uma comunidade bastante ativa, e, por possuir código livre, o tratamentos de bugs do próprio framework é maior, garantindo sua confiabilidade.

2.4.4 Spring MVC

O Spring é um framework que começou a ser desenvolvido em 2002, e seu propósito inicial não era o de dar suporte ao desenvolvimento web. Com o tempo, a comunidade do Spring começou a criar um framework MVC próprio utilizando recursos modernos, e o resultado foi o Spring MVC. Durante sua expansão, o Spring adotou inúmeras mudanças para se tornar uma estrutura Java em larga escala para aplicações web, e seu conjunto de ferramentas, componentes modernos e flexibilidade o tornou um dos mais famosos frameworks MVC da atualidade.

2.5 Spring MVC

2.5.1 O que é o Spring MVC

Spring MVC é um framework baseado em ações utilizado para o desenvolvimento de aplicações WEB. Ele faz parte do conjunto de frameworks Spring e traz diversas funcionalidades e características que o diferenciam de outros frameworks e facilitam o desenvolvimento de JAVA para WEB. Nele as ações são definidas dentro dos controllers, específicos para cada requisição, e através delas se define as funções que o sistema deverá exercer.

Em resumo, o fluxo de uma requisição dentro do funcionamento do framework pode ser descrito como:

1. Acessa-se uma URL que envia uma requisição HTTP para o servidor da aplicação WEB,

2. O controlador principal do framework, chamado de DispatcherServlet, recebe a requisição e procura qual classe controller é responsável para o tratamento daquela requisição, disponibilizando os dados enviados na requisição.

3. O controller específico envia esses dados ao model, que executa as regras de negócio, cálculos, validações, acesso ao banco de dados, entre outras funções.

4. O resultado das operações é retornado ao controller.

5. O controller indica o nome da view a ser invocada, juntamente com um objeto que guarda os dados que ela deve apresentar.

6. O framework encaminha a view, que processa e apresenta os dados, transformando-se em um HTML.

7. O HTML gerado é retornado ao usuário.

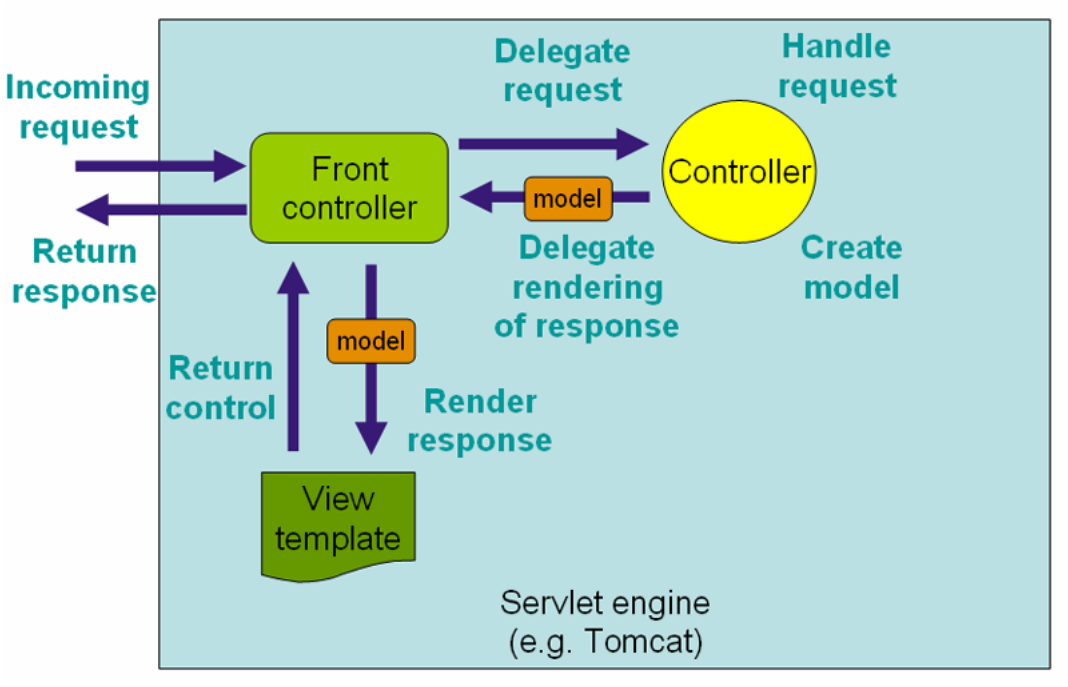

Figura 4 – Fluxo de uma requisição no Spring MVC

2.5.2 Características

Existem algumas características importantes que definem o funcionamento e estrutura do Spring MVC. Dentre elas:

