

Workshop: Spring Boot & Data Übungen

Ablauf

Dieser Workshop gliedert sich in mehrere Vortragsteile, die den Teilnehmern die Thematik Spring Framework und Spring Boot mit Spring Data usw. näherbringen. Im Anschluss daran sind jeweils einige Übungsaufgaben von den Teilnehmern – idealerweise in Gruppenarbeit – am Rechner zu lösen.

Voraussetzungen

- 1) Aktuelles JDK 11, idealerweise auch JDK 17, installiert
- 2) Aktuelles Eclipse installiert (Alternativ: Spring Tool Suite oder IntelliJ IDEA)

Teilnehmer

- Entwickler und Architekten mit Java-Erfahrung, die ihre Kenntnisse zu Spring, Spring Boot und Data vertiefen möchten

Kursleitung und Kontakt

Michael Inden

Derzeit freiberuflicher Buchautor und Trainer

E-Mail: michael_inden@hotmail.com

Blog: <https://jaxenter.de/author/minden>

Weitere Kurse (Java, Unit Testing, ...) biete ich gerne auf Anfrage als Inhouse-Schulung an.

TAG 2: Spring Boot Basics

Aufgabe 16: Using Spring Initializr

20 min

Using the Spring-Initializr (<https://start.spring.io/>) create a spring boot app (java 11 version) called "ex16-my-first-spring-boot-app" having web as dependency.

**Project**☒ Maven Project ☐ Gradle Project**Language**☒ Java ☐ Kotlin ☐ Groovy**Spring Boot**☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (M3) ☐ 2.5.6 (SNAPSHOT) ☒ 2.5.5
☐ 2.4.12 (SNAPSHOT) ☐ 2.4.11**Project Metadata**

Group com.example

Artifact demo

Name ex16-my-first-spring-boot-app

Description Demo project for Spring Boot

Package name com.example.demo

Packaging ☒ Jar ☐ WarJava ☐ 17 ☒ 11 ☐ 8

- Create a simple `index.html` page just showing a short greeting message.
- Expose an endpoint **`/greeting`** to return "Hi Workshops Participants".
- Change the default Spring Boot banner shown below by providing a file `banner.txt`



to "Schönen Feierabend", "auf zum Bier", "Spring Boot is cool ☺"

[Hint: use <http://bit.ly/2T2ShFU> or <http://bit.ly/2HfReeo>


- Just for fun, rename `animated-banner.gif` into `banner.gif` and start again! Surprise!

Aufgabe 17: AsmiqAcademyApp From Spring To Spring Boot**30 min**


In exercise 14 we implemented a REST-Service for the AsmiqAcademyApp. Now migrate the entire AsmiqAcademyApp to Spring-Boot. To migrate to spring-boot do the following:


- a. Use the maven project "**ex17-asmiq-academy-boot-app-template**" as starting point and copy this into the project to "**ex17-asmiq-academy-boot-app**"

- b. Delete the highlighted packages in the above project. Why?

▼  ex17-asmiq-academy-boot-app


▼  src/main/java


>  ch.asmiq


>  ch.asmiq.config

>  ch.asmiq.controller

>  ch.asmiq.interfaces

>  ch.asmiq.model

>  ch.asmiq.service

>  ch.asmiq.tomcat

>  ch.asmiq.webinitializer

- c. **Remove all maven dependencies in pom.xml!**

Just add the "spring-boot-starter-web" as the only dependency. Does it sound good?

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.5.5</version>
</dependency>
```

Remove the original class AsmiqAcademyApp and create a new one like the following. Complete the TODOs in the AsmiqAcademySprinBootApplication class

```
//TODO
public class AsmiqAcademySprinBootApplication {
    public static void main(String[] args) {
        //TODO
    }
}
```

- d. Run the App and execute <http://localhost:8080/courses> and verify with the result:

```
[{"name": "Java (Online)", "price": 50, "quantity": 2}, {"name": "Design Patterns (Online)", "price": 60, "quantity": 3}, {"name": "Testing (Online)", "price": 40, "quantity": 1}]
```

- e. One of customers required the courses response to be in XML instead of JSON. How would you do that? Try with suitable Accept-Headers and **produces =**

```
curl -H "Accept: application/json" http://localhost:8080/courses
curl -H "Accept: application/xml" http://localhost:8080/courses]
```

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.12.5</version>
</dependency>
```

