# Semana do Desenvolvedor Java & Spring

Organizador: TreinaWeb

Data: De 03 a 16/07/2019

## AULA 01 – As configurações básicas do Spring

**Start.spring.io:** Configuração de todo o projeto e suas dependências;

Importar como projeto Maven normalmente;

Não é necessário colocar o prefixo "controller" nas classes de controller. É apenas uma convenção, tendo em vista que o Spring reconhece o que é um controller pela anotação @Controller;

**HttpActions:** Métodos HTTP (GET, POST, PUT, DELETE);

Página que o Thymeleaf retornar precisa estar dentro de Templates;

O Spring gera um servidor automático (Tomcat), no exemplo da Treina Web foram feitas as configurações de rodagem dentro de Run Configurations > Maven Build;

Toda a compilação de servidor, Maven, Eclipse e tudo o mais é feita pelo Spring Boot;

Ao criar uma dependência no pom.xml:

* groupId: Nome da empresa que gerencia a dependência
* artifactId: Biblioteca a ser importada
* version: Auto-explicativo

Todas as bibliotecas, até mesmo de front-end como jQuery e Bootstrap, podem ser importadas pelo Maven via pom.xml.

## AULA 02 – JPA, Spring Data e Hinernate

**Trigramação:** convenção utilizada em alguns locais para nomear tabelas que consiste em colocar três letras como prefixo que representam de maneira geral a entidade das tabelas;

Utilizar a notação @Entity da JPA, em razão da independência entre Hibernate e outros frameworks;

A validação na JPA é feita campo por campo;

**JPARepository:** Recebe o modelo e o tipo do atributo ID. Estendendo isso na interface, toda a interação é feita pelo Spring;

**@GetMapping:** Designa o método a usar o método GET para a ação desejada;

**@Autowired:** Injeção de dependências, sem precisar instanciar o objeto. Tudo feito pelo Spring;

**Fragmentos:** Pedaços de página reaproveitados em outras páginas;

**th:fragment:** Tag do Thymeleaf que permite o reaproveitamento em outras páginas;

**th:each:** Tag do Thymeleaf para o forEach;

**th:text:** Tag do Thymeleaf para inserir dados, podendo capturar EL para isso;

**Helper:** Recurso do Thymeleaf acionado por # (#dates.format());

**th:field:** Tag para fazer o Binding do campo do formulário para o campo da tabela;

**@NotNull:** Notação para validar o campo diretamente no modelo;

**th:if:** Tag do Thymeleaf para um if.

## AULA 03 – Spring Security e Autenticação

A declaração na interface do método de busca faz com que o Spring Data faça a busca automaticamente (Ex: Usuario findByEmail);

**ServicePattern:** Classe para validações de negócio;

**@Service:** Notação Spring para identificar que é uma classe de serviço;

**BCryptPasswordEncoder:** Criptografia da senha;

**@Configuration:** Notação Spring para designar uma classe de configuração;

**WebSecurityConfigurerAdapter:** Classe de configuração do Spring Security.

## AULA 04 – Deploy da aplicação para o Heroku e como está o mercado Java

Adição de FK para que cada usuário tenha uma ou mais tarefas atreladas a ele;

**@OneToMany:** Notação JPA para designar que determinado dado compõe a relação 1-N;

**@ManyToOne:** Notação JPA para designar que determinado dado compõe a relação N-1;

**mappedBy:** Propriedade que faz a relação entre as relações citadas anteriormente;

**fetch:** Propriedade que ordena o tipo de carregamento dos dados (EAGER/LAZY);

**@JoinColumn:** Notação JPA para indicar qual campo é utilizado como relacionamento entre a duas tarefas;

**@Query:** Notação do Spring Data para consultas personalizadas (neste caso, foi utilizado dentro do RepositorioTarefa);

**JPQL (Java Persistence Query Language):** Linguagem Java para consultas, bem similar ao SQL padrão;

**:emailUsuario:** Exemplo de parâmetro JPQL;

**@Param:** Notação do Spring Data para definir um nome de parâmetro a ser utilizado pela JPQL;

**Heroku:** PaaS (Plattform as a Service) de acordo com a linguagem;

Heroku fornece instâncias de banco gratuitas, porém não MySQL. É moldado em cima do Postgree;

Adicionar no pom.xml a dependência do driver Postgree e modificar o application.properties, designando diretamente o nome do driver Postgree;

Comandos heroku:

* heroku login – Efetua o login com o usuário criado no Heroku;
* heroku create - Gera uma aplicação no Heroku;
* heroku addons:create - Cria um add-on no Heroku;
* heroku addons:create heroku-postgresql - Cria uma instância Postgree no Heroku.

Heroku usa o git por debaixo dos panos para fazer o deploy. Por isso, ele necessita do git;

O Heroku trata todos os envios como uma aplicação (app);

**Git push heroku master:** Envio do Git local criado pelo Heroku para a master do ambiente remoto;

**https://still-journey-11649.herokuapp.com/** - Link no Heroku da aplicação.