

LIGA DE ENSINO DO RIO GRANDE DO NORTE
CENTRO UNIVERSITÁRIO DO RIO GRANDE DO NORTE
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

JOSÉ DIEGO MARQUES RAMOS

UTILIZANDO SPRING BOOT NO DESENVOLVIMENTO DE APLICAÇÕES COM
MICROSSERVIÇOS

NATAL/RN

2016

JOSÉ DIEGO MARQUES RAMOS

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

Trabalho apresentado como requisitos de avaliação à disciplina de Estágio Supervisionado do curso de graduação em Sistemas de Informação.

Professo: Ytalo Rosendo do Amaral

Natal/RN

2016

Catálogo na Publicação – Biblioteca do UNI-RN
Setor de Processos Técnicos

Ramos, José Diego Marques.

Utilizando spring boot para o desenvolvimento e aplicações com
microsserviços. / José Diego Marques Ramos. – Natal, 2016.
36 f.

Orientador: Prof^a. Me. Rômulo Fagundes Cantanhede.

Monografia (Bacharelado em Sistemas de Informação) – Centro
Universitário do Rio Grande do Norte.

1. Spring Boot - Monografia. 2. Spring Framework - Monografia. 3.
Microsserviços - Monografia. I. Cantanhede, Rômulo Fagundes. II. Título.

RN/UNI-RN/BC

CDU 004

JOSÉ DIEGO MARQUES RAMOS

**UTILIZANDO SPRING BOOT PARA O DESENVOLVIMENTO E APLICAÇÕES
COM MICROSERVIÇOS**

Trabalho de Conclusão de Curso Bacharelado em Sistemas de Informação apresentado ao centro Universitário do Rio Grande do Norte (UNI-RN) como requisito final para obtenção do título de Bacharel em Sistemas de Informação.

ORIENTADOR: Prof. Msc. Romulo Fagundes Cantanhede

NATAL/RN

2016

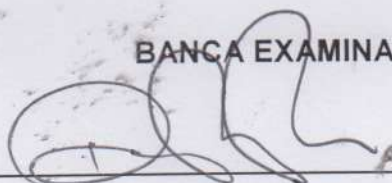
JOSÉ DIEGO MARQUES RAMOS

**UTILIZANDO SPRING BOOT PARA O DESENVOLVIMENTO E APLICAÇÕES
COM MICROSERVIÇOS**

Trabalho de Conclusão de Curso
Bacharelado em Sistemas de Informação
apresentado ao Centro Universitário do Rio
Grande do Norte (UNI-RN) como requisito
final para obtenção do título de Bacharel
em Sistemas de Informação.

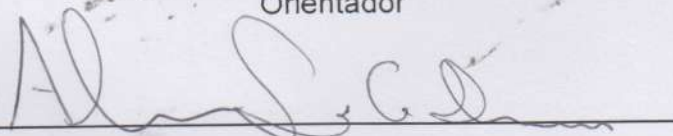
Aprovado em: 05 / 12 / 2016.

BANCA EXAMINADORA



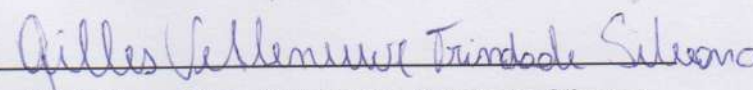
Prof. Msc. Romulo Fagundes Cantanhede

Orientador



Prof. Msc. Alexandre Luiz Galvão Damasceno

Membro



Prof. Msc. Gilles Veleneuve Trindade Silvano

Membro

À minha mãe Zildete, minha esposa
Heloísa e ao meu filho Benjamim, dedico
este trabalho por serem inspiração e
motivo de todas as ações da minha vida.

AGRADECIMENTO

Agradeço a Deus em primeiro lugar, afinal, todas as coisas são feitas por seu intermédio e sob sua vontade.

À minha esposa **Heloísa** e ao meu filho **Benjamim**, por todo amor, carinho e inspiração que me concedem.

Agradeço aos amigos **Rafael, Helvétius, Maxwell e Wilson**, que foram sempre de grande ajuda durante o curso dessa empreitada, dedico a vocês um grande abraço.

Ao orientador **Romulo** pelos direcionamentos neste trabalho.

RESUMO

Este trabalho possui a finalidade de estudar a aplicação do framework Spring Boot no desenvolvimento de aplicações baseadas na Arquitetura de Microserviços. Sendo o Spring Boot uma ferramenta de alavancagem de produtividade e qualidade software, facilita a criação de projetos de software baseados na linguagem Java que fazem uso do Spring Framework. Estudaremos a gênese do Spring Boot a partir de sua relação com o Spring Framework e, conseqüentemente, a relação deste último com as especificações da plataforma Java EE. Observaremos que Microserviços são naturalmente ágeis, por serem focados na solução de funcionalidades específicas, facilmente publicados e altamente distribuíveis, o Spring Boot vem complementar estas características de microserviços potencializando o processo de desenvolvimento através da definição opinativa de configurações, controle de dependências e facilidade na criação de novos projetos com baixo nível de esforço despendidos pelos desenvolvedores. Serão verificadas as vantagens que podem surgir a partir da união dessas duas tecnologias.

Palavras-Chave: Spring Boot, Spring Framework, Microserviços.

ABSTRACT

This work has the purpose of studying the application of the Spring Boot framework in the development of applications based on the Architecture of Microservices. As Spring Boot is a leverage tool for productivity and software quality, it facilitates the creation of software projects based on the Java language that make use of the Spring Framework. We will study the genesis of Spring Boot from its relation with the Spring Framework and, consequently, the relationship of the latter with the specifications of the Java EE platform. We will observe that Microservices are naturally agile, because they are focused on the solution of specific, easily published and highly distributable functionalities, Spring Boot complements these micro-services characteristics, enhancing the development process through the opinative definition of configurations, dependency control and ease of creation Of new projects with low level of effort detached by the developers. The advantages that can arise from the union of these two technologies will be verified.

Keywords: Spring Boot, Spring Framework, Microservices.

