OSP
Otto Group Solution Provider

Spring Boot 2 - Grundlagenworkshop

Theorie und Übungsaufgaben

Dresden | 20.05.2019

Agenda für Einleitung

OSP
Otto Group Solution Provider

- Was ist Spring / Spring Boot?
- Dependency Injection
- Application Context
- Component Scan
- Interne und externe Konfiguration
- Unit- und Integrationstests
- Spring Boot Starter
- Spring Boot Initializr

OSP
Otto Group Solution Provider

Was ist Spring / Spring Boot?

Was ist Spring / Spring Boot?

OSP
Otto Group Solution Provider

- Ziel:
  - Infrastruktur auf Anwendungsebene
  - Entwickler soll sich auf Implementierung der Geschäftslogik konzentrieren können

- Spring Boot setzt auf Spring auf
- Kein eigenes Framework, sondern ermöglicht und beschleunigt das Bauen von Spring-basierten Anwendungen

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Was ist Spring / Spring Boot?

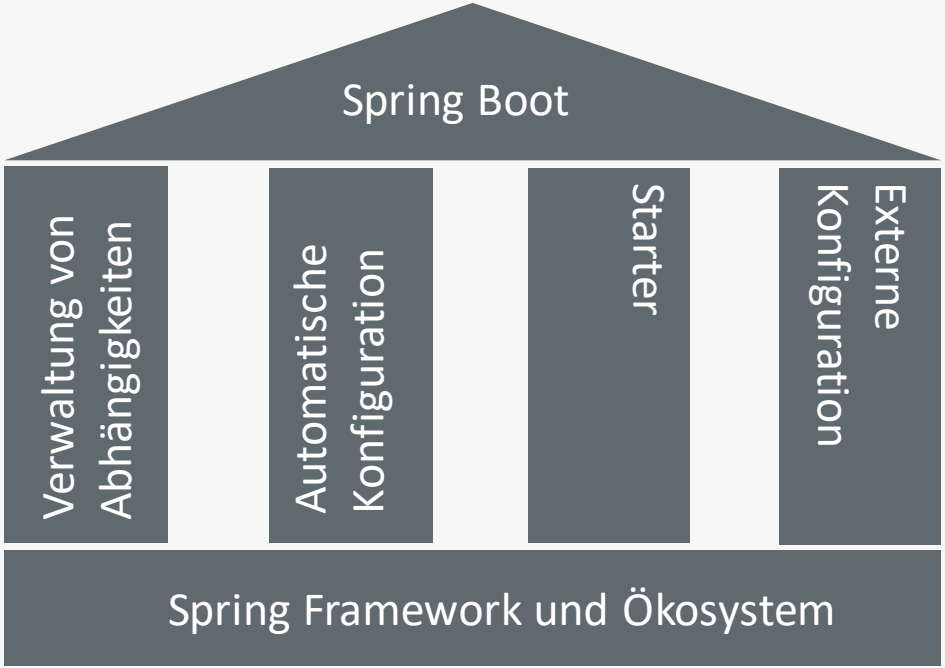
OSP
Otto Group Solution Provider

- Spring-Boot-Applikationen bieten:
  - Eigenständige Anwendungen ohne externe Laufzeitabhängigkeiten
  - Eingebettete Container (Tomcat, Jetty, …)
  - Automatische Konfiguration
  - Häufig benötigte nicht-funktionale Eigenschaften: Metriken, Health Checks, externe Konfiguration
- „Convention over Configuration“ + Einsatz von Annotationen = Bau eines lauffähigen Artefaktes
- Einfaches Erzeugen von Docker-Images mit lauffähigem Artefakt
- Sehr leichtes Verteilen des Artefaktes in Cloud (AWS)

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Was ist Spring / Spring Boot?

OSP
Otto Group Solution Provider



Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

OSP
Otto Group Solution Provider

Dependency Injection

Dependency Injection

OSP
Otto Group Solution Provider

- Wünschenswertes Ziel der Software-Entwicklung:
  - Lose Kopplung von Komponenten: Kopplung existiert nur an definierten Schnittstellen, Komponenten können leicht ausgetauscht werden.
  - Hohe Kohäsion von Komponenten: Zusammenspiel zwischen Methoden und Attributen einer Klasse ist eng. Die Klasse erfüllt genau eine Aufgabe.
  - Möglichkeit, das zu erreichen: *Dependency Injection*
    - *Kontrolle der Erzeugung von Komponenten wird einem Framework übergeben (Inversion of Control, IoC)*

