



Spring Workshop

Schnelleinstieg Spring

https://github.com/Michaeli71/Spring_Intro_3d

Michael Inden
Freiberuflicher Consultant und Trainer



Agenda

Workshop Contents



- **PART 1: Einführung**
 - Geschichte von Spring
 - Spring Architecture & ApplicationContext
- **PART 2: Dependency Injection**
 - Grundlagen IoC / DI
 - Spring Beans
 - Configuration
 - XML
 - Annotation
 - Java Config
 - Arten der Dependency Injection
 - Dependency Resolution Strategies

Workshop Contents



- **PART 3: Bean Initialization / Scopes + Lifecycle**
 - Bean Initialization & Laziness
 - Zirkuläre Abhängigkeiten
 - Bean Scopes
 - Bean Lifecycle Callbacks
- **PART 4: Spring MVC**
 - Grundlagen DispatcherServlet & Controller
 - Beispiele Simple Rest Controller
 - Beispiele mit Thymeleaf
- **PART 5: Spring Tool Suite**



PART 3:

Bean Initialization /

Scopes + Lifecycle

- Bean Initialization & Laziness
 - Zirkuläre Abhängigkeiten
 - Bean Scopes
 - Bean Lifecycle Callbacks
-



Bean Initialization





Bean Initialization

- Bisher keine Gedanken gemacht
- immer den Standard genutzt
- standardmäßig werden Beans DIREKT BEIM START initialisiert
- **hilft, Fehler bei der Konfiguration von Abhängigkeiten zu vermeiden**



Beispielklasse

```
@Component
public class SimpleCar {

    public void run() {
        //start battery
        //start engine
        //start the wheel
    }

    public void playMusic(){
        audioPlayer.playMusic();
    }

    @Autowired
    public void setAudioPlayer(AudioPlayer pl){
        this.audioPlayer = pl;
    }

    //setter injected battery,engine,
    //wheel & audioplayer
}
```



Anwendungskontext

```
@Component  
public class AndroidAuto implements AudioPlayer {
```

```
    public AndroidAuto() {  
        //check the internet connection (3 secs)  
        //google account login (2 secs)  
        //access for microphone and camera (4 secs)  
        //load the playlists (2 secs)  
        //cache the playlists (5 secs)  
    }  
}
```

```
var context = new AnnotationConfigApplicationContext(App.class);  
SimpleCar simpleCar = context.getBean(SimpleCar.class);  
simpleCar.run();
```

```
//later if user press the play button  
simpleCar.playMusic();
```



Laziness is good 😊

Lazy Init als eine Abhilfe



```
@Component
public class SimpleCar {

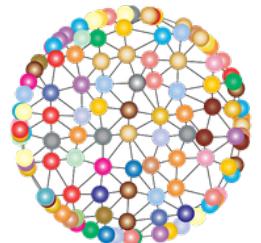
    public void run() {
        //start battery
        //start engine
        //start the wheel
    }

    public void playMusic(){
        if (audioPlayer == null)
            audioPlayer = context.getBean(AudioPlayer.class);
        audioPlayer.playMusic();
    }

    //setter injected battery,engine,wheel
}
```

```
@Lazy
@Component
public class AndroidAuto implements AudioPlayer {

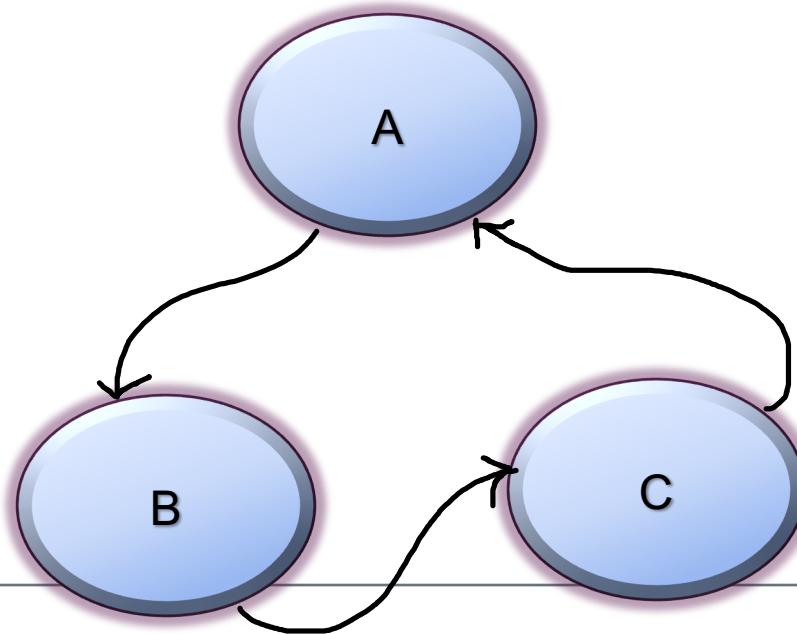
    public AndroidAuto() {
        //check the internet connection (3 secs)
        //google account login (2 secs)
        //access for microphone and camera (4 secs)
        //load the playlists (10 secs)
        //cache the playlists (5 secs)
    }
}
```



**Was sollte man generell
im Design verbessern?**



Zirkuläre Abhängigkeiten



Circular Dependency Example



```
@Component  
public class Student {  
  
    private Department department;  
  
    @Autowired  
    public Student(Department department) {  
        ...  
    }  
}
```

```
@Component  
public class Department {  
  
    private Student student;  
  
    @Autowired  
    public Department(Student student) {  
        ...  
    }  
}
```

Caused by: org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'department': Requested bean is currently in creation: Is there an unresolvable circular reference?