LISTA DE ILUSTRAÇÕES

Figura 1 – Código Groovy	25
Figura 2 – Execução Spring Boot	26
Figura 3 – Reposta	27
Figura 4 – Novo Projeto STS	28
Figura 5 – Novo Projeto STS Definições	28
Figura 6 – pom.xml	29
Figura 7 – Arquivo de Propriedades	30
Figura 8 – Classe Principal	30
Figura 9 – Classe de Domínio	31
Figura 10 – Repositório	32
Figura 11 – Serviço	32
Figura 12 – Manipulador de Exceções	33
Figura 13 – Artigo Recurso	34

LISTA DE ABREVIATURAS E SIGLAS

Java EE	Java Enterprise Edition
IoC	Inversion of Control
DI	Dependency Injection
EJB	Enterprise Java Beans
POJO	Plain Old Java Objects
MVC	Model-View-Controller

SUMÁRIO

1. INTRODUÇÃO	11
2. OBJETIVO	12
2.1. OBJETIVO GERAL	12
2.2. OBJETIVO ESPECÍFICO	12
2.3. JUSTIFICATIVA	12
3. REFERENCIAL TEÓRICO	13
3.1. JAVA EE	13
3.1.1. Problemas Apresentados Pelo Java EE	14
3.1.1.1. Dificuldade de Compressão das Especificações	14
3.1.1.2. Complexidade na Configuração de Projetos	14
3.1.1.3. Desperdício de Recursos	15
3.2. O SPRING FRAMEWORK	15
3.2.1. O Porquê da Criação do Spring Framework	15
3.2.2. Princípios do Spring Framework	16
3.2.2.1. Injeção de Dependências	16
3.2.2.2. Programação Orientada a Aspectos	16
3.3. O SPRING BOOT	17
3.3.1. Como Funciona	18
3.3.2. Gestão de Dependências	19
3.3.2.1. Maven	19
3.4. MICROSERVIÇOS	19
3.4.1. Princípios de Microserviços	20
3.4.1.1. Uma Responsabilidade por Serviço	21
3.4.1.2. Microserviços são autônomos	21
3.4.2. Características de Microserviços	21
3.4.2.1. Acessibilidade à Microserviços	21
3.4.2.2. Contrato de Serviços	22
3.4.2.3. Baixo Acoplamento	22
3.4.2.4. Reuso de Serviços	23
3.4.2.5. Não compartilhamento de estado	23
3.4.2.6. Descoberta de serviços	24
3.4.2.7. Interoperabilidade de serviços	24
3.4.2.8. Composição de Serviços	25

4. CRIAÇÃO DE PROJETOS BASEADO EM SPRING BOOT	25
4.1. UTILIZANDO O COMMAND-LINE TOOL	25
4.2. UTILIZANDO O SPRING TOOLS SUITE	27
CONCLUSÃO	35
REFERÊNCIA BIBLIOGRÁFICA	36

1. INTRODUÇÃO

Desde o lançamento de sua primeira versão estável em abril de 2014, o Spring Boot trouxe para o cenário de desenvolvimento de aplicações e microsserviços uma ferramenta para alavancagem de produtividade, qualidade de código e facilidade na criação de projetos no desenvolvimento de software. Para tal, o Spring Boot adota um conceito opinativo de configuração, ou seja, as configurações necessárias ao projeto que não sejam explicitamente declaradas serão definidas através de convenção baseada nas principais características do projeto criado. Vale enfatizar, também, que à semelhança da definição opinativa de configuração, o Spring Boot possui atribuição opinativa de dependências de projeto, por exemplo a definição do Web Container Tomcat como padrão para aplicações web ou a *utilização do Thymeleaf* como framework para desenvolvimento front-end.

Para compreendermos o motivo pelo qual o Spring Boot foi criado e sua aplicação, é necessário entendermos o cenário que viabilizou a criação do Spring Framework, do qual o Spring Boot faz parte, e sua relação com a tecnologia J2EE, hoje conhecida como Java EE. Por uma questão didática a partir deste ponto, trataremos a tecnologia J2EE pelo seu nome mais recente e mais usado, o chamaremos apenas de Java EE.

Também abordaremos neste trabalho o modelo de arquitetura em microsserviços, largamente utilizando na atualidade por empresas consolidadas e quem possuem um nível altíssimo de dependência à tecnologia aplicada ao negócio, a título de exemplo podemos citar a Netflix e a Uber.

Neste trabalho abordaremos a facilidade para desenvolvimento de aplicações que o Spring Boot trouxe e sua aplicação ao modelo de arquitetura de microsserviços, desta forma foram unidos um poderoso framework para ganho de desempenho em desenvolvimento e um modelo de arquitetura moderna repleta de benefícios para produção de aplicações distribuídas.

2. OBJETIVO

2.1. OBJETIVO GERAL

O binômio formado por produtividade e qualidade de código apresenta um dos maiores objetivos das tecnologias de desenvolvimento de softwares, o desenvolvimento rápido de aplicações com o mínimo de bugs transitando entre as fases de concepção e entrega do produto. O objetivo deste trabalho é verificar a capacidade do framework Spring Boot em potencializar a solução desta necessidade. Partiremos de sua premissa em abstrair configurações e controle de dependências permitindo que os desenvolvedores foquem seus esforços nas regras de negócio das aplicações para verificarmos o êxito do framework na tarefa supracitada.

2.2. OBJETIVO ESPECÍFICO

- Verificar o impacto do conceito de convenção sobre configuração no desenvolvimento de microsserviços.
- Verificar vantagens em utilizarmos o Spring Boot
- Desenvolver um exemplo de microsserviço utilizando o Spring Boot na criação do projeto.

2.3. JUSTIFICATIVA

É necessário entendermos o impacto que tecnologias como Spring Boot ocasionam no desenvolvimento de aplicações, a necessidade de processos de desenvolvimento cada vez mais ágeis e simples de serem iniciados favorece a utilização de frameworks de alavancagem de produtividade, e temos no Spring Boot um expoente deste nicho de mercado, atendendo a diversos modelos de aplicação, vemos, na utilização deste framework, a possibilidade de alcançarmos os objetivos propostos

3. REFERENCIAL TEÓRICO

3.1. JAVA EE

O Java EE faz parte da plataforma Java, que originalmente foi criada para desenvolvimento de aplicações embarcadas. Sendo o Java EE responsável por especificar a forma de desenvolvimento para aplicações corporativas baseadas em arquiteturas multicamadas, modulares e distribuídas, sendo necessário o uso de servidores de aplicações para implementação de aplicações que fazem uso das especificações Java EE. Um conjunto de especificações, sendo algumas delas: JDBC, JPA, JSF, JNDI, JTA e outras mais, compõem o Java EE. Também foram apresentadas definições de componentes como EJB's, Servlets, Portlets e diversas tecnologias web. O objetivo do Java EE é permitir a criação de aplicações corporativas portáteis entre servidores de aplicações com foco no negócio.

Todas essas especificações, ao serem aplicadas em forma de infraestrutura, permitem a criação de aplicações corporativas. Tais aplicações são atendidas através de Java EE em suas necessidades de concorrência, escalabilidade, portabilidade entre servidores que implementam as especificações Java EE, gestão de componentes e aplicações publicadas.

