

# Beispiele zu JPA, Spring, Spring Boot und Spring Rest

## JPA standalone

### ▪ **sk.train.x11\_01\_JPA\_Solution**

Zu gegebener Entity "Employee" mit entsprechender Tabelle in der H2-Datenbank, ist ein entsprechendes rudimentäres Repository "EmpService" erstellt worden. Der JPA-Layer wird via Starter-Klasse initialisiert und das Repository genutzt.

### ▪ **sk.train.x11\_02\_JPA\_Solution**

(Diesmal sind zwei Entities gegeben, die den Tabellen "Employees" und "Departments" mitsamt den Schlüssel-Fremdschlüssel-Beziehungen gegeben. Es ist ein entsprechendes rudimentäres Repository "EmpService" vorhanden, welches auch auf Departments zugreift.

### ▪ **sk.train.x11\_03\_JPA\_Solution**

Das Repository wurde um entsprechende Methoden unter Nutzung von JPQL bzw. nativen Queries ergänzt.

## JPA mit Spring bzw. Spring Boot

### ▪ **sk.train.x11\_05\_JPA\_Spring\_Solution**

Auf Basis von sk.train.x11\_01\_JPA\_Solution umgestellt:  
JPA durch Spring gemanaged + deklarative Transaktionsunterstützung

### ▪ **sk.train.x11\_05\_JPA\_SpringBoot\_Solution**

JPA durch Spring gemanaged + deklarative Transaktionsunterstützung, aber jetzt via SpringBoot realisiert: wir brauchen keine Konfig mehr für diesen Fall.  
Nur noch die DB-Properties für Spring-Boot.

### ▪ **sk.train.x11\_07\_JPA\_Spring\_SpringBoot\_Solution**

JPA durch Spring Boot gemanaged + deklarative Transaktionsunterstützung + Repository-Generierung mit individuellen Ergänzungen.

- **sk.train.x11\_07\_JPA\_Spring\_SpringBoot\_WithTest\_Solution**

Zusätzlich mit klassischem Test via Junit mittels "@SpringBootTest".

- **sk.train.x11\_07\_JPA\_Spring\_SpringBoot\_WithTest\_Solution**

Statt kompletter Test jetzt nur Slice-Test für den JPA-Layer.

Anmerkung: Sofern wir auf "@DataJpaTest" wechseln, um nur den Repository-Layer zu testen, muss der "CommandLineRunner" aus der Konfig entfernt werden, da dieser im Starter enthalten ist und versucht wird, diesen zu erzeugen. Das scheitert aber, da er nicht zu den für den Test notwendigen Komponenten gehört (hier steht sich Spring ein wenig selbst im Weg!).

## **JPA mit Spring Boot und REST Frontend**

- **sk.train.x12\_01\_JPA\_Rest\_Boot\_Solution**

Das bisherige sk.train.x11\_07\_JPA\_Spring\_SpringBoot\_Solution mit einfachem Rest-Frontend, welches die CRUD-Operationen abdeckt und JSON liefert.

- **sk.train.x12\_02\_JPA\_Rest\_Boot\_Solution**

Rest mit JPA, Spring Boot und entsprechender Fehlerbehandlung bzw. Statuscodes.

- **sk.train.x12\_03\_JPA\_Rest\_Boot\_Solution**

Rest mit JPA, Spring Boot und XML-MessageFormat mit Jackson.  
Dazu sind weitere Bibliotheken im ClassPath nötig.

- **sk.train.x12\_03\_JPA\_Rest\_Boot\_Solution3\_Swagger**

Vorheriges Beispiel ergänzt um Open-Api-Generierung inklusive Swagger-UI:

<http://localhost:8080/v3/api-docs>

default-URL für Open-Api-Doc

<http://localhost:8080/swagger-ui/index.html>

default-URL für Swagger-UI

- **sk.train.x12\_06\_JPA\_Rest\_Client\_Solution**

Java-Client zum Rest-Service in der Version sk.train.x12\_01\_JPA\_Rest\_Boot\_Solution via Spring RestTemplate.