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Dependency Injection

Otto Group Solution Provider

- Objekte („Beans“) instanziieren die benötigten Kollaborateure nicht selbst
- Abhängige Objekte werden stattdessen injiziert

```
public class SomeController {

    @Autowired
    private SomeRepositoryInterface someRepository;

    @Autowired
    public SomeController(SomeRepositoryInterface someRepository) {
        this.someRepository = someRepository;
    }

    public void doSomething() {
        this.someRepository.doSomethingVeryImportant();
    }

}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Dependency Injection

OSP
Otto Group Solution Provider

- Zentrales Element des Spring-Frameworks: Spring Container
- Der Container verwaltet „Beans“
- Beans werden durch ihn konfiguriert und instanziiert
- Er verwaltet den gesamten Lebenszyklus von Beans
- Die Abhängigkeiten zwischen allen Beans sind als Meta-Daten vorhanden
- Beans können mit verschiedenen Scopes definiert werden:
  - Singleton (Default) → zustandslos implementieren!
  - Prototype
  - Request, Session → Webanwendungen

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

OSP
Otto Group Solution Provider

Application Context

Application Context

OSP
Otto Group Solution Provider

- Bildet den Einstiegspunkt in das Spring-Framework
  - Bietet Möglichkeit zur Konfiguration von Beans und deren Abhängigkeiten
- Es existieren verschiedene Implementierungen:
  - XML-basiert
  - Annotationen
  - Java-basiert auf Config-Klassen
- Auf XML-Dateien sollte in modernen Anwendungen verzichtet werden!

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Application Context

OSP
Otto Group Solution Provider

- Konfiguration per XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id=„someBean" class="de.example.SomeBean" />

    <bean id="anotherBean" class="de.example.AnotherBean">
        <constructor-arg ref="someBean" />
    </bean>

</beans>
```

Application Context

OSP
Otto Group Solution Provider

- Konfiguration per Java

```
@Configuration
public class JavaConfig {
    @Bean(name = "someBean")
    public SomeBean someBean() { return new SomeBean(); }

    @Bean
    @Qualifier("anotherBean")
    public AnotherBean anotherBean() {
        return new AnotherBean(someBean());
    }
}
```

Hinweis: Trotz „new“-Operator wird hier nur eine Instanz der Bean erzeugt, weil Spring intern Proxies einsetzt!

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Application Context

OSP
Otto Group Solution Provider

- Konfiguration per Annotation

```java
import org.springframework.stereotype.Component;

@Component
public class SomeBean {
}
```

```java
import org.springframework.stereotype.Component;

@Component
public class AnotherBean {
    public AnotherBean(SomeBean someBean) {

    }
}
```

Vorteil: geringerer Konfigurationsaufwand

Nachteil: fachliche Objekte werden mit Annotationen „verunreinigt“

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

OSP
Otto Group Solution Provider

Component Scan

Component Scan

OSP
Otto Group Solution Provider

- Automatische Suche nach Komponenten (@Component) per **@SpringBootApplication**
- **@SpringBootApplication** fasst zusammen: **@EnableAutoConfiguration**, **@ComponentScan**, **@Configuration**

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

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

}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Component Scan

Otto Group Solution Provider

- @EnableAutoConfiguration
  - Aktiviert die automatische Konfiguration anhand von Abhängigkeiten
  - Beispiel: Hinzufügen der Abhängigkeit zu H2 (in-memory DB) konfiguriert diese automatisch
- @ComponentScan
  - Sucht im Package der Applikation (z.B. "com.example.demo") nach Klassen mit der Annotation @Component
- @Configuration
  - Ermöglicht das Registrieren weiterer Beans per Konfigurationsklassen
  - Beispiel: @Import({*JavaConfig.class*, *SomeComponent.class* })

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

OSP
Otto Group Solution Provider

Interne und externe Konfiguration

Interne und externe Konfiguration

OSP
Otto Group Solution Provider

- Ziel: Eine Applikation sollte nicht für verschiedene Umgebungen unterschiedlich gebaut werden müssen
- Die Applikation sollte sich an die Umgebung (Dev, Test, Prod, …) anpassen können
- Spring Boot setzt hierfür auf „Properties“
- Diese können aus verschiedenen Quellen (PropertySources) mit Werten befüllt werden (aufsteigende Priorität):
  - application.properties/.yml bzw. application-{profile}.properties/.yml
  - Umgebungsvariablen des Betriebssystems
  - Anwendungsparameter (--property=value)
  - Werte aus @SpringBootTest
  - Einstellungen der „devtools“ im Benutzerverzeichnis

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Interne und externe Konfiguration

Externe Konfiguration

OSP
Otto Group Solution Provider

- Die Konfigurationsdatei **application.yml / application.properties**
  - Suchpfad: ./config -> akt. Arbeitsverz. -> Package /config -> Wurzelverz. des Classpath
  - Werte aus den Dateien überschreiben sich in dieser Reihenfolge
  - Beispiel:

.yml

```yaml
myproperty: myvalue

server:
  port: 9090

example:
  servers:
      - name: wert
        url: wert
      - name: wert
        url: wert
```

.properties

```properties
myproperty = myvalue

server.port = 9090

example.servers[0].name = wert
example.servers[0].url = wert
example.servers[1].name = wert
example.servers[1].url = wert
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Interne und externe Konfiguration

Externe Konfiguration

OSP
Otto Group Solution Provider

- Zugriff auf definierte Properties:

```
@ConfigurationProperties
@Component
@Validated
public class Configuration {

    @Value("${myproperty}")
    @Size(min=5)
    private String myproperty;

    public String getMyproperty() {
        return myproperty;
    }
}
```

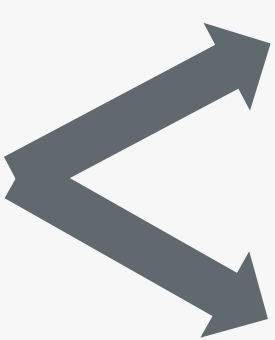
```
@Autowired
private Configuration configuration;

public void method() {
    // ...
    String myproperty = configuration.getMyproperty();
    // ...
}
```

Interne und externe Konfiguration

Externe Konfiguration

- Mit Hilfe von „Profilen“ können Konfigurationen gruppiert werden
- Damit können Konfigurationen für verschiedene Umgebungen (Entwicklung, Testsystem, Integrationssystem, Produktion, …) definiert werden

```yaml
spring:
  profiles: "test"
myproperty: "Wert für das Testsystem"
server:
  port: 8080
---
spring:
  profiles: "prod"
myproperty: "Wert für Produktion!!!"
server:
  port: 9090
---
spring:
  profiles:
    active: "dev"
```



application-test.yml

application-prod.yml

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Interne und externe Konfiguration

Externe Konfiguration

OSP
Otto Group Solution Provider

- Setzen des aktiven Profiles:
  - Definition in der application.yml per Property **spring.profiles.active**

```yaml
spring:
  profiles:
    active: "dev"
```

  - Starten der Applikation:

```
java -jar application.jar --spring.profiles.active=prod
```

  - Starten per Maven:

```
mvn spring-boot:run -Dspring.profiles.active=prod1,prod2
```

Interne und externe Konfiguration

Interne Konfiguration

OSP
Otto Group Solution Provider

- Interne Konfiguration: Mit Hilfe der Annotation **@Profile** können Beans für Profile aktiviert oder deaktiviert werden

```
@Configuration
@Profile("prod")
public class JavaConfig {
    @Bean(name = "someBean")
    public SomeBean someBean() { return new SomeBean(); }

    @Bean(name = "anotherBean")
    public AnotherBean anotherBean() {
        return new AnotherBean(someBean());
    }
}
```

Interne und externe Konfiguration

Interne Konfiguration

Otto Group Solution Provider

- Das Profil kann auch direkt an der Komponente / Bean definiert werden
- Es ist die Angabe einer Liste von Profilen möglich

```
@Component
@Profile("prod")
public class SomeBean {
}
```

```
@Component
@Profile({"test","!prod"})
public class AnotherBean {
    public AnotherBean(SomeBean someBean) {

    }
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

OSP
Otto Group Solution Provider

Unit- und Integrationstests

Unit- und Integrationstests

OSP
Otto Group Solution Provider

- Framework bietet Unterstützung für Unit- und Integrationstests (über spring-boot-starter-test)
- Unit-Tests können prinzipiell wie gewohnt implementiert werden
- Wichtig dafür ist, dass alle Abhängigkeiten einer Komponente dem Konstruktor übergeben werden
  - Dadurch entstehen keine Komponenten in ungültigem Zustand
  - Abhängigkeiten können gemockt werden
  - Spring Boot bringt dafür **Mockito** mit
- JUnit 4 und JUnit 5 werden unterstützt
  - JUnit 4 ist Standard, für Version 5 müssen Abhängigkeiten angepasst werden (pom.xml)

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Unit- und Integrationstests

Integrationstests

OSP
Otto Group Solution Provider

- Eine Testklasse kann mit dem JUnit-Runner „SpringRunner.class“ ausgeführt werden
  - **@RunWith(SpringRunner.class)**
- Dadurch wird ein Kontext gestartet und die Abhängigkeiten der Testklasse erfüllt
- Zusätzlich kann per **@ContextConfiguration** Einfluss auf den Application Context genommen werden
  - Hierfür kann z.B. eine Konfigurationsklasse angegeben werden, welche die Beans für den Test erzeugt
  - @ContextConfiguration(classes = MyJavaConfig.class)
- Alternativ führt **@SpringBootTest** eine (vollständige) Konfiguration durch und erlaubt das Setzen von Properties
  - @SpringBootTest(properties = „myproperty=value“)