Em acordo com Souza (2011, p. 15)

J2EE (Java EE) foi criada para desenvolvimento de aplicações de grande porte, deixando muito complexa o desenvolvimento de aplicações menores. Para amenizar e reduzir esse problema surgiram os frameworks, criados pela comunidade de desenvolvedores Java.

O objetivo proposto pelo Java EE é conceder ao desenvolvedor o foco no desenvolvimento das funcionalidades de negócio da aplicação e alavancar a produtividade e qualidade.

3.1.1. Problemas Apresentados Pelo Java EE

Mesmo tendo alcançado um sucesso razoável, o Java EE apresentou alguns problemas, e estes favoreceram o surgimento de alternativas que intencionavam dar uma resposta às dificuldades encontradas na aplicação das especificações Java EE, sendo o Spring Framework uma das possíveis soluções.

3.1.1.1. Dificuldade de Compressão das Especificações

Conforme enfatizado por Oki (2015)

...apesar de todo o cuidado e visão das especificações, o como elas eram implementadas e coordenadas em um servidor de aplicações Java EE ficou bastante vago e até mesmo ausente em alguns aspectos nas primeiras versões...

Ainda que as especificações do Java EE objetivassem o desenvolvimento de aplicações portáteis entre os servidores, não havia clareza na forma como as especificações deveriam ser implementadas e coordenadas nos servidores, isto acabou resultando na utilização de ferramentas próprias para configuração e especializações de configuração, fazendo com que os projetos não fossem portáteis entre os servidores.

3.1.1.2. Complexidade na Configuração de Projetos

Outro problema da plataforma Java EE era a dificuldade para configuração dos projetos, a configuração de projetos Java EE era complexa, em acordo com o que descreve Oki "Desde o início soube-se da complexidade da configuração no Java EE" (Oki, 2015), e possuía características únicas de acordo com o servidor de aplicações utilizado.

Conforme Weissmann (2012, p. 17)

Como não haviam ainda anotações, o desenvolvedor ao implementar seus EJB's precisava satisfazer uma série de interfaces e ainda gastava um

bom tempo escrevendo arquivos de configuração no formato XML, como o famigerado ejb-jar.xml.

3.1.1.3. Desperdício de Recursos

Por usar um servidor de aplicações, um projeto Java EE conta com a disponibilidade de todos os recursos deste, ainda que os recursos não sejam utilizados, e conseqüentemente deverá pagar

Ao contar com todas as facilidades oferecidas por servidores de aplicação, um projeto Java EE deve arcar com todo ônus gerado pela disponibilidade de tais recursos, ainda que eles não sejam utilizados. Em termos práticos aplicações que seriam perfeitamente atendidas por Containers Web, como o Tomcat ou Jetty, eram obrigadas a serem publicadas em Servidores Web mais custosos como o JBoss.

3.2. O SPRING FRAMEWORK

3.2.1. O Porquê da Criação do Spring Framework

“Ele (Spring Framework) desafiou os modelos de programação pesados da época com um modelo de programação mais leve baseado em POJO's”(WALLS, 2015). Apresentado ao público em 2004 por Rod Johnson, o Spring Framework é um framework de código aberto lançado com o objetivo de fazer frente a complexidade de desenvolvimento e manutenção de aplicações corporativas que utilizam as especificações do Java EE, sendo ele uma alternativa menos custosa em termos de tempo de entrega de aplicações, com maior facilidade no gerenciamento e melhoria de performance. De acordo com Gutierrez, “O Spring Framework provê um modelo de programação e configuração para criação de aplicações corporativas em Java.” (Gutierrez, 2014, p. 2, Tradução nossa). Possui em sua concepção o ideal de permitir o desenvolvimento de aplicações que através do Java EE obrigaria as equipes de desenvolvimento a utilizarem EJB's.

Conforme afirma JR (2016)

O Spring não é um framework apenas, mas um conjunto de projetos que resolvem várias situações do cotidiano de um programador, ajudando a criar aplicações Java com simplicidade e flexibilidade.

É composto por diversos módulos como o Spring Data, Spring Security e outros. Essa modularidade permite que ele seja utilizado em todo ou em partes de acordo com a necessidade de cada aplicação. Seu módulo principal, que pode ser utilizado em qualquer aplicação Java, implementa um container de inversão de controle – IoC que faz uso de injeção de dependências e Programação Orientada a Aspectos.

O uso extensivo de arquivos XML para configurar o Spring Framework foi um dos pontos sensíveis da plataforma, entretanto, após a versão 2.5 o cenário é alterado através da adoção de anotações como meio de configuração.

3.2.2. Princípios do Spring Framework

3.2.2.1. Injeção de Dependências

“O Spring Framework é baseado em um princípio simples: Injeção de Dependência” (Gutierrez, 2014, p. 7, tradução nossa). A utilização de Injeção de Dependências fez com que a comunidade adotasse o Spring Framework, pois atua diretamente a favor de um poderoso conceito de desenvolvimento o baixo acoplamento. Aplicações com baixo acoplamento são mais extensíveis, modularizáveis, testáveis e alcançam um maior nível de qualidade quando comparadas a aplicações fortemente acopladas. Há, também, o entendimento de que utilizar Injeção de Dependências no desenvolvimento de aplicações favorece a produtividade dos desenvolvedores.

3.2.2.2. Programação Orientada a Aspectos

Surgiu como um complemento à Programação Orientada a Objetos, possui o objetivo de facilitar a modularização do projeto e reduzir a quantidade de código produzido. Foi desenvolvido por Gregor Kiczales e sua equipe na Xerox Parc,

juntamente com a AspectJ que é considerada a primeira e mais popular linguagem POA.

POA permite separação de código de acordo com a sua importância para o projeto, ou seja, código que são muito importantes para determinados objetos mas não estão diretamente relacionados aos seus escopos. Geralmente aspectos não estão ligados a regra de negócio da aplicação, sendo mais difundido o seu uso para controle de segurança, gerenciamento de log, transações e tratamentos de exceções.

A POA utiliza os seguintes conceitos para implementação de suas funcionalidades:

Aspectos: Conjunto de pointcuts e advices

Advices: Código a ser executado, seria tecnicamente a implementação do aspecto, define o que deve ser feito e quando será feito.

Pointcut: Onde será executado o advice.

Joinpoint: Disponibilizam os pontos (métodos) onde o aspecto será conectado para execução na aplicação.