Aufgabe 18: Spring Beans List

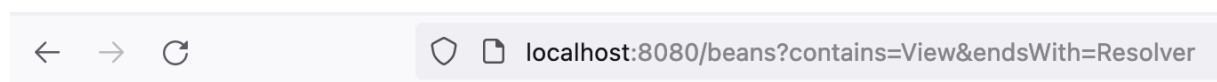
30 min

Auf den Folien und in der Demo hatte ich das Bereitstellen von Informationen zu den in Spring definierten Beans gezeigt. Auch wurde das Filtern von Daten anhand von Query-Parametern vorgestellt. Dieses Wissen soll nun kombiniert werden und folgende Filterungen anbieten:

- startsWith / endsWith
- contains

Darüber hinaus ist eine einfache Listendarstellung mit Thymeleaf gewünscht, die in etwa wie unten gezeigt aussieht und die zudem die aktivierten Filterungen anzeigt.

Als Basis dient das Projekt "**ex18-spring-boot-web-with-ui-template**" und muss geeignet umbenannt und erweitert werden.



Beans Lists

Filter Condition(s):

Ends with: Resolver
Contains: View

Name
beanNameViewResolver
conventionErrorViewResolver
defaultViewResolver
mvcViewResolver
thymeleafViewResolver

Aufgabe 19: PersonApp

45 min

Erstelle eine Spring Boot App, die einen einfachen REST-Controller für eine Klasse Person anbietet:

- POST einmal mit allen Attributen als Parameter und
- POST mit den Daten als JSON-String
- GET
- GET `"/filter-older-than"` und GET `"/filter-by-nationality"`

Nutze folgende Fragmente sowie eine Liste zur Datenhaltung als Ausgangspunkt.

```
@RestController
@RequestMapping("/persons")
public class PersonController {

    private List<Person> persons = new ArrayList<>();

    ---

    public class Person
    {
        private String name;
        private String nationality;
        private int age;
        ...
    }
}
```

a) Füge per Hand verschiedene Personen ein.

```
{ "name" : "Beat", "nationality" : "swiss", "age" : 35 }
{ "name" : "Peter", "nationality" : "german", "age" : 29 }
{ "name" : "Heinz", "nationality" : "german", "age" : 55 }
{ "name" : "Yannis", "nationality" : "german", "age" : 6 }
```

Nutze curl und Eingaben wie nachstehend:

```
curl -d '{ "name" : "Beat", "nationality" : "swiss", "age" : 35 }' \
-H "Content-Type: application/json" \
-X POST http://localhost:8080/persons

curl -d 'name=Mike&nationality=german&age=47' \
-X POST http://localhost:8080/persons/fromAttributes
```

b) Abfragen mit curl

- Ermittle alle Personen.
- Ermittle alle Schweizer.
- Ermittle alle Personen älter als 30.

c) Gestalte eine einfache Webseite, die diese Abfragen bereitstellt:



Ermittle alle Personen:

Ermittle alle Schweizer:

Ermittle alle Personen älter als 30:

- d) Füge ein paar Daten bereits zum Start-Up der Applikation ein. Starte mit folgender Modifikation:

```
@SpringBootApplication
public class Application implements CommandLineRunner
{
    @Autowired
    private PersonController controller;

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    public void run(String... args) throws Exception
    {
        // TODO
    }
}
```

Aufgabe 20: PersonApp - Architektur – Design - Layering

30 min

In Aufgabe 19 wurde eine einfache Spring Boot App mit einem REST-Controller und einigen Endpoints erstellt. Dabei war es das Ziel, erstmal prototypisch die grundlegenden Konzepte und APIs zu verstehen. Analysiere einmal, was mögliche Schwachstellen im Design sein könnten? Welche architektonischen Verbesserungspotenziale erkennst du? Denke an folgende Fragestellungen: Wie sollte und kann die Datenhaltung abstrahiert werden? Wie kann man die Funktionalität wiederverwendbar bereitstellen? Wie sieht es mit dem Single Responsibility Principle aus? Welche Aufgaben erfüllt der Controller derzeit und wie kann man das geeignet aufteilen?