Circular dependencies solution

Modify some dependencies to be injected through setters => after construction

```
@Component  
public class Student {  
  
    private Department department;  
  
    public Student(Department department) {  
        ...  
    }  
  
    @Autowired  
    public void setDepartment(Department dpt){  
        this.department = dpt;  
    }  
}
```



Circular dependencies solution

Lazy initialize certain dependencies => Proxy gets created (*Interface as Indirection)

```
@Component  
public class Student {  
  
    private Department department;  
  
    public Student(@Lazy Department department) {  
        ...  
    }  
}
```

```
@Component  
public class Student {  
  
    private Department department;  
  
    private int age=36;  
  
    public Student(@Lazy Department department) {  
        this.department = department;  
    }  
  
    public int getAge() {  
        return age;  
    }  
}
```

department= Department\$\$EnhancerBySpringCGLIB\$\$f030f2cb (id=38)
 ↳ CGLIB\$BOUND= false
 ↳ CGLIB\$CALLBACK_0= CglibAopProxy\$DynamicAdvisedInterceptor (id=58)
 ↳ CGLIB\$CALLBACK_1= CglibAopProxy\$DynamicUnadvisedInterceptor (id=63)
 ↳ CGLIB\$CALLBACK_2= CglibAopProxy\$SerializableNoOp (id=65)



Bean Scopes



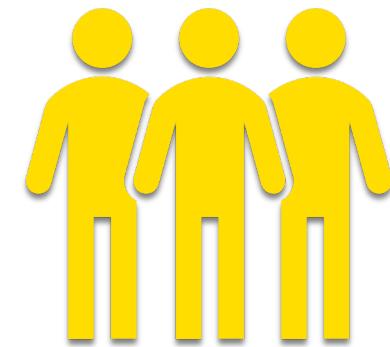


Bean Scopes

Default



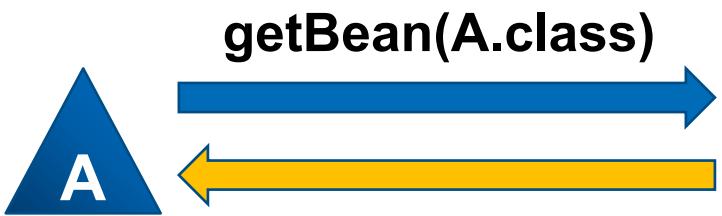
Singleton



Prototype



Singleton



Spring



Singleton

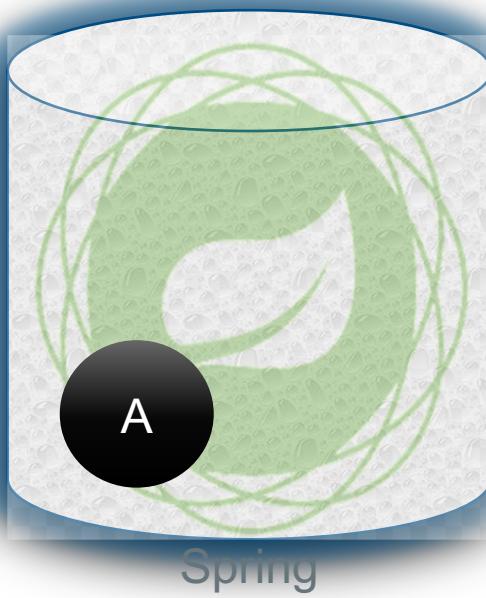


getBean(A.class)





Prototype



getBean(A.class)





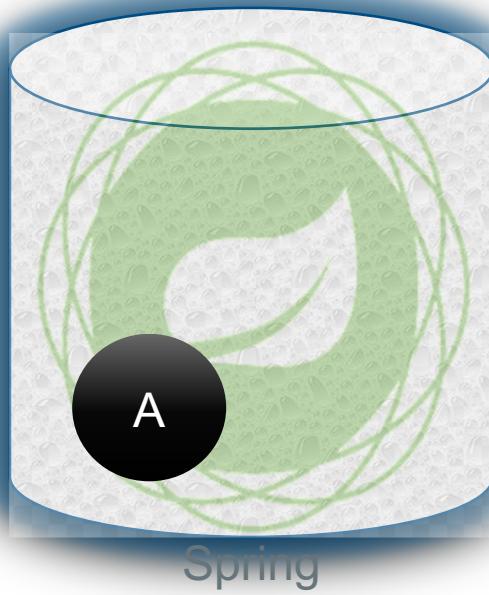
Prototype



getBean(A.class)