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Unit- und Integrationstests

Integrationstests mit Test-Slices

OSP
Otto Group Solution Provider

- Mit Hilfe von Test-Slices können Integrationstests gezielt für einzelne Komponenten durchgeführt werden
  - Dabei wird kein vollständiger Kontext aufgebaut
- Testklassen können mit Annotationen versehen werden, die abhängig vom zu testenden Kontext sind
  - **@WebMvcTest** – Baut einen Testkontext nur mit „web-relevanten“ Komponenten (z.B. Controller) auf
  - **@DataJpaTest** – Baut einen Testkontext für Tests der Persistenzschicht auf, bietet automatisch eine in-memory H2-Datenbank

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

OSP
Otto Group Solution Provider

Spring Boot Starter

Spring Boot Starter

Otto Group Solution Provider

- Module, die Abhängigkeiten und automatische Konfiguration mitbringen
- Lassen sich durch Hinzufügen als Abhängigkeit im Projekt einbinden
- Automatische Konfiguration kann überschrieben werden

```
<dependencies>
   …
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
   </dependency>
   …
</dependencies>
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Spring Boot Starter

Beispiele

Otto Group Solution Provider

- spring-boot-starter-test
  - Unit-Tests mit Application Context
- spring-boot-starter-web
  - Implementierung von Microservices und Webanwendungen
  - Bringt einen vorkonfigurierten Tomcat-Server mit
- spring-boot-starter-security
  - Authentifizierung und Autorisierung
- spring-boot-starter-data-jpa / spring-boot-starter-data-mongodb
  - Persistente Datenhaltung (relational und dokumentbasiert)

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Spring Boot Starter

Abhängigkeiten von Spring Data JPA

Otto Group Solution Provider

```
[INFO] +- org.springframework.boot:spring-boot-starter-data-jpa:jar:2.1.2.RELEASE:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-aop:jar:2.1.2.RELEASE:compile
[INFO] |  |  \- org.aspectj:aspectjweaver:jar:1.9.2:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-jdbc:jar:2.1.2.RELEASE:compile
[INFO] |  |  +- com.zaxxer:HikariCP:jar:3.2.0:compile
[INFO] |  |  \- org.springframework:spring-jdbc:jar:5.1.4.RELEASE:compile
[INFO] |  +- javax.transaction:javax.transaction-api:jar:1.3:compile
[INFO] |  +- javax.xml.bind:jaxb-api:jar:2.3.1:compile
[INFO] |  |  \- javax.activation:javax.activation-api:jar:1.2.0:compile
[INFO] |  +- org.hibernate:hibernate-core:jar:5.3.7.Final:compile
[INFO] |  |  +- org.jboss.logging:jboss-logging:jar:3.3.2.Final:compile
[INFO] |  |  +- javax.persistence:javax.persistence-api:jar:2.2:compile
[INFO] |  |  +- org.javassist:javassist:jar:3.23.1-GA:compile
[INFO] |  |  +- net.bytebuddy:byte-buddy:jar:1.9.7:compile
[INFO] |  |  +- antlr:antlr:jar:2.7.7:compile
[INFO] |  |  +- org.jboss:jandex:jar:2.0.5.Final:compile
[INFO] |  |  +- com.fasterxml:classmate:jar:1.4.0:compile
[INFO] |  |  +- org.dom4j:dom4j:jar:2.1.1:compile
[INFO] |  |  \- org.hibernate.common:hibernate-commons-annotations:jar:5.0.4.Final:compile
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Spring Boot Starter