3.3.O SPRING BOOT

“O Spring Boot é uma ferramenta para o desenvolvimento de aplicações que utilizam o framework Spring, sem que seja necessária praticamente nenhuma configuração.” (FELIPE, 2016), sendo o Spring Boot uma ferramenta de alavancagem de produtividade, faz uso da experiência do desenvolvedor no uso do Spring Framework, em todos os seus módulos, sem o ônus de consumir muito tempo para configuração do. Nele é possível desenvolver aplicações com infraestrutura enxuta e embarcada, permitindo que os cuidados com a infraestrutura de servidores web seja minimizado, de tal forma que cada funcionalidade ou microserviço disponibilizado poderá utilizar um servidor web ou container que melhor atender as suas necessidades específicas, vejamos o que diz Gutierrez sobre isso: “Você vai descobrir que, com o Spring Boot, você pode criar aplicativos autônomos que usam um servidor

incorporado, tornando-os aplicativos 100% executáveis.” (GUTIERREZ, 2016, p. 1, tradução nossa).

Algo importante a ser salientado em relação ao Spring Boot é fato dele não ser um gerador de código conforme enfatizado por Gutierrez ao descrever características do Spring Boot, “Spring Boot não é um gerador de código.” (GUTIERREZ, 2016, p. 6, tradução nossa).

Partindo da premissa de convenção sobre configuração, o Spring Boot permite a criação de projetos para desenvolvimento de software de forma ágil e precisa, convencionando as configurações e dependências a partir da identificação das principais características da aplicação a ser desenvolvida. “O Spring Boot trouxe agilidade, e te possibilita focar nas funcionalidades da sua aplicação com o mínimo de configuração” (JR, 2016, p. 13). Assim sendo, o ganho de produtividade fica sensivelmente perceptível, tendo em vista que o desenvolvendo não utilizará seu tempo em configuração de ferramentas e frameworks necessários ao projeto, Gutierrez também comenta esta característica do Spring Boot ao afirmar que é possível, através deste framework, criar aplicações prontas para serem executadas sem realizar a configuração dos arquivos XML, “O Spring Boot permite que você crie aplicativos sem precisar de configuração usando arquivos XML.” (GUTIERREZ, 2016, p. 6).

3.3.1. Como Funciona

Deve existir uma classe principal que habilitará o funcionamento do Spring Boot através da anotação `@SpringBootApplication`, entraremos em mais detalhes ao verificarmos nossa API de exemplo, desta forma as configurações da aplicação serão definidas pelo Spring Boot e todas as classes que estiverem no mesmo pacote da classe anotada, e nos pacotes filhos, serão escaneadas e gerenciadas pelo Spring Boot,

Conforme afirma Gutierrez(2016, p 40, tradução nossa)

“O Spring Boot usará autoconfiguração com base no seu *classpath*, suas anotações e sua configuração para adicionar corretamente as

tecnologias necessárias e criar uma aplicação adequada. Isso significa que todas essas anotações facilitam como o Spring Boot irá configurar a sua aplicação.”

Contudo, ainda é possível sobrescrever configurações definidas pelo Spring Boot através do arquivo `application.properties`.

3.3.2. Gestão de Dependências

Para realizar o gerenciamento de dependências é possível escolher entre o Maven e o Gradle, ambos são suportados pelo Spring Boot.

Vejamos o que afirma Gutierrez (2016, p. 13, tradução nossa)

Se você já usa o Maven (<https://maven.apache.org/>) ou o Gradle (<http://gradle.org/>) como ferramentas para compilação, testes e construção, você também pode usar Spring Boot. E como você pode imaginar, você precisa incluir algumas dependências para usar Spring Boot.

3.3.2.1. Maven

Apache Maven é uma ferramenta de gerenciamento e compreensão de projetos de software. Com base no conceito de um modelo de objeto de projeto (POM), o Maven pode gerenciar a compilação, o relatório e a documentação de um projeto a partir de uma informação central.

3.3.2.2. Gradle

Gradle é uma ferramenta de automação de compilação de código aberto para a Era de Distribuição Contínua (CD).

3.4. MICROSERVIÇOS

Microserviços consistem em um estilo de arquitetura de software utilizado por diversas organizações como um meio de alcançar um novo grau em agilidade,

velocidade de entrega e escala. "Microserviços nos provê um meio de desenvolvimento de sistemas com módulos fisicamente separados." (RV, 2016, tradução nossa)

De acordo com RV, "microserviços são um modelo arquitetural disruptivo que alcança um novo patamar em agilidade de desenvolvimento de software, aumento da velocidade de disponibilização de sistemas e escalonamento." (RV, 2016) Com tais características, microserviços permite às empresas de desenvolvimento de sistemas entregar software com qualidade e agilidade. Somando-se a isto a capacidade de entrega de módulos fisicamente separados, esta tecnologia favorece a entrega e manutenção de funcionalidades de software sem impacto direto sobre as outras funções existentes.

Conforme afirmam FOWLER e LEWIS (2014, tradução nossa)

Resumindo, microserviços são uma abordagem para o desenvolvimento de uma única aplicação, ou sistema, como um conjunto de pequenos serviços, cada um rodando em seu próprio processo e se comunicando através de mecanismos leves, geralmente uma API HTTP. Estes serviços são construídos através de pequenas funcionalidades de negócio e publicados em produção de maneira independente através de processos de *deploys* automatizados. Há um gerenciamento centralizado mínimo para estes serviços, que podem ser escritos em diferentes linguagens e usarem diferentes tecnologias para armazenamento de dados.

Sob o ponto de vista de Fowler e Lewis, microserviços é caracterizado pelo desenvolvimento de um conjunto de serviços leves e autônomos que rodam em infraestrutura própria e independente. A comunicação entre os serviços ocorre através de protocolos de comunicação leves. Ainda que a abordagem de desenvolvimento de sistemas através de microserviços permita a utilização de diversas linguagens de programação distintas e diversos mecanismos de armazenamento de dados, é necessário que exista um gerenciamento mínimo centralizados para os microserviços.

3.4.1. Princípios de Microserviços

3.4.1.1. Uma Responsabilidade por Serviço

“Ele afirma que uma unidade (serviço) só deve ter uma reponsabilidade”. (RV, 2016, *tradução nossa*)

O princípio de responsabilidade única define uma limitação que cada unidade deve possuir, ele apregoa que um serviço deve possuir apenas uma responsabilidade, ou seja, deve atender apenas a uma funcionalidade do sistema e não deve compartilhá-la com outras funções do sistema. Este princípio está de acordo com o princípio de responsabilidade do padrão de projetos SOLID.

