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_08_JPA_SpringBoot_RepoTest_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 ensprechender 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_03_JPA_Rest_Boot_Solution3_Swagger_Actuator

Vorheriges Beispiel ergänzt um Actuator-Einbindung zur Überwachung der Anwendung (inklusive eigener Infos über Info-Endpunkt):

http://localhost:8080/v3/api-docs	default-URL für Open-Api-Doc
http://localhost:8080/swagger-ui/index.html	default-URL für Swagger-UI
http://localhost:8080/actuator	default-URL für Actuator-Endpunkte

- ## 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.