Inversão de Controle (IoC): Pode ser descrito como um padrão de projeto e uma série de técnicas de programação em que o fluxo de controle do sistema é invertido em comparação com o modo tradicional de interação. (GUPTA; GOVIL, 2010). Essa característica é presente nos frameworks em geral, onde o framework dita como o fluxo da execução irá funcionar. Especificamente no Spring MVC, esse conceito funciona juntamente com o CDI (Context and Dependency Injection). Através da anotação,por exemplo, @Autowired, sobre um construtor de um objeto, indica-se que o framework deve resolver e injetar aquela dependência. Em alguns casos, com uso de outros recursos como o spring-context.xml, pode-se criar uma cadeia de dependências onde o framework é totalmente responsável por sua criação (IoC), como visto em ***Apostila de Java para desenvolvimento WEB*** (CAELUM, 2012)

CDI (Context and Dependency Injection): CDI é a especificação do Java EE que cuida da parte de injeção de dependências, mais especificamente a JSR-299. O Spring só dá suporte limitado ao CDI mas aproveita as anotações padronizadas pela especificação JSR-330. (CAELUM, 2012). Logo, podemos substituir algumas anotações como o exemplo da @Autowired, pelo @Inject da CDI.

Separação explícita e severa de papéis: Cada papel, seja controller, objeto de comando, objeto de formulário, objeto de domínio, DispatcherServlet, HandlerMapping ou ViewResolver, é descrito como um objeto único e especializado, o que acrescenta em conceitos como alta modularidade. Também ajuda na implementação do padrão MVC.

Adaptabilidade, não intrusividade e flexibilidade: Pode-se definir qualquer assinatura (como @RequestParam, @RequestHeader, @PathVariable e mais) para os métodos dos controllers, o que torna os métodos mais flexíveis e de fácil manutenção de acordo com sua ação a ser realizada, facilita também o tratamento dos dados.

Validação e amostra de erros personalizada: O framework permite que a validação seja mais flexível e definida pelo Bean Validation, especificação do Java EE que define-se uma série de anotações e uma API para criação de validações para serem utilizadas em Java Beans, que podem ser validados agora em qualquer camada da aplicação. (CAELUM, 2012). Através da anotação Valid, o framework identifica onde a validação deve ser feita, facilitando o tratamento das regras de negócio. O framework pode guardar os erros de validação em um objeto do tipo BindingResult, que evita o lançamento de exceções e melhora a interação com o usuário.

Mapeamento de URL e resolução de visões customizáveis: O Handler Mapping e o ViewResolver permitem através da manipulação de URLs em uma configuração simples a construção de estratégias de resolução sofisticadas e específicas.

Transferência flexível de modelos: O framework permite a manipulação e transferência de dados dos modelos fácil pelos diversos fluxos da requisição.

Suporte a JSP e própria biblioteca de tags: O framework disponibiliza suporte a utilização de JSP e possui sua própria biblioteca de tags que facilita a interação dos dados com a visualização.

Lightweight: O Spring pode ser implantado de forma fácil e leve em um WEB Container como o Tomcat ou Jetty, e para implementar, por exemplo apenas uma parte dele, não é necessário utilizar todos os componentes Spring. Isso torna o Spring MVC de mais fácil implantação.

RESTful: Possui recursos que permitem a criação de uma API RESTful (Que aplicam os princípios REST). O REST consiste em princípios/regras/constraints que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. Desta forma, permitindo, por exemplo, que aplicações se comuniquem. (PIRES, 2017)

2.5.3 Principais Componentes

É usual a presença de alguns componentes quando se implementa o framework Spring MVC. Pode-se citar:

DispatcherServlet: É a implementação Spring do front controller. É o primeiro controller que interage com as requisições. É onde também se implementa a interface Servlet. Ele controla todo o fluxo da aplicação.

Controller: É o componente usado para processar as requisições. Ele encapsula a lógica de navegação. É também o controller que delega serviços para a parte de Service da camada Model.

View: É responsável por renderizar a visualização, assim como no padrão MVC.

ModelAndView: É um tipo de objeto que indica a view que será mostrada e guarda dados dos modelos que devem estar presentes na view, interagindo com o ViewResolver. É gerenciado pelo Controller daquela requisição.

ViewResolver: É uma interface que auxilia a seleção de páginas retornadas pelas ações.