Abhängigkeiten von Spring Web MVC

Otto Group Solution Provider

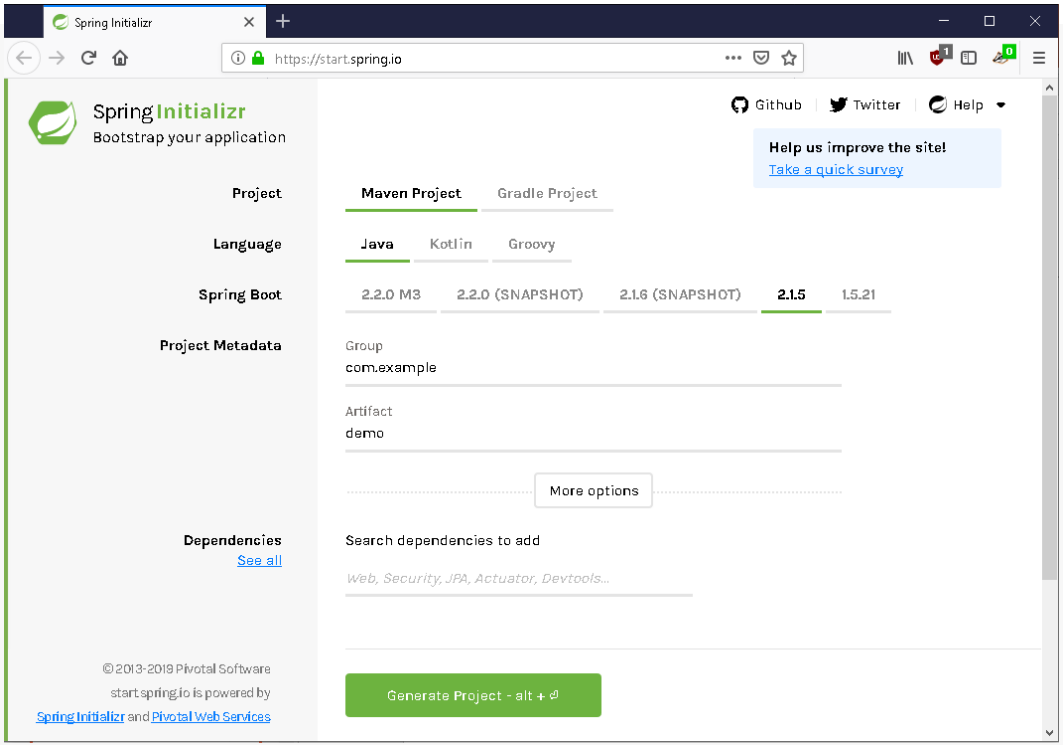
```
[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:2.1.2.RELEASE:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-json:jar:2.1.2.RELEASE:compile
[INFO] |  |  +- com.fasterxml.jackson.core:jackson-databind:jar:2.9.8:compile
[INFO] |  |  |  +- com.fasterxml.jackson.core:jackson-annotations:jar:2.9.0:compile
[INFO] |  |  |  \- com.fasterxml.jackson.core:jackson-core:jar:2.9.8:compile
[INFO] |  |  +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.9.8:compile
[INFO] |  |  +- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.9.8:compile
[INFO] |  |  \- com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.9.8:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter-tomcat:jar:2.1.2.RELEASE:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-core:jar:9.0.14:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-el:jar:9.0.14:compile
[INFO] |  |  \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:9.0.14:compile
[INFO] |  +- org.hibernate.validator:hibernate-validator:jar:6.0.14.Final:compile
[INFO] |  |  \- javax.validation:validation-api:jar:2.0.1.Final:compile
[INFO] |  +- org.springframework:spring-web:jar:5.1.4.RELEASE:compile
[INFO] |  \- org.springframework:spring-webmvc:jar:5.1.4.RELEASE:compile
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

OSP
Otto Group Solution Provider

Spring Initializr

Spring Initializr



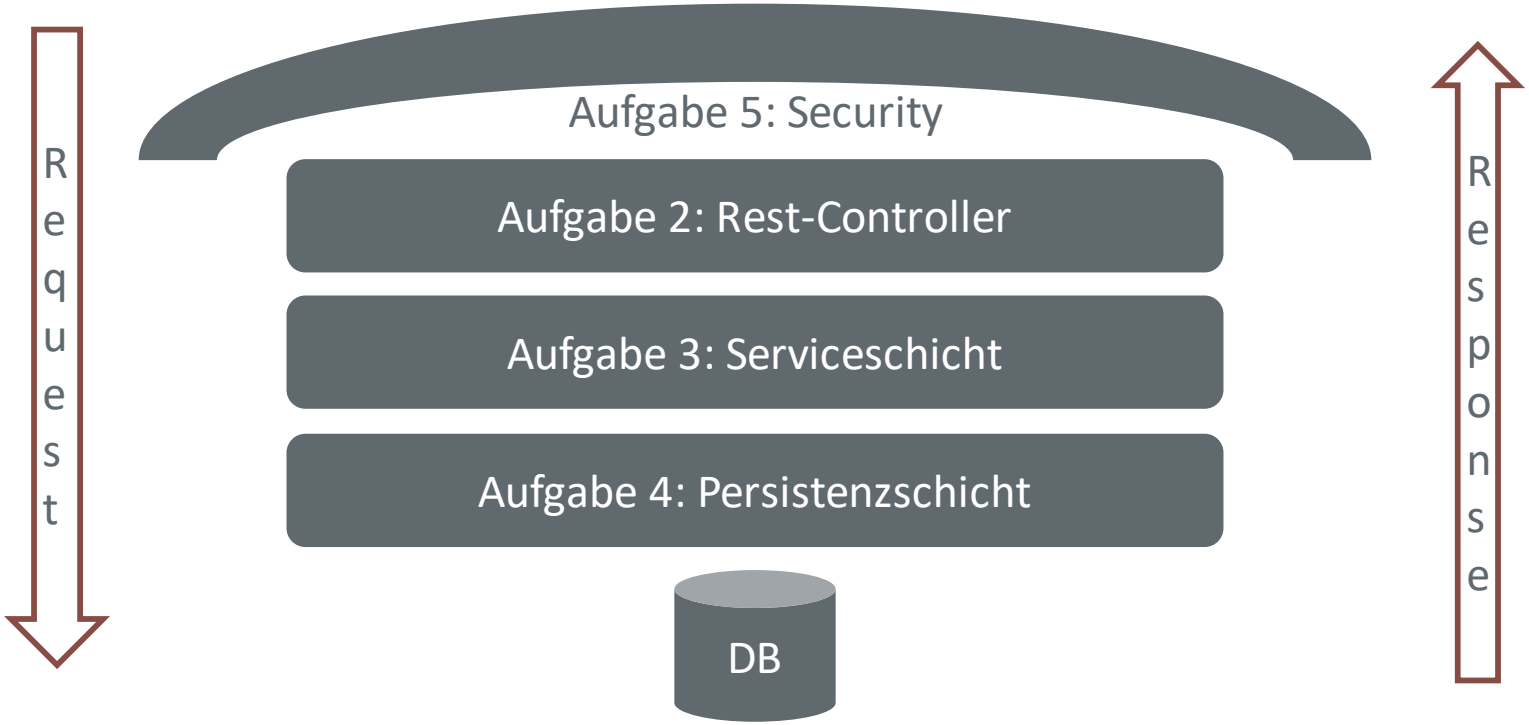
Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Agenda für Übungsaufgaben

OSP
Otto Group Solution Provider