Ao utilizar definir apenas uma responsabilidade por serviço o baixo acoplamento é favorecido nas aplicações estruturadas em microsserviços.

3.4.1.2. Microsserviços são autônomos

Eles são autossuficientes, independentemente implementáveis e possuem toda responsabilidade sobre sua execução e regras de negócio. (RV, 2016, *tradução nossa*)

Autonomia é um forte princípio de microsserviços, para atender este princípio é necessário que sejam assumidas inúmeras responsabilidades por cada serviço integrante do conjunto de serviços. Estão incluídas nestas responsabilidades todo o seu ambiente de execução e as regras de negócios da aplicação.

Serviços autônomos empacotam suas dependências, bibliotecas, servidores web e containers, sendo estes últimos os responsáveis pela abstração de recursos físicos, assim, podendo rodar como processos autônomos.

3.4.2. Características de Microsserviços

3.4.2.1. Acessibilidade à Microsserviços

Microserviços são acessíveis através de API e abstraem os seus detalhes de implementação. A lógica interna, arquitetura, linguagem de programação, banco de dados, mecanismo de qualidade de serviços entre outras, tudo é completamente oculto àqueles que acessam os serviços por meio da API.

No mundo dos microservices, os serviços são cidadãos de primeira classe. Microservices expõem pontos de acesso aos serviços como APIs e abstraem todos os seus detalhes realizações. A lógica interna implementação, arquitetura e tecnologias (incluindo linguagem de programação, banco de dados, qualidade dos mecanismos de serviços, e assim por diante) são completamente escondidas por trás da API de serviços. (RV, 2016)

A título de exemplo, em uma aplicação desenvolvida em como microserviços que possua a funcionalidade de cadastro de produtos, haverá uma API que permita o cadastro de produtos e outra que permita a consulta aos produtos de forma que a lógica interna fica escondida das aplicações que acionem as APIs.

3.4.2.2. Contrato de Serviços

Semelhante a SOA, microservices são descritos por meio de contratos de serviços bem definidos. No mundo dos microservices, JSON e REST são universalmente adotados para a comunicação entre serviços. (RV, 2016, tradução nossa)

Contratos de serviço definem, ou padronizam, a forma de comunicação com os serviços acessíveis através da API. Ao desenvolvermos microserviços, por causa da sua liberdade para desenvolvimento e atualização de serviços, é bom evitar que ao realizar correções e atualizações nos serviços sejam evitadas alterações no contrato.

3.4.2.3. Baixo Acoplamento

Em conformidade com RV (2016, tradução nossa)

Microservices são independentes e de baixo acoplamento. Na maioria dos casos, os serviços aceitam um evento como entrada e respondem com outro evento. Messaging, HTTP e REST são comumente usados para interação entre microsserviços. endpoints baseadas em mensagens fornecem níveis mais elevados de dissociação.

Quando se desenvolve um sistema utilizando a arquitetura de microsserviços deve-se ter a visão de que cada serviço/funcionalidade deverá ser independente das demais por embarcar seu ambiente de execução e regras de negócios pode-se alcançar o baixo acoplamento em um nível altamente satisfatório. São adotadas, também, regras claras e bem definidas de comunicação entre os serviços utilizando padrões de comunicação largamente utilizados.

3.4.2.4. Reuso de Serviços

De acordo com RV (2016, tradução nossa)

Microservices são serviços granularizados e reutilizáveis. São acessados por dispositivos móveis e canais de desktop, outros microservices, ou mesmo outros sistema.

Segundo Rajesh RV, microsserviços são altamente reutilizáveis, considerando que cada serviço deverá conter uma responsabilidade única e não compartilhada em um nível de granularidade elevado e por adotar padrões de comunicação bem desenhados os serviços são acessíveis por diversos tipos de tecnologias como aplicações, outros serviços, dispositivos móveis e outros.

3.4.2.5. Não compartilhamento de estado

Não compartilhar estado é uma característica de microsserviços, devido à transparência ao usuário caso um serviço caia e seja substituído ou pela necessidade de resposta por qualquer serviço do ecossistema às requisições dos usuários, a arquitetura não pode limitar o funcionamento de suas funcionalidades a um estado armazenado, por exemplo, em sessão do usuário no servidor que recebeu

determinada consulta e impedindo o usuário de receber o retorno da transação em outro servidor caso o original caia durante o processamento da requisição,

Em conformidade com RV (2016, Tradução nossa)

Microservices bem desenhados não compartilham seu estado ou estado de conversação mantido pelos serviços. Havendo uma exigência para manter o estado do serviço é possível que seja mantido em um banco de dados ou talvez na memória.

3.4.2.6. Descoberta de serviços

Microservices podem ser descobertos. Em um ambiente típico microservices, microservices auto-propaganda de sua existência e se tornam disponíveis para a descoberta. Quando os serviços de morrer, eles automaticamente se retiram do ecossistema microservices. (RV, 2016, tradução nossa)

3.4.2.7. Interoperabilidade de serviços

Conforme afirma RV (2016, tradução nossa)

Serviços são interoperáveis devido a utilizarem protocolos padrão e padrões de troca de mensagens como Messaging, HTTP e outros. Mecanismos de transporte REST/JSON são os métodos mais populares para o desenvolvimento de serviços interoperáveis no mundo de microservices. Nos casos em que outras otimizações são necessárias nas comunicações, protocolos como Protocol Buffers, Thrift, Avro, ou Zero MQ também podem ser usados. No entanto, o uso desses protocolos podem limitar a interoperabilidade global dos serviços.

Os serviços criados de acordo com os conceitos de microserviços são naturalmente interoperáveis, diversos serviços podem trabalhar em conjunto para atingirem um objetivo fim. Considerando uma aplicação de e-commerce, um serviço de recuperação de carrinho de comprar poderá fornecer informações sobre os produtos selecionados pelo usuário para um serviço de emissão com vendas.

3.4.2.8. Composição de Serviços

Microservices são composeable. composeability serviço é conseguida quer através da orquestração de serviços ou coreografia serviço. (RV, 2016, tradução nossa)

4. CRIAÇÃO DE PROJETOS BASEADO EM SPRING BOOT

4.1. UTILIZANDO O COMMAND-LINE TOOL

A título de demonstração inicial do poder do Spring Boot, utilizaremos a ferramenta Command-Line para desenvolvermos uma simples aplicação web afim de mostrarmos a capacidade deste framework em atribuir um padrão de configuração de projetos que permite aos desenvolvedores direcionarem os seus esforços ao desenvolvimento de regras de negócio. Neste modelo criaremos um arquivo groovy e, através de anotações, configuraremos tal arquivo como um controller e o Spring Boot será responsável por definir o web container usado na publicação, bem como a configuração de porta desta aplicação.

