Webprojekt
Umsetzung JPA, Hibernate mit Entity Manger, Javax

Einleitung:

Bisher haben Sie ein Java Projekt mit Maven als Basis verwendet.

Wie sieht es bei einer Webapplikation mit SpringBoot MVC aus?

Das Spring Framework im Hintergrund unterstützt die Verwendung von *Dependency Incjected Objecs*.

Im ViewController wird das Attribut des Entity Manager mit *@PersistenceContext* annotiert, damit verwaltet das Framework die Instanz des Entity Manager als *dependency on a container-managed EntityManager and its associated persistence context*.

Die Transaktion werden von Javax übernommen, dazu werden die Methoden, in denen eine Transaktion durchgeführt wird mit *@Transactional* annotiert.

Das Sping Framework lädt auch die Konfiguration, die in der Datei *src/main/resources/application.properties* steht.

Sie nutzen die Eigenschaften von JPA mit Beziehungen umzugehen.

- ⇨ Spring Boot MVC Projekt mit Maven, Verwendung von JPA, Hibernate EntityManager.
- ⇨ In dieser Variante wird der EntityManager von Javax unterstützt.

Ziele:

- ⇨ Sie können Entitäten aus einer Datenbank über das Persistenz-Framework und das Interface JPA mit Objekten der Applikation verwalten.
- ⇨ Sie setzen den EntityManager und Javax ein.
- ⇨ Sie setzen die Konfiguration für ein Spring Boot MVC Projekt um.

Inhaltsverzeichnis

Inspiration zu Spring Boot mit Entity Manager:

Inspiration war diese Seite, allerdings muss man einiges adaptieren.

https://www.baeldung.com/hibernate-entitymanager

Die folgenden Schritte zeige grob, wie Sie vorgehen, um das Beispiel von Employee und Department abzubilden.

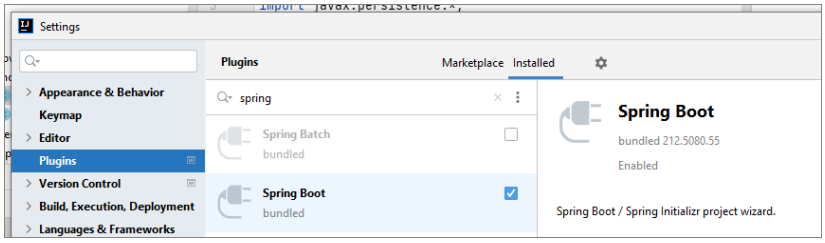
Aufgabe Spring Boot Projekt erstellen:

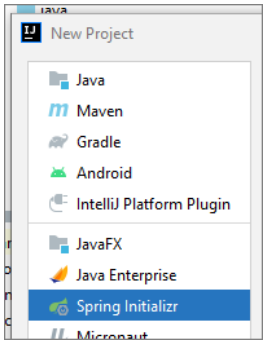
Die Basis ist ein Spring Boot MVC Projekt.

Entweder starten Sie über:

https://start.spring.io/

Oder direkt in IntelliJ, wo Sie allenfalls die Spring Boot Plugins aktivieren müssen.

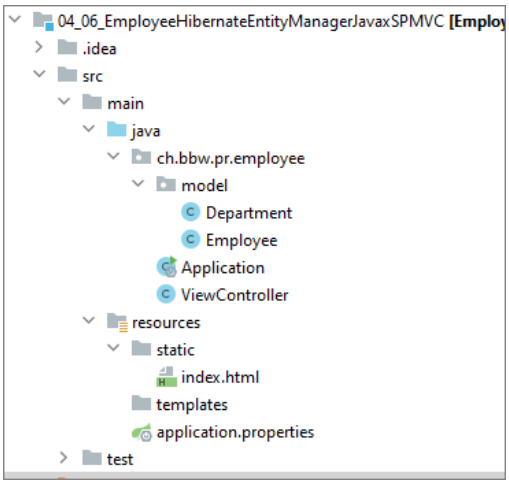




Ausgewählte Dependencies:

- Spring Web
- Thymeleaf
- Spring Data JPA
- MySQL Driver
- Spring Boot Dev Tools
- Spring Boot Actuator

Aufgabe Projektinhalte ergänzen / List implementieren:



Das Model mit den beiden Klassen *Department* und *Employee* übernehmen Sie aus den vorangehenden Projekten. An den JPA Annotierungen ändert sich nichts.

Die Klasse *Application* wird beim Erstellen des Projektes generiert.

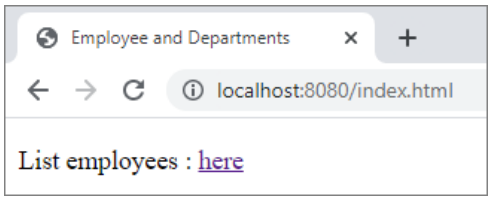
Die ViewControllerklasse reagiert auf die Inputs der View. Die Klasse wird unten beschrieben.

