**14. JPA Entities**

JPA = Java Persistance API

In JPA we will use POJO ( Plain Old Java Objects)

The packages that make reference to JPA entities are named “domain” or “model”

JPA requires none Constructor

Spring Data JPA it is made to work with Hibernate

Hibernate it is generating DDL SQL statement

**MVC** = it was created in 70 by Xerox

Keep the controler light! It should be like a traffic cop! No database connections in it.

Annotate Controller Class with @Controller. This will register the class as a Spring Bean and as a Controller in Spring MVC

For maping methods to http request paths use @RequestMapping

**Thymeleaf** = Java Template Engine, este o alternativa la JSP (Java Server Pages)

= acesta produce: XML,XHTML,HTML5

= nu este un web browser

= scopul lui este sa ia un template, si sa il converteasca catre HTML

= **thymeleaf templates** practic sunt niste documnete HTML

**Spring Boot** = if there it is no Spring Web context, the spring will start and after that stop.

**Toate aceste stereotipuri(@Controller, @Service) nu difera una de cealalta.** Singurul lucru este este ca acestea ii spun spring-ului ca sunt **Spring managed component**.

**@Autowired = iti leaga un field sa zicem de un bean.**

**@Qualifier = spune care Bean alege**

**@Primary = daca nu ai pus niciun Qualifier, foloseste automat Bean-ul care are stereotipul @Primary**

**@Profile =** o caracteristica din Spring Framework foarte puternica. Te ajuta sa iti controlezi aplicaia in medii de rulare diferite.

**Implementation Naming =**

* < Interface Name > + Impl  **for a class that implements just one interface**

**SpringBoot** = este un Spring Framework care are niste configurari!

**!!!Spring Boot version it is configured to work with a certain Spring Framework version.**

**Maven** = proprietatile Maven sunt mostenite de la un Spring Boot Parent POM.

= cand e posibil, nu specifica versiunile in POM.

**Starters =** top level dependencies for **popular** Java libraries

**Ex:** Starter ‘spring-boot-starter-data-jpa’

**@SpringBootApplication = @Configuration**(declares class as Spring Configuration) **+** @**EnableAutoConfiguration**(enables auto configuration) + **@ComponentScan** (scan for components in curent package and all child packages)

**Spring Bean Scopes:**

**Singleton** = doar o singura instanta de **bean** este creata in containerul IoC.

**Prototype** = o noua instanta este creata de fiecare data cand un **bean** este chemat

**Request =**  o singura instanta per http request

**Session =** o singura instanta per sesiune http

**Application = bean-ul** tine cont de ciclul de viata al lui ServletContext

**Websocket** = vizeaza o singura definite de bean pentru un ciclu de viata al unui WebSocket!

**Custom Scope** = Spring Scopes sunt mostenibile, si poti extinde si sa iti definesti scopul tau prin implementarea lui Spring ‘Scope’ interface

By default scope-ul fiecarui bean este singleton.

**Convention** = definirea lui startup data si proceselor de startup se fac in pachetul de bootstrap.

**!** Daca avem un o clasa adnotata cu **@Component** si implementam **CommandLineRunner**, va trebuie sa implementam metoda run. Tot ce vom scrie acolo se va intampla dupa ce **@SpringBootB**  este initializat.

${} = Spring expression language

YAML = yet another markup language

**LiveReload** = este o tehnologie care declanseaza automat un refresh de browser cand resursele sunt schimbate.

**Spring Boot Developer Tools** include un **LiveReload**.

**JHipster** = code generator project, built around AngularJS and SpringBoot

**Snake case** -> ex: @Column(name = “first\_name”)

@Column(name = “last\_name”)

* First name si last name sunt “snake case”