Neste exemplo utilizaremos Spring Boot versão 1.4.1.

Figura 1 - Código Groovy

```

app.groovy
1 @RestController
2 Class SereiExecutada{
3
4     @RequestMapping("/")
5     String home(){
6         "Olá banca!"
7     }
8 }

```

Fonte: Código do autor

A Figura 1 apresentada o código da classe da groovy, pode-se observar o uso da anotação `@RestController`, ao utilizarmos esta anotação estamos informando ao

utilizada para realizar o deploy de uma aplicação, permitindo que o desenvolvedor esteja focado apenas na aplicação em si.

Figura 3 – Reposta



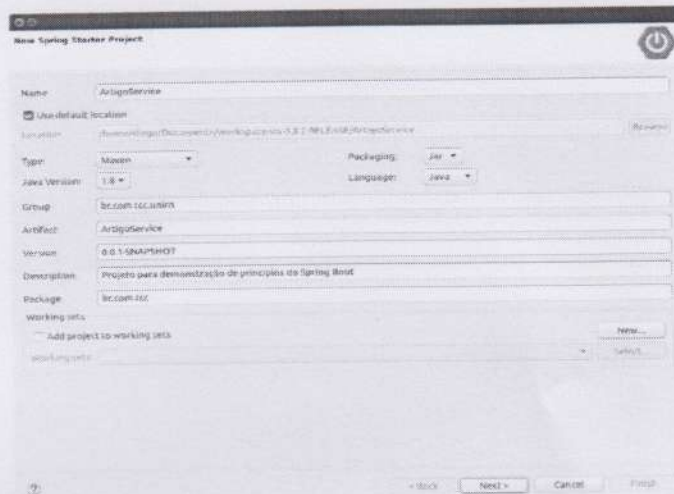
Fonte: o autor

4.2. UTILIZANDO O SPRING TOOLS SUITE

Neste ponto faremos uma API mais robusta que aplicação anterior e complementaremos os conceitos que já foram utilizados no tópico 4.1. Utilizaremos o Sprint Tool Suite que se trata de uma versão do eclipse com plug-ins do Spring já adicionados, o que auxilia no desenvolvimento de aplicações Spring. Deve-se ter em mente que para o Spring Cloud, módulo do Spring direcionado ao controle de balanceamento de carga, descoberta e registro de serviços, centralização de configurações e outros aspectos relativos a distribuição de microsserviços, é um potencializador dos resultados obtidos com o Spring Boot, facilitando o processo de implementação das principais características que geram o sucesso no desenvolvimento de aplicações baseadas na arquitetura de microsserviços.

Na Figura 4 temos a primeira tela utilizada para criação do projeto exemplo, que será chamado ArtigoService. Como pode ser verificado, optamos pela utilização do Maven para gerenciamento das dependências deste projeto e outro detalhe importante a ser notado é que, mesmo se tratado de uma aplicação o web, o empacotamento do arquivo é JAR e não WAR. Após inserirmos as informações básicas referentes a configuração do maven, passaremos para Figura 5, onde serão definidas as configurações iniciais do projeto Spring Boot.

Figura 4 – Novo Projeto STS



Fonte: Spring Tool Suite

Figura 5 – Novo Projeto STS Definições



Fonte: Spring Tools Suite

Na segunda tela de configuração do projeto a ser criado, estamos definindo as dependências que serão inseridas no projeto. Nesse caso optamos pelas dependências JPA, MySQL e Web. A partir de ponto podemos afirmar que a nossa aplicação será um projeto Web que utilizará a especificação JPA para fazer a conexão com um BD MySQL através da implementação Hibernate. Ao finalizarmos a

configuração do projeto, clicando no botão “Finish”, será gerado um projeto maven e as dependências configuradas no processo de criação do projeto serão baixadas.

Analisando a Figura 6 temos um pom.xml padrão, com suas definições de estrutura e dependências. Como estamos usando o Spring Boot nesta aplicação, veremos apenas os detalhes que estão diretamente relacionados a estrutura desse tipo de projeto.

Primeiro ponto de interesse é o <artifactId> da tag <parent>, ao atribuir-lhe o valor “spring-boot-starter-parent” está sendo definido que esta aplicação herdará as configurações do Spring Boot e todas as dependências necessárias para que um projeto pronto para ser executado possa ser desenvolvido. Outro detalhe importante é a definição da tag <version> onde está definida a versão do Spring Boot que será utilizada pela aplicação, podemos observar que as dependências “starter” não possuem versão definida, desta forma o maven poderá garantir que o versionamento entre as dependências será sempre compatível uns com os outros.

Figura 6 - pom.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6   <modelVersion>4.0.0</modelVersion>
7   <groupId>br.com.tcc.unimms</groupId>
8   <artifactId>ArtigoService</artifactId>
9   <version>0.0.1-SNAPSHOT</version>
10  <packaging>jar</packaging>
11  <name>ArtigoService</name>
12  <description>Projeto para demonstracao de principios do Spring Boot</description>
13  <parent>
14    <groupId>org.springframework.boot</groupId>
15    <artifactId>spring-boot-starter-parent</artifactId>
16    <version>1.4.2.RELEASE</version>
17    <relativePath><!-- lookup parent from repository -->
18  </parent>
19  <properties>
20  </properties>
21  <dependencies>
22    <dependency>
23      <groupId>org.springframework.boot</groupId>
24      <artifactId>spring-boot-starter</artifactId>
25    </dependency>
26    <dependency>
27      <groupId>org.springframework.boot</groupId>
28      <artifactId>spring-boot-starter-web</artifactId>
29    </dependency>
30    <dependency>
31      <groupId>org.springframework.boot</groupId>
32      <artifactId>spring-boot-devtools</artifactId>
33      <scope>runtime</scope>
34    </dependency>
35    <dependency>
36      <groupId>mysql</groupId>
37      <artifactId>mysql-connector-java</artifactId>
38      <scope>runtime</scope>
39    </dependency>
40    <dependency>
41      <groupId>org.springframework.boot</groupId>
42      <artifactId>spring-boot-starter-test</artifactId>
43      <scope>test</scope>
44    </dependency>
45  </dependencies>
46  <build>
47  </build>
48 </project>

```

Fonte: Spring Tool Suite

Havendo a necessidade de realizarmos configurações personalizadas o Spring Boot aceitará as definições realizadas pelo desenvolvedor. No arquivo `application.properties` definimos a string de conexão, usuário e senha do banco de dados, assim como adicionamos uma configuração JPA para que as tabelas do banco de dados sejam geradas a partir das classes de domínio. Isto pode ser verificado na Figura 7.

Figura 7 - Arquivo de Propriedades

`application.properties`

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/basetcc
2 spring.datasource.username=root
3 spring.datasource.password=123456
4
5 spring.jpa.hibernate.ddl-auto=update
6
```

Fonte: Spring Tool Suite

Na Figura 8 vemos a classe principal do projeto, a anotação `@SpringBootApplication` habilita o funcionamento do Spring Boot, é muito importante que esta classe esteja na raiz do projeto, todas as classes que estiverem em seu pacote será verificadas pelo Spring Boot em busca de configurações e anotações.