HandlerMapping: Mapeia requisições específicas para os controllers individuais. Identifica a requisição e chama o controller que lida com aqueles serviços específicos.

Anotações: Não é um componente muito específico, porém as anotações são amplamente usadas para facilitar a programação, como nas validações, definições e mapeamento de URL, definindo tipos para a classe, CDIs etc.

2.5.4 Comparativo de Frameworks MVC

Este tópico dedica-se à comparar e contrastar os frameworks Spring MVC, JSF, Play 2 e Struts 2. Será utilizado como parâmetro as seguintes categorias:

Velocidade de prototipagem

Complexidade do framework

Escalabilidade

Complexidade do Framework

Spring MVC: A arquitetura do Spring MVC é relativamente simples, mas ainda há muitas camadas e abstrações que às vezes são difíceis de depurar se algo for feito errado. Além de ser altamente dependente do Spring Core. É um framework antigo e maduro que tem uma quantia numerosa de modos para estendê-lo e configurá-lo, tornando-o, de certa forma, bastante complexo.

JSF: O JSF é incrivelmente complexo e essa é a maior fraqueza do framework. No entanto, isso não é culpa do JSF, pois isso se deve à especificação do Java EE e à inflexibilidade de runtimes disponíveis. Existem implementações de código aberto da especificação JSF que permitem o uso de contêineres, como o Tomcat, que reduz tremendamente a complexidade de ter que executar um servidor de aplicativos Java

Enterprise completo. A complexidade vem com o benefício de acesso ao restante da pilha do Java EE.(ZeroTurnaround, 2013)

Play: Play é um framework complexo, isso pode ser explicado ao considerar o quão robusto é o ecossistema que ele fornece. O framework vem com Netty, SBT, Akka, um template Scala, e vários outros módulos embutidos. Play enfatiza a convenção sobre a configuração, então o framework é responsável por realizar através do scaffolding, a ligação e configuração entre módulos, o que reduz a complexidade experimentada pelo desenvolvedor.

Struts: Struts é um MVC puro com uma arquitetura direta, não possuindo nenhum componente extra que possa adicionar complexidade.

Velocidade de prototipagem

Spring MVC: Spring não deve ser utilizado se gerar um projeto rápido e limpo é necessário. Esse framework é massivo e difícil para compreender quando se está começando. Para começar a prototipar é essencial possuir muito conhecimento preexistente sobre Spring. Além disso JSP natural e Controladores não fornecem componentes prontos para que possam ser usados.

JSF: JSF não é possui ferramentas build-in e apresenta tantas configurações para a prototipagem quanto para uma aplicação completa. Entretanto esses problemas que afetam a velocidade de prototipagem são mitigados através de ferramentas como o Maven,os diversos exemplos de projetos disponíveis online e pelo suporte das IDEs, que geram a maioria dos configurações e códigos clichês.

Play: Iniciar usando o framework Play é simples. O Play possui um componente binário que realiza uma técnica chamada scaffolding, tornando a prototipagem extremamente rápida. A seção Getting Started da documentação do framework Play é um ótimo tutorial básico e há poucas barreiras para desenvolvedores tornarem-se, pelo menos, moderadamente produtivos.(ZeroTurnaround, 2013)

Struts: Considerado por muitos uma tecnologia legado, o Struts não fornece muitas facilidades para geração de códigos. Além disso, é necessário realizar um elevado número de configurações para iniciar a prototipar. Segundo (ZeroTurnaround, 2013), o Struts tem um plugin de convenção, que impõe alguns protocolos sobre configurações e fornece anotações para configurar mapeamentos de URL.

Escalabilidade

Spring MVC: Spring MVC tem como uma de suas características fortes a escalabilidade, tanto que é usado em grande escala mundialmente. Spring contém componentes que realizam processos paralelos e através do Spring Batch, é possível criar aplicações multi-threaded. Spring pode ser modularizado e conectado através do JMS(Java Message

Service).

JSF: JSF necessitada da especificação Java EE para implementar chamadas assíncronas e schedules.

Play: Altamente escalável, Play permite alta performance quando acoplado com a implementação Akka. Pode ser modularizado com suporte de JSON à integração web.

Struts: Struts é um framework que possui como característica uma fraca escalabilidade, possuindo como único recurso de ajuda nesse quesito, o suporte AJAX.