Aufgabenkomplex 1: Projekt erstellen	11:00 – 11:30
Aufgabenkomplex 2: REST-Controller	12:30 – 14:30
Aufgabenkomplex 3: Serviceschicht	14:30 – 15:30
Aufgabenkomplex 4: Persistenzschicht	15:45 – 16:45
Aufgabenkomplex 5: Security	16:45 – 17:55

Übungsaufgaben



Übungsaufgaben

Otto Group Solution Provider

https://github.com/DennyPazak0101/spring-boot-workshop

Bearbeitungszeit: 11:00 – 11:30

OSP
Otto Group Solution Provider

Aufgabenkomplex 1: Projekt erstellen

Aufgabenkomplex 1: Projekt erstellen

Zielsetzung

OSP
Otto Group Solution Provider

- Initialisierung eines Spring Boot Projekts durch den Spring Initializr (https://start.spring.io)
  - Erzeugt ein Maven-Projekt mit allen notwendigen Abhängigkeiten
- Spring Boot Anwendung starten
  - Ausführung von mvn spring-boot:run
- Actuator aufrufen
  - Erweiterung von REST-Endpoints für Metriken, Healtchecks, Beans des Application Context
  - Aufrufbar über HTTP GET

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Bearbeitungszeit: 12:30 – 14:30

OSP
Otto Group Solution Provider

Aufgabenkomplex 2: REST-Controller

Aufgabenkomplex 2: REST-Controller

Zielsetzung

Otto Group Solution Provider

- Erstellung von REST-Endpoints für Tierhandlung Webservice
  - Auflisten, erstellen und entfernen von Haustieren
- Fehlerbehandlung von REST-Endpoints
  - Eigene Fehler von Tierhandlung behandeln
  - Bestehende Validierungsfehler von Spring Boot anpassen
- Temporäre Persistenz durch java.util.Map bereitstellenp

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 2: REST-Controller

Aufbau eines REST-Controller

Otto Group Solution Provider