Figura 8 – Classe Principal

`ArtigoServiceApplication.java`

```
1 package br.com.tcc;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class ArtigoServiceApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(ArtigoServiceApplication.class, args);
11     }
12 }
13
```

Fonte: Spring Tool Suite

A anotação `@SpringBootApplication` é o equivalente às seguintes anotações do Spring:

- `@Configuration` – Define a classe anotada como um bean de configuração da aplicação.
- `@EnableAutoConfiguration` – Acionar as convenções do Spring, será analisado o classpath da aplicação e as configurações definidas pelo desenvolvedor para definir as convenções a serem utilizadas.
- `@ComponentScan` – Varre o pacote, em que a classe anotada está inserida, e busca componentes, serviços e configurações que estejam definidas.
- `@EnableWebMvc` – Define o projeto como uma Aplicação Web do Spring.

O Spring Boot possibilita o uso de POJO's nas aplicações desenvolvidas, na Figura temos a classe de domínio da aplicação. Este é o POJO referente ao domínio da aplicação, as anotações aqui utilizadas são referentes ao JPA (`@Entity`, `@Id`, `@GeneratedValue` e `@Column`) ou referente ao Jackson (`@JsonInclude` e `@JsonFormat`), framework responsável em converter nossa classe de domínio em um modelo JSON.

Figura 9 – Classe de Domínio

```

Artigo.java

1 package br.com.tcc.dominio;
2
3 import java.util.Date;
4
5 @Entity
6 public class Artigo {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private Long id;
11
12    @JsonInclude(Include.NON_NULL)
13    private String titulo;
14
15    @JsonInclude(Include.NON_NULL)
16    private String instituicao;
17
18    @JsonInclude(Include.NON_NULL)
19    private String area;
20
21    @JsonInclude(Include.NON_NULL)
22    private String autor;
23
24    @Column(uptdatable = false)
25    @JsonFormat(pattern = "dd/MM/yyyy hh:mm:ss")
26    private Date cadastro;
27
28    //Foram ocultados os Getters and setters
29
30 }

```

Fonte: Spring Tool Suite

Conforme vemos na Figura 10, a interface `ArtigoRepository` estende `JpaRepository` e passa como argumentos a classe de domínio (Figura 9) e o tipo de dado do id classe, com este processo já temos o suficiente para realizar a persistência de objetos do tipo informado como argumento, em resumo, temos um CRUD já funcional.

Figura 10 – Repositório

`ArtigoRepository.java`

```
1 package br.com.tcc.repositorio;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7 public interface ArtigoRepository extends JpaRepository<Artigo, Long> {
8
9 }
10
```

Fonte: Spring Tool Suite

Vejamos agora a classe responsável pelas regras de negócio, exibida na Figura 11.

Figura 11 - Serviço

`ArtigoServico.java`

```
1 package br.com.tcc.servico;
2
3 import java.util.Date;
4
5
6
7
8
9
10
11
12
13
14 @Service
15 public class ArtigoServico {
16
17     @Autowired
18     private ArtigoRepository artigoRepository;
19
20     public List<Artigo> listar() {
21         return artigoRepository.findAll();
22     }
23
24     public Artigo buscar(Long id) {
25         Artigo artigo = artigoRepository.findOne(id);
26         if (artigo == null) {
27             throw new RecursoNaoEncontradoException("Artigo não encontrado.");
28         }
29         return artigo;
30     }
31
32     public Artigo salvar(Artigo artigo) {
33         artigo.setId(null);
34         artigo.setCadastro(new Date());
35         return artigoRepository.save(artigo);
36     }
37
38     public void deletar(Long id) {
39         try {
40             artigoRepository.delete(id);
41         } catch (EmptyResultDataAccessException e) {
42             throw new RecursoNaoEncontradoException("Artigo não encontrado.");
43         }
44     }
45
46     public void atualizar(Artigo artigo) {
47         verificarExistenciaArtigo(artigo);
48         artigoRepository.save(artigo);
49     }
50
51     private void verificarExistenciaArtigo(Artigo artigo) {
52         buscar(artigo.getId());
53     }
54 }
55
```

Fonte: Spring Tool Suite

Através da anotação `@Service` estamos definindo a classe responsável pelas regras de negócio da aplicação. E a anotação `@Autowired` informa ao Spring que o objeto `artigoRepository` deverá ser injetado pelo Spring.

Figura 12 – Manipulador de Exceções

ManipuladorExcecaoRecursos.java

```

1 package br.com.tcc.manipulador;
2
3 import javax.servlet.http.HttpServletRequest;
12
13 @ControllerAdvice
14 public class ManipuladorExcecaoRecursos {
15
16     @ExceptionHandler(RecursoNaoEncontradoException.class)
17     public ResponseEntity<ErroDetalhe>
18         manipuladorExcecaoRecursos(RecursoNaoEncontradoException e,
19         HttpServletRequest request) {
20
21         ErroDetalhe erro = new ErroDetalhe();
22
23         erro.setStatus(404);
24         erro.setTitulo("Recurso não encontrado");
25         erro.setMensagem("Verifique os dados informados.");
26         erro.setTimestamp(System.currentTimeMillis());
27
28         return ResponseEntity.status(HttpStatus.NOT_FOUND).body(erro);
29     }
30 }

```

Fonte: Spring Tool Suite

Temos, também, uma classe para realizar a interceptação de exceções, observe a Figura 12, ela possui a anotação `@ControllerAdvice`, ao utilizá-la estamos definindo um mecanismo de captura de exceções onde elas ocorrerem, neste caso específico em qualquer parte do programa em que for levantada a exceção `RecursoNaoEncontradoException`, a classe `Manipulador` deverá assumir o tratamento desta exceção através do método anotado com `@ExceptionHandler`, podemos criar outros métodos com essa anotação e as capturarmos onde elas ocorrerem.