3 Conclusões

Em suma, o MVC é um padrão arquitetural de software que separa a aplicação entre as camadas Modelo, Visão e Controle, e ele consegue reduzir o acoplamento e aumentar a coesão de um projeto, além de aumentar sua flexibilidade, visto que as interfaces são isoladas das regras de negócio. Os Frameworks MVC são soluções de código para resolver problemas específicos do MVC, sendo divididos entre os baseados em componentes e baseados em ações. Os frameworks baseados em componentes conseguem gerar uma abstração maior no desenvolvimento WEB com componentes visuais de alto nível, e isto pode ser observado no Framework JSF, que possui um processamento iniciado na visão. Já os frameworks baseados em ações, como Struts, Play e Spring MVC, são menos acoplados com a visão, iniciando seu processamento sempre pela camada de controle.

Este trabalho elencou a utilidade dos frameworks MVC por meio de uma análise focada no Spring MVC, um framework moderno e muito utilizado que possui funcionalidades e características benéficas no desenvolvimento de um software, como Inversão de Controle, CDI, flexibilidade, configurações personalizáveis, recursos RESTful, entre outras. Ademais, o Spring MVC também é conhecido pela organização que ele provê, pois gera uma separação severa de papéis entre seus componentes, aumentando sua modularidade e auxiliando na implementação do padrão MVC.

Referências

AFONSO, A. *O que é Spring MVC?* 2017. Último acesso em 30/07/2018. Disponível em: <http://blog.algaworks.com/spring-mvc/>. Nenhuma citação no texto.

ALMEIDA, A. *Entenda os MVCs e os frameworks Action e Component Based.* 2012. Último acesso em 30/07/2018. Disponível em: <http://blog.caelum.com.br/entenda-os-mvcs-e-os-frameworks-action-e-component-based/>. Nenhuma citação no texto.

APACHE. *Tag Reference.* 2018. Último acesso em 03/08/2018. Disponível em: <https://struts.apache.org/tag-developers/tag-reference.html>. Citado na página 17.

BOAGLIO, F. *Play Framework.* 2014. Último acesso em 03/08/2018. Disponível em: <https://www.casadocodigo.com.br/pages/sumario-play-framework-java>. Citado na página 17.

BRIKMAN, Y. *The Play Framework at LinkedIn.* 2013. Último acesso em 03/08/2018. Disponível em: <https://engineering.linkedin.com/play/play-framework-linkedin>. Citado na página 17.

CAELUM. *Java para Desenvolvimento Web.* 2012. Último acesso em 30/07/2018. Disponível em: <https://www.caelum.com.br/apostila-java-web/>. Citado 2 vezes nas páginas 19 e 20.

CAELUM. *Lab. Java com Testes, JSF e Design Patterns.* 2014. Último acesso em 04/08/2018. Disponível em: <https://www.caelum.com.br/apostila-java-testes-jsf-web-services-design-patterns/>. Citado na página 16.

CHAUBEY, M. *Pros and cons of Struts.* 2012. Último acesso em 03/08/2018. Disponível em: <http://www.indiastudychannel.com/resources/148944-Pros-and-cons-of-Struts.aspx>. Citado na página 17.

COUTO, R. S.; FOSCHINI, I. J. Análise comparativa entre dois frameworks mvc para a plataforma java ee: Jsf e vraptor. T.I.S. - Tecnologias, Infraestrutura e Software, p. 249–259, 2015. Nenhuma citação no texto.

GUPTA, P.; GOVIL, P. M. Spring web mvc framework for rapid open source j2ee application development: A case study. International Journal of Engineering Science and Technology, p. 1684–1689, 2010. Citado na página 19.

KUMAR, S. *Why is Java Spring framework called a "lightweight"framework?* 2015. Último acesso em 30/07/2018. Disponível em: <https://www.quora.com/Why-is-Java-Spring-framework-called-a-lightweight-framework>. Nenhuma citação no texto.

MEDEIROS, H. *Introdução ao Padrão MVC.* 2013. Último acesso em 30/07/2018. Disponível em: <https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>. Nenhuma citação no texto.

PIRES, J. *O que é API? REST e RESTful? Conheça as definições e diferenças!* 2017. Último acesso em 30/07/2018. Disponível em: <https://becode.com.br/o-que-e-api-rest-e-restful/>. Citado na página 20.

RAMOS, A. *O que é MVC?* 2015. Último acesso em 30/07/2018. Disponível em: <https://tableless.com.br/mvc-afinal-e-o-que/>. Nenhuma citação no texto.

REENSKAUG, T. M. H. *MODELS - VIEWS - CONTROLLERS*. 1979. Disponível em: <https://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf>. Citado 2 vezes nas páginas 13 e 14.