```
@RestController
public class SuperMarketRestController {
    private SupermarketService service;

    @Autowired public SuperMarketRestController(SupermarketService service) {
        this.service = service;
    }

    public Iterable<Article> listArticles(...) {...}
    public Article createArticle(...) {...}
    public void deleteArticle(...) {...}
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 2: REST-Controller

Aufbau eines Exception-Handlers

Otto Group Solution Provider

```
@ControllerAdvice
public class SuperMarketExceptionHandler {
    @ExceptionHandler(SuperMarketApiException.class)
    public ResponseEntity<Object> handle(SuperMarketApiException e) {
        return new ResponseEntity<>(new ApiError(e.getMessage()), HttpStatus.BAD_REQUEST);
    }
}
```

Aufgabenkomplex 2: REST-Controller

Request Matching

Otto Group Solution Provider

```
curl -X GET http://localhost:8080/supermarket/spieces?producer=ostmann -H "Accept:application/json"
```

```
@RestController
@RequestMapping("/supermarket")
public class SuperMarketRestController {
    @GetMapping(path = "/{category}", produces = MediaType.APPLICATION_JSON_VALUE)
    public Iterable<Articles> listArtictles(@PathVariable("category") String category,
                                            @RequestParam("producer") String producer) {
        // implementation omitted
    }
}
```

Aufgabenkomplex 2: REST-Controller

Request Matching

Otto Group Solution Provider

```
curl -X POST http://localhost:8080/supermarket/spieces -H "Content-Type:application/json" -H "Accept:application/json" -D
"{"producer": "ostmann", "name": "paprika powder"}"


@RestController
@RequestMapping("/supermarket")
public class SuperMarketRestController {
    @PostMapping(path = "/{category}", consumes = MediaType.APPLICATION_JSON_VALUE,
                    produces = MediaType.APPLICATION_JSON_VALUE)
    public Article createArticle(@PathVariable("category") String category,
                                 @RequestBody @Validated Article article) {
        // implementation omitted
    }
}
```

Aufgabenkomplex 2: REST-Controller

Request Matching

Otto Group Solution Provider

```
curl -X DELETE http://localhost:8080/supermarket/spieces/1
```

```
@RestController
@RequestMapping("/supermarket")
public class SuperMarketRestController {
    @DeleteMapping(path = "/{category}/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteArticle(@PathVariable("category") String category,
                              @PathVariable("id") String id) {
        // implementation omitted
    }
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 2: REST-Controller

Aufbau eines Web-MVC Tests

Otto Group Solution Provider

```
@RunWith(SpringRunner.class)
@WebMvcTest(SuperMarketRestController.class)
public class SuperMarketRestControllerTest {
    @Autowired private MockMvc mockMvc;

    @Test public void test() {
        MockHttpServletRequestBuilder request = MockMvcRequestBuilders.get("/supermarket/spieces")
                .accept(MediaType.APPLICATION_JSON_UTF8);
        mockMvc.perform(request)
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8));
    }
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 2: REST-Controller

Annotations Übersicht

Otto Group Solution Provider

- **@RestController:** Kennzeichnet Klasse als Controller. Methoden liefern „web responses“.
- **@RequestMapping(„/path“):** Kennzeichnet Klasse (oder Methoden), welche „web requests“ behandeln
- **@GetMapping(„/path“), @PostMapping(„/path“), @DeleteMapping(„/path“)**
  - Kennzeichnet Methode als Handler für einen HTTP-Request
  - Attribute „produces“ und „consumes“ erlauben das Setzen eines MediaTypes
- **@PathVariable:** Kennzeichnet Parameter einer Methode als Platzhalter in URI
- **@RequestBody:** Kennzeichnet Parameter einer Methode als Body des HTTP-Requests
- **@ControllerAdvice:** Kennzeichnet Klasse, welche ExceptionHandler für alle Controller bereitstellt
- **@ExceptionHandler(MyException.class):** Kennzeichnet Methode, welche eine Ausnahme behandelt und z.B. ein Objekt vom Typ *ResponseEntity<Object>* zurückgibt.
- **@RunWith(SpringRunner.class):** Kennzeichnet einen JUnit-Test, welcher durch SpringRunner gestartet werden soll
- **@WebMvcTest:** Kennzeichnet einen JUnit-Test als „Test-Slice“ für Webanwendung

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Bearbeitungszeit: 14:30 – 15:30

OSP|
Otto Group Solution Provider

Aufgabenkomplex 3: Serviceschicht

Aufgabenkomplex 3: Serviceschicht

Zielsetzung

Otto Group Solution Provider

- Erstellen einer Serviceschicht für Tierhandlung Webservice
  - Verschiebung von Fachlogik aus REST-Controller in Service
  - Temporäre Persistenz durch java.util.Map verschieben