In diesem Projekt ist nur eine sehr einfache View als *index.html* implementiert.

Die Datei *application.properties* enthält die Konfiguration.

index.html

Vorerst ist die View sehr einfach gehalten. Über eine *href="/list"* wird die entsprechende Methode im *ViewController* aufgerufen.



ViewController.java

```
@Controller
public class ViewController {
    @PersistenceContext
    EntityManager entitymanager;

    @GetMapping("/list")
    @Transactional
    public String list() {
        System.out.println("Controller.list");

    List<?> employees = entitymanager.createNamedQuery("Employee.findAll")
            .getResultList();
    System.out.println("Controller.list: " + employees);

    return "redirect:/index.html";
}
```

Im ViewController wird das Attribut des Entity Manager mit *@PersistenceContext* annotiert, damit verwaltet das Framework die Instanz des Entity Manager als *dependency on a container-managed EntityManager and its associated persistence context*.

Die Transaktion werden von Javax übernommen, dazu werden die Methoden, in denen eine Transaktion durchgeführt wird mit *@Transactional* annotiert.

Keine DAO oder Persistence Klassen?

In der gezeigten Lösung gibt es keine DAO oder Persistence Klassen.

Es kann durchaus diskutiert werden, ob die Zugriffe auf die Datenbank weiter gekapselt werden sollen.

Ich habe mich für den Weg «ohne» entschieden, finde den Zugriff über den EntityManager mit nur einer Zeile lesbar.

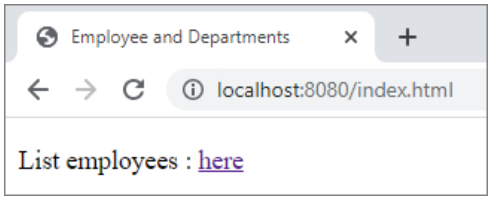
Application.properties

application.properties

```
spring.datasource.url=jdbc:mysql://localhost/employeedepartment
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Resultat:

Aufruf über index.html:



Anzeige in der Console:

```
Controller.list
Controller.list: [Employee{id=3, firstname='hans', lastname='muster', department=null},
 Employee{id=4, firstname='paula', lastname='kuster', department=Department{id=1,
 description='sales'}}]
```

Hilfe zu Table 'DBNAME.hibernate_sequence' doesn't exist

Ich bekam die Fehlermeldung:

Table 'DBNAME.hibernate_sequence' doesn't exist

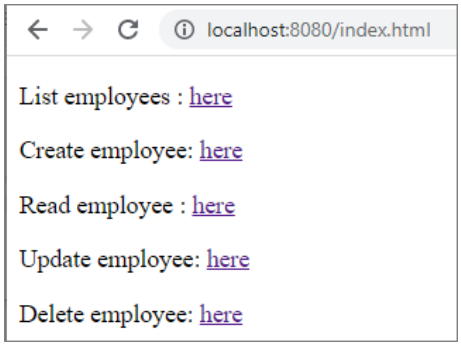
Und fand hier Hilfe:

https://stackoverflow.com/questions/49813666/table-dbname-hibernate-sequence-doesnt-exist

Aufgabe CRUD ergänzen:

Analog zur ersten Methode im ViewController implementieren Sie das vollständige CRUD für Employee mit weiteren Links in der View und Methoden im ViewController.

```
@GetMapping("/list")
@Transactional
public String list() {
...
```




```
Controller.list
Controller.list: [Employee{id=3, firstname='hans', lastname='muster', department=null}, Employee{id=4,
 firstname='paula', lastname='kuster', department=Department{id=1, description='sales'}}, Employee{id=11,
 firstname='Lucky', lastname='Lucke', department=Department{id=2, description='development'}}]
Controller.create
Controller.list
Controller.list: [Employee{id=3, firstname='hans', lastname='muster', department=null}, Employee{id=4,
 firstname='paula', lastname='kuster', department=Department{id=1, description='sales'}}, Employee{id=11,
 firstname='Lucky', lastname='Lucke', department=Department{id=2, description='development'}}, Employee{id=12,
 firstname='Lucky', lastname='Lucke', department=Department{id=2, description='development'}}]
Controller.read
Controller.read Employee{id=12, firstname='Lucky', lastname='Lucke', department=Department{id=2,
 description='development'}}
Controller.update
Controller.read
Controller.read Employee{id=12, firstname='Karl', lastname='Lucke', department=Department{id=2,
 description='development'}}
Controller.delete
Controller.list
Controller.list: [Employee{id=3, firstname='hans', lastname='muster', department=null}, Employee{id=4,
 firstname='paula', lastname='kuster', department=Department{id=1, description='sales'}}, Employee{id=11,
 firstname='Lucky', lastname='Lucke', department=Department{id=2, description='development'}}]
```