E, finalmente, temos a classe `ArtigoRegcurso`, Figura 13, que é responsável por disponibilizar os recursos da aplicação através de URL's. Observe que ela possui a anotação `@RestController`, que já foi comentada ao descrevermos a Figura 1

referente ao arquivo `app.groovy`, temos, também, a anotação `@RequestMapping("/artigo")`, o que nos indica que os métodos que disponibilizarem endereços de acesso ao recurso artigo a partir da classe `ArtigoRecurso` serão acessíveis pelo endereço `http://<ipdoservidor>:8080/artigo`. Vamos escolher um método da classe `ArtigoRecurso` para ilustrar o comportamento, vejamos o método `listar()`, ele está anotado com `@RequestMapping(method = RequestMethod.GET)`, ao definirmos o método desta forma disponibilizando a listagem de artigos através do endereço raiz do recurso e estamos informando ao Spring que requisições do tipo GET a este endereço deverão ser por este método. Observe também que o método `salvar()` estará acessível no mesmo endereço, entretanto, este último deverá tratar apenas requisições do tipo POST e espera receber um objeto do tipo artigo como parâmetro do corpo da requisição, isso é perceptível pela anotação `@RequestBody`.

Figura 13 – Artigo Recurso

```

ArtigoRecurso.java

1 package br.com.tcc.recurso;
2
3 import java.net.URI;
4
5 @RestController
6 @RequestMapping("/artigo")
7 public class ArtigoRecurso {
8
9     @Autowired
10    private ArtigoServico artigoServico;
11
12    @RequestMapping(method = RequestMethod.GET)
13    public ResponseEntity<List<Artigo>> listar() {
14        return ResponseEntity.status(HttpStatus.OK).body(artigoServico.listar());
15    }
16
17    @RequestMapping(method = RequestMethod.POST)
18    public ResponseEntity<Void> salvar(@RequestBody Artigo artigo) {
19
20        artigo = artigoServico.salvar(artigo);
21        URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
22            .buildAndExpand(artigo.getId()).toUri();
23        return ResponseEntity.created(uri).build();
24    }
25
26    @RequestMapping(value =("/{id}", method = RequestMethod.GET)
27    public ResponseEntity<?> buscar(@PathVariable("id") Long id) {
28
29        Artigo artigo = artigoServico.buscar(id);
30        return ResponseEntity.status(HttpStatus.OK).body(artigo);
31    }
32
33    @RequestMapping(value =("/{id}", method = RequestMethod.DELETE)
34    public ResponseEntity<Void> deletar(@PathVariable("id") Long id) {
35
36        artigoServico.deletar(id);
37        return ResponseEntity.noContent().build();
38    }
39
40    @RequestMapping(value =("/{id}", method = RequestMethod.PUT)
41    public ResponseEntity<Void> atualizar(@RequestBody Artigo artigo, @PathVariable("id")
42    Long id) {
43
44        artigo.setId(id);
45        artigoServico.atualizar(artigo);
46        return ResponseEntity.noContent().build();
47    }
48 }
49
50
51

```

Fonte: Spring Tool Suite

CONCLUSÃO

Nos testes realizados com a criação de projetos utilizando o Spring Boot foi possível perceber a facilidade em iniciar novos projetos, o conceito de convenção sobre configuração fornece um grau de liberdade sem precedentes para o desenvolvedor, simplesmente abstraímos qualquer preocupação com configuração, a menos que fosse estritamente necessário e sabíamos que o Spring Boot se encarregaria de fornecer as configurações mais apropriadas para infraestrutura do projeto, incluindo o servidor Tomcat que foi embarcado na aplicação. Este cenário permitiu que o esforço fosse direcionado às regras de negócios.

Contudo, voltando-se para o mundo de Microsserviços percebe-se que o Spring Boot é um potencializador de desempenho para equipes de desenvolvimento, se para criarmos um projeto simples, disponibilizando apenas um recurso, o Spring Boot permitiu o desenvolvimento com qualidade e confiabilidade na estrutura e configuração, consideramos que havendo a necessidade de produzir diversos microsserviços o efeito será o mesmo, embora em escala maior.

Este trabalho não julgar encerrar em si o assunto, mas se propõe a abrir espaço para aqueles que o tomando como ponto de partida desejem complementá-lo, os campos são vastos para esta possível obra. Dentre os possíveis caminhos a serem adotados um deveras interessante seria aplicar o uso do Spring Cloud sobre os conceitos aqui apresentados e gerar apenas gerar um novo artefato mas distribuí-lo e vê-lo funcionando em prática.

REFERÊNCIA BIBLIOGRÁFICA

RV, Rajesh. **Spring Microservices**. 1. ed. Birmingham B3 2PB, UK: Packt Publishing, 2016.

GUTIERREZ, Felipe. **Introducing Spring Framework**. 1. ed. New York, NY: Apress, 2014.

GUTIERREZ, Felipe. **Pro Spring Boot**. 1. ed. New York, NY: Apress, 2016.

WALLS, Craig. **Spring In Action**. 4. ed. Shelter Island, NY: Manning, 2015.

SOUZA, Ana Carolina Carvalho de Paula. **Estudo do desenvolvimento de uma aplicação web utilizando o Spring Framework**. Maringá: 2011. 53 f. Monografia (Especialização em Desenvolvimento de Sistemas para Web) - Universidade Estadual de Maringá, Maringá, 2011.

WEISSMANN, Henrique Lobo. **Vire o jogo com Spring Framework**. São Paulo: Casa do Código, 2012.

FOWLER, Martin. LEWIS, James. **Microservices**, Disponível em: <<http://martinfowler.com/articles/microservices.html>>. Acesso em: 26 Set. 2016.

OKI, Willian. **Dominando o Spring Boot**. Disponível em: <<http://www.devmedia.com.br/dominando-o-spring-boot/33654>>. Acessado em: 30 Out 2016.

JR, Normandes. **Primeiros Passos com Spring MVC**, 1. ed. São Paulo, SP: AlgaWorks, 2016.

FELIPE, Eduardo. **Desenvolvendo uma Aplicação Web com Spring Boot e Spring MVC**. Disponível em: <<http://www.devmedia.com.br/desenvolvendo-uma-aplicacao-web-com-spring-boot-e-spring-mvc/34122>>. Acessado em: 25/09/2016.

Spring Boot. Disponível em: <<http://projects.spring.io/spring-boot/>>. Acessado em: 17/11/2016

Apache Maven Project. Disponível em: <<http://maven.apache.org/>>. Acessado em: 02/12/2016.

Gradle. Disponível em: <<https://gradle.org/getting-started-gradle/>>. Acessado em: 02/12/2016.