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 3: Serviceschicht

Aufbau eines Service

Otto Group Solution Provider

```
@Service
public class SupermarketService {
    private ArticleRepository articleRepository;

    @Autowired public SupermarketService(ArticleRepository articleRepository) {
        this.articleRepository = articleRepository;
    }

    public Iterable<Article> listArticle(...) {...}
    public Article createArticle(...) {...}
    public void deleteArticle(...) {...}
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 3: Serviceschicht

Annotations Übersicht

Otto Group Solution Provider

- @Service
  - Kennzeichnet eine Klasse (oder Interface) als Service (und als „Komponente“)
- @Autowired
  - Kann an Methode oder Attribut verwendet werden. Veranlasst Spring, die Bean zu „injizieren“
- @MockBean
  - Kann an Methode/Attribut in JUnit-Test verwendet werden. Spring injiziert einen Mockito-Mock der Bean
- @Import
  - Kann an Testklasse verwendet werden, um für den Test weitere Klassen in Application-Context aufzunehmen

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Bearbeitungszeit: 15:45 – 16:45

OSP
Otto Group Solution Provider

Aufgabenkomplex 4: Persistenzschicht

Aufgabenkomplex 4: Persistenzschicht

Zielsetzung

Otto Group Solution Provider

- Erstellen einer Persistenzschicht für Tierhandlung Webservice
  - Ablösen von temporärerer Persistenz mit java.util.Map durch Repository
  - Erweitern von Repository um eigene Finder bzw. Query-Methoden

Aufgabenkomplex 4: Persistenzschicht

Aufbau eines Repository

Otto Group Solution Provider

```
@Repository
public interface ArticleRepository extends CrudRepository<Article, Long> {
    List<Article> findByProducer(String producer);

    @Query(„select a from Article a where a.name = :name“)
    List<Article> findByName(String name);
}
```

Aufgabenkomplex 4: Persistenzschicht

Aufbau eines Spring Data Tests

Otto Group Solution Provider

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class ArticleRepositoryTest {
    @Autowired private ArticleRepository repository;

    @Test public void test() {
        List<Articles> articles = repository.findByProducer("ostmann");
        assertThat(pets)
                .isNotNull()
                .isNotEmpty()
                .extracting("name")
                .contains(tuple("paprika powder"));
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 4: Persistenzschicht

Annotations Übersicht

Otto Group Solution Provider

- @Repository
  - Kennzeichnet Klasse/Interface als Repository und „Komponente“
- @Query(„select u from Universe u where x.name = :name“)
  - Kennzeichnet Methode als „finder“ unter Verwendung des SQL (JPQL)
- @DataJpaTest
  - Kennzeichnet einen JUnit-Test als „Test-Slice“ für Persistenzschicht

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Bearbeitungszeit: 16:45 – 17:55

OSP
Otto Group Solution Provider

Aufgabenkomplex 5: Security

Aufgabenkomplex 4: Persistenzschicht

Zielsetzung

Otto Group Solution Provider

- Absichern des Tierhandlung Webservice durch unbefugten Zugriff
  - Verwendung von Authentifizierung gegen In-Memory-Realm
  - Verwendung von Autorisierung mit Rollen bei REST-Endpoints

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 5: Security

Aufbau einer Security-Konfiguration

Otto Group Solution Provider

```
@Configuration @EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
                .authorizeRequests()
                .antMatchers(HttpMethod.GET, "/supermarket**").permitAll()
                .antMatchers(HttpMethod.POST, "/supermarket**").hasRole("ADMIN")
                .anyRequest().authenticated()
                .and()
                .httpBasic()
                .and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 5: Security

Aufbau eines Web-MVC Tests mit Security Anonymous

Otto Group Solution Provider

```
@RunWith(SpringRunner.class)
@WebMvcTest(SuperMarketRestController.class)
public class SuperMarketRestControllerTest {
    @Autowired private MockMvc mockMvc;

    @Test public void test() {
        MockHttpServletRequestBuilder request = MockMvcRequestBuilders.get("/supermarket/spieces")
                .accept(MediaType.APPLICATION_JSON_UTF8)
                .with(anonymous());
        mockMvc.perform(request)
                .andDo(print())
                .andExpect(status().isUnauthorized());
    }
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018

Aufgabenkomplex 5: Security

Aufbau eines Web-MVC Tests mit Security mit HTTP-Basic-Auth

Otto Group Solution Provider

```
@RunWith(SpringRunner.class)
@WebMvcTest(SuperMarketRestController.class)
public class SuperMarketRestControllerTest {
    @Autowired private MockMvc mockMvc;

    @Test public void test() {
        MockHttpServletRequestBuilder request = MockMvcRequestBuilders.get("/supermarket/spieces")
                .accept(MediaType.APPLICATION_JSON_UTF8)
                .with(httpBasic("invaliduser", "invalidpassword"));
        mockMvc.perform(request)
                .andDo(print())
                .andExpect(status().isUnauthorized());
    }
}
```

Quelle: „Spring Boot 2 – Moderne Softwareentwicklung mit Spring 5“ – Michael Simons, 1. Auflage 2018