PART Spring Data Access

Aufgabe 21: PersonApp - JPA-Verbindung zur Datenbank

45 min

Aufgabe 21a: Nutze nun ein Repository zur Verbindung mit einer Datenbank im REST-Controller. Welche Auswirkungen hat das auf die beiden Filterungen? Wie bildet man diese nun am besten ab? Wie muss die Klasse Person angepasst werden?

```
@Autowired
```

```
private PersonRepository personRepository;
```

Denke an die weiteren Dependencies, etwa:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>2.5.6</version>
</dependency>
```

Aufgabe 21b: Binde die H2 Web Console durch die nachfolgende WebConfiguration und Einträge in den application.properties ein, um Abfragen auf der Datenbank zu ermöglichen und um die Auswirkungen von Kommandos prüfen zu können.

```
@Configuration
```

```
public class WebConfiguration
```

```
{
```

```
    @Bean
```

```
    ServletRegistrationBean h2servletRegistration()
```

```
    {
```

```
        final ServletRegistrationBean registrationBean =
            new ServletRegistrationBean(new WebServlet());
```

```
        registrationBean.addUrlMappings("/console/*");
```

```
        return registrationBean;
```

```
    }
```

```
}
```

Aufgabe 21c: Rufe die H2 Web Console mit <http://localhost:8080/console> auf und verwende `jdbc:h2:mem:testdb` als Connection String für die JDBC URL. Führe SELECTs aus und mache dich mit den Möglichkeiten der Web Console vertraut. Versuche etwa INSERTs wie folgt – was ist problematisch?

```
INSERT INTO PERSON (ID, NAME, NATIONALITY, AGE) VALUES (-10, 'Mike', 'german', 47);
INSERT INTO PERSON (ID, NAME, NATIONALITY, AGE) VALUES (-11, 'Karthi', 'german', 33);
```

Aufgabe 22: Design-Diskussion

20 min

Mit dem Wissen um DAOs und REST-Applikationen, wie sollte ein Design aussehen. Skizzieren Sie einen REST-Controller, Service und Datenzugriff mit DAO oder Repository. Wie handhabt man die Konvertierung von Entity in DTO. Benötigt man diese noch? Wie erfolgt das Mapping? Schauen Sie sich einmal das Tool MapStruct an: <https://mapstruct.org/>

Verbindung mit MongoDB

Aufgabe 23: Stelle die Entity-Klasse so um, dass sie für MongoDB als Document genutzt werden kann. Modifiziere das Repository, sodass es für MongoDB geeignet ist. Welche Änderungen sind im REST-Controller bzw. der Application nötig?

```
import org.springframework.data.annotation.Id;
```

Denke an:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
  <version>2.5.6</version>
</dependency>
```

Aufgabe 24: Installiere ein MongoDB Datenbank-Tool und inspiziere die Datenbank. In welcher Collection landen die Personen-Daten?

- <https://docs.mongodb.com/compass/master/install/>
- <https://nosqlbooster.com/downloads>

Aufgabe 25: Füge einige Personen hinzu, etwa folgende:

```
{ "name" : "Beat", "nationality" : "swiss", "age" : 35 }
{ "name" : "Peter", "nationality" : "german", "age" : 29 }
{ "name" : "Heinz", "nationality" : "german", "age" : 55 }
{ "name" : "Yannis", "nationality" : "german", "age" : 6 }
```

Erweitere das Repository, um Aktionen wie folgende und führe ein paar dieser Abfragen aus:

```
List<Person> findByAgeBetween(int lower, int upper);
int countByAgeBetween(int lower, int upper);
```

```
List<Person> findTop3ByAgeLessThan(int maxAge);
List<Person> findByAgeLessThanOrderByNameAsc(int maxAge);
```

```
List<Person> getNameLike(String name);
```

```
List<Person> findByNameOrNationality(String name,
                                     String lastName);
```

```
List<Person> findByNameInAndAgeBetween(Collection<String> names,
                                       int lower, int upper);
```


Validation

Aufgabe 26: Validation

20 min

Gegeben sei eine Klasse **Customer.java**, bislang ohne Validierung. Diese soll nun hinzugefügt und ausgewertet werden. Folgende Bedingungen sollen gelten:

- Der Vorname ist nicht leer und zwischen 2 und 100 Zeichen lang
- Der Nachname darf nicht leer sein.
- Das Alter muss positiv oder 0 sein.
- Das Geburtsdatum muss in der Vergangenheit liegen.
- Der Start des nächsten Urlaubs muss heute oder in der Zukunft liegen.
- Schließlich soll die Kontakt-Email ein gültiges Format besitzen.

Erweitern und führen Sie zur Prüfung die Klasse **CustomerValidationExample.java** aus.