README.md

SpringBoot

1. O que o SpringBoot?

O Spring Boot é um sub-projeto da Spring que visa facilitar o processo de configuração e deploy de suas aplicações.

Ele transforma o processo tedioso e complexo de se criar uma aplicação java para web, por exemplo, em uma experiência bastante agradável, gerando aplicações enxutas, rápidas e robustas. E mais, se você quiser ele embute o servidor (tomcat, jetty ou outro) no seu `Über.jar` e a sua aplicação está pronta para ser distribuída!

É mágica?

Na verdade, parte da mágica, está no fato dos projetos do SpringBoot conterem, sempre, um projeto pai (`parent`), que é um projeto onde estão todas as dependências necessárias para o SpringBoot, além de outras configurações padrões para o seu projeto `pom.xml` . Observe a tag `<parent>` no arquivo `pom.xml` abaixo:

```
<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>1.5.17.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -->
</parent>
```

Veja que o seu `pom.xml` herda de um pai (`parent`).

O restante da magia fica por conta do fato de o SpringBoot usar toda a tecnologia que está por traz desse framework, como por exemplo: injecção de dependência, inversão de controle etc. Além disso, o Spring possui no seu ecossistema diversos outros frameworks para acesso a dados, segurança, RestFull etc. Isso faz com que a sua aplicação seja parecida com um jogo de montar peças que se encaixam. Basta que você diga para ele quais módulos você deseja utilizar (WEB, Template, Persistência, Segurança, etc.) que ele vai reconhecer e configurar para você. Simples não?

Configuração ou Convenção? Como o SpringBoot trabalha?

O Spring Boot usa a chamada `convention over configuration` , isto é, se você não especificar uma configuração própria, ele irá usar o que está convencionado.

Eu explico melhor:

Imagine que você queira especificar as propriedades do seu projeto em um arquivo denominado `application.properties` , o que, sem dúvida, é uma boa prática.

Se você não disser em que local está esse arquivo, o SpringBoot irá procurar por uma configuração (`*.properties`) na pasta `src/main/resources` , que é o local que **foi convencionado para isso**. Entendeu?

::: 👍 Boa Prática :::

Use um arquivo de configuração, por exemplo: `application.properties` no seu projeto contendo as principais configurações do seu banco de dados, contexto da aplicaçãao etc. Essa prática torna o seu projeto mais limpo e centraliza as configurações em um só local

Apesar do SpringBoot, através da convenção, já deixar tudo configurado, nada impede que você crie as suas customizações caso elas sejam necessárias.

O nosso foco deve estar nas regras do negócio

::: 📌 Importante :::

> O que é mais importante, as regras de suas aplicações ou a tecnologia para fazê-las funcionar? Eu penso que são as regras, são elas que dão a "inteligência" ao seu negócio. Nessa linha de raciocínio o SpringBoot lida com as configurações e nos deixa mais livres para pensarmos nas regras de negócio da nossa aplicação.

Bem, chega de blá, blá e vamos construir uma aplicação usando o SpringBoot, afinal "um tutorial vale mais do que mil palavras".

2. Construindo uma aplicação RestFul com o SpringBoot

::: 🚶 Passo a passo :::

Vamos constuir a nossa aplica em 4 etapas+1, eu explico melhor: a nossa aplicação será um estudo de caso que será construído em 4 versões mais 1 conjunto de testes de integração.

Na versão V1 nós vamos contextualizar o RestFull frente à outras tecnologias usadas na computação distribuída, realizar um configurar mínima necessárias para uma aplicação REST usando o Spring e por fim iremos construir uma aplicação RestFul com apenas um método - uma aplicação "Hello World".

Na versão V2 iremos mostrar como trabalhar com registros (logs), demonstrar as vantagens de se utilizar YAML para o arquivo de configuração no lugar de arquivos de propriedades, criar uma classe para persistir os Municípios do estudo de caso, utilizar o Liquibase para manter o banco de dados, utilizar uma interface @Repository para os Municípios, utilizar uma classe de serviços e, por fim, criar um método Get para exibir todos os Municípios em formato JSON.

Na versão V3 iremos criar uma classe persistente para Departamentos, atualizar o Liquibase com as novas tabelas e constraintes, criar uma interface @Repository para Departamento, utilizar uma classe de serviços para Departamento, complementar a classe MunicipioResource com outros métodos do CRUD, criar a classe DepartamentoResource e fazer tratamento de erro bem como utilizar classes úteis em REST.

Na última versão (V4) será quase que totalmente implementada pelo estudante que deverá criar uma classe persistente, o prepositório, classe de serviço e métodos REST para o Empregado.

Então vamos lá. Passe para a implementação V1 do nosso Estudo de Caso.