Webprojekt
Umsetzung JPA, Hibernate
CrudRepository Interface

Einleitung:

Unter Spring Boot gibt es neben der Möglichkeit über den *EntityManager* Daten in einer Datenbank zu persistieren, auch noch eine andere Möglichkeit.

Spring Data Repositories abstrahieren den Zugriff verringern so den *boilerplate* code, der für eine Implementierung notwendig wird.

Sie nutzen die Eigenschaften von JPA mit Beziehungen umzugehen.

- ⇨ Spring Boot MVC Projekt mit Maven, Verwendung von JPA.
- ⇨ In dieser Variante wird das CrudRepository verwendet.

Ziele:

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

Inhaltsverzeichnis

Beispiel zu Spring Boot mit CrudRepository:

Es gibt verschiedene Spring Data Repositories:

- *CrudRepository:* unterstützt die CRUD-Funktionen.
- *PagingAndSortingRepository*: bietet Methoden zum Paginieren und Sortieren von Datensätzen.
- *JpaRepository:* stellt einige JPA-bezogene Methoden bereit, z. B. das Leeren des Persistenzkontexts und das Löschen von Datensätzen in einem Stapel.

Was nun einfacher ist, hängt von der Anwendung ab.

https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html

Ein Beispiel zur Anwendung zeigt diese Seite, allerdings muss man einiges adaptieren.

https://zetcode.com/springboot/crudrepository/

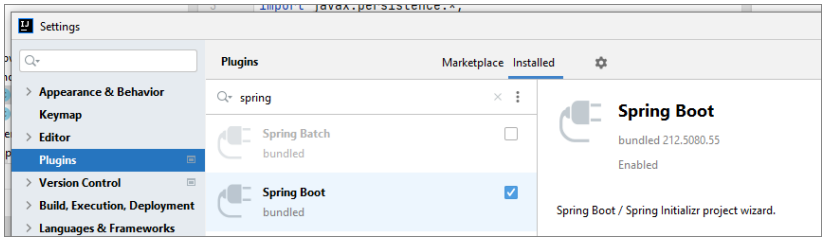
Aufgabe Spring Boot Projekt erstellen:

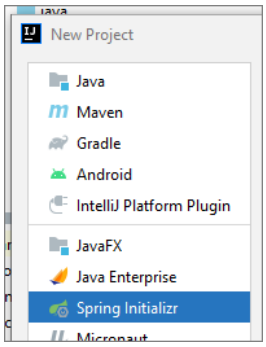
Die Basis ist ein Spring Boot MVC Projekt.

Entweder starten Sie über:

https://start.spring.io/

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

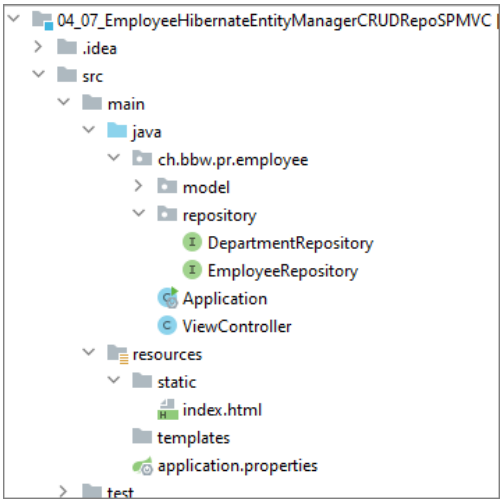




Ausgewählte Dependencies:

- 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.

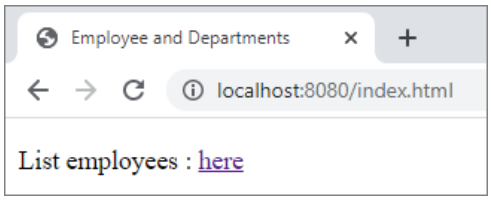
Die ViewControllerklasse reagiert auf die Inputs der View. Die Klasse wird unten beschrieben.

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.



Controller.java

```
@Controller
public class ViewController {
    @Autowired
    private EmployeeRepository employeeRepo;
    @Autowired
    private DepartmentRepository departmentRepo;

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

          Iterable<?> employees = employeeRepo.findAll();
          employees.forEach(e -> System.out.println(e));

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

Im ViewController wird das EmployeeRepository mit *@Autowired* annotiert, damit verwaltet das Framework die Instanz des EmployeeRepository als *dependency*.

Beim Zugriff auf die Datensätze mittels findAll erhält man eine Iterable zurück.
Im Beispiel werden alle Elemente auf die Console ausgegeben.

Für den Zugriff stellt das Repository einige Methoden bereit.
ZBsp:

- findAll
- findById
- save auch für merge zu verwenden
- delete
- …

EmployeeRepository.java

Die Klasse *EmployeeRepository* ist «nur» ein Interface.

Die zugehörige Entitätsklasse *Employee* und der Datentyp des Schlüssels müssen bei den Generics angegeben werden.

```
                                                          //Klasse, id-Typ
public interface EmployeeRepository extends CrudRepository<Employee, Integer>{
}
```

Sofern Sie «nur» die Standardmethoden des CrudRepository verwenden, müssen Sie keine Methoden implementiert.

Das Interface ist also denkbar einfach.

Keine DAO oder Persistence Klassen?

In der gezeigten Lösung gibt es keine DAO oder Persistence Klassen um das CrudRepository zu kapseln.

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 direkt über das CrudRepository für dieses Beispiel zumutbar

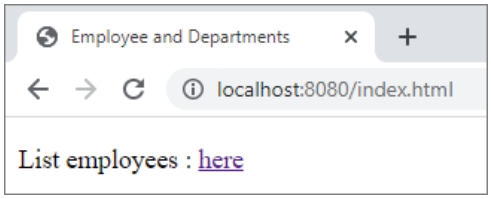
Application.properties

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
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:

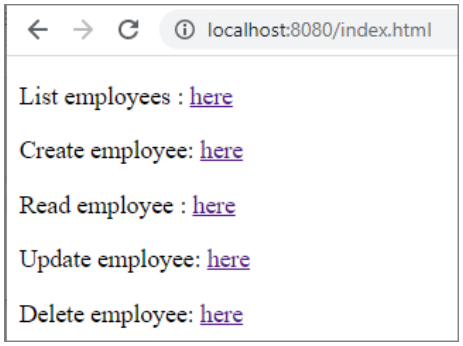
Analog zur ersten Methode im ViewController implementieren Sie das vollständige CRUD für Employee mit weiteren Links in der View und Methoden im ViewController.

Beispiel für das Hinzufügen eines Employee, ohne ihm ein Department zuzuweisen.

```
@GetMapping("/create")
public String create() {
   System.out.println("Controller.create");

   Employee employee = new Employee();
   employee.setFirstname("Lucky");
   employee.setLastname("Lucke");
   employeeRepo.save(employee);

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




```
Controller.list
Employee{id=3, firstname='hans', lastname='muster', department=null}
Employee{id=4, firstname='paula', lastname='kuster', department=Department{id=1, description='sales'}}
Controller.create
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=15, firstname='Lucky', lastname='Lucke', department=Department{id=2, description='development'}}
Controller.read
Controller.read Employee{id=15, firstname='Lucky', lastname='Lucke', department=Department{id=2, description='development'}}
Controller.update
Controller.read
Controller.read Employee{id=15, firstname='Karl', lastname='Lucke', department=Department{id=2, description='development'}}
Controller.delete
Controller.list
Employee{id=3, firstname='hans', lastname='muster', department=null}
Employee{id=4, firstname='paula', lastname='kuster', department=Department{id=1, description='sales'}}
```