→

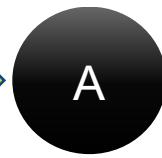
←



getBean(A.class)

←

→



Singleton Example

```
@ComponentScan("ch.javaprofi_academy")
public class BeanScopesApp {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                new ClassPathXmlApplicationContext("bean-def.xml");

        var shape1 = context.getBean(Shape.class);
        System.out.println(shape1);  Shape [color=yellow]

        shape1.setColor("blue");
        System.out.println(shape1);  Shape [color=blue]

        var shape2 = context.getBean(Shape.class);
        System.out.println(shape2);  Shape [color=blue]

        context.close();
    }
}
```

```
@Component
public class Shape {

    private String color = "yellow";

    //setter

}
```

Scope Configuration



`@Scope(scopeName=ConfigurableBeanFactory.SCOPE_PROTOTYPE)`

`@Scope(scopeName=ConfigurableBeanFactory. SCOPE_SINGLETON)`

`@Scope("prototype")`

`@Scope("singleton")`

`@Scope("somethingwrong")`

Caused by: `java.lang.IllegalStateException: No Scope registered for scope name 'somethingwrong'`

Prototype Example

```
@ComponentScan("ch.javaprofi_academy")
public class BeanScopesApp {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("bean-def.xml");

        var shape1 = context.getBean(Shape.class);
        System.out.println(shape1);  Shape [color=yellow]

        shape1.setColor("blue");
        System.out.println(shape1);  Shape [color=blue]

        var shape2 = context.getBean(Shape.class);
        System.out.println(shape2);  Shape [color=yellow]

        context.close();
    }
}
```

```
@Component
@Scope("prototype")
public class Shape {

    private String color = "yellow";

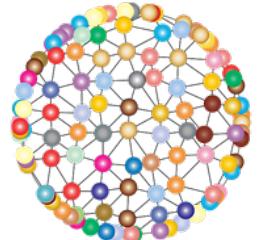
    //setter

}
```

```
};
```



**Was bedeutet Singleton
für Java? Wo ist die
Eindeutigkeit gegeben?**





Singleton is per container and not per classloader (JVM)



```
@ComponentScan("ch.karthi")
public class App {

    public static void main(String[] args) {
        var container = new AnnotationConfigApplicationContext(App.class);
        var container2 = new AnnotationConfigApplicationContext(App.class);

        var shape1 = container.getBean(Shape.class);
        LOG.info(shape1);  INFO: Shape [color=yellow]

        shape1.setColour("blue");
        LOG.info(shape1);  INFO: Shape [color=blue]

        var shape2 = container2.getBean(Shape.class);
        LOG.info(shape2);  INFO: Shape [color=yellow]
    }
}
```

```
@Component
public class Shape {

    // shape
}
```



other singleton pitfalls



```
<?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="scopeTest" class="ch.javaprofi_academy.Shape" scope="singleton">
        <property name="color" value="blue"/>
    </bean>

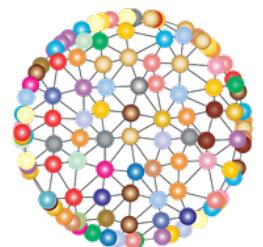
    <bean id="scopeTestDuplicate" class="ch.javaprofi_academy.Shape"
          scope="singleton">
        <property name="color" value="red"/>
    </bean>
</beans>
```



other singleton pitfalls



```
public class SingletonSurprise {  
    public static void main(String[] args) {  
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");  
  
        Shape shape1 = ctx.getBean("scopeTest", Shape.class);  
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);  
  
        System.out.println(shape1 == shape2);  
        System.out.println(shape1 + " :: " + shape2);  
    }  
}
```



Was ist die Ausgabe?



other singleton pitfalls



```
public class SingletonSurprise {  
    public static void main(String[] args) {  
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");  
  
        Shape shape1 = ctx.getBean("scopeTest", Shape.class);  
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);  
  
        System.out.println(shape1 == shape2);           false  
        System.out.println(shape1 + " :: " + shape2);   Shape [color=blue]::Shape [color=red]  
    }  
}
```



other singleton pitfalls

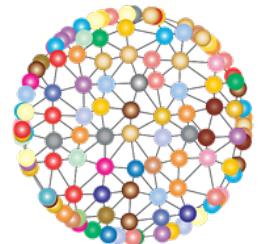


```
public class SingletonSurprise {
    public static void main(String[] args) {
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");

        Shape shape1 = ctx.getBean("scopeTest", Shape.class);
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);

        System.out.println(shape1 == shape2);
        System.out.println(shape1 + "://" + shape2);

        Shape shape3 = ctx.getBean("scopeTest", Shape.class);
        Shape shape4 = ctx.getBean("scopeTestDuplicate", Shape.class);
        System.out.println(shape1 == shape3);
        System.out.println(shape2 == shape4);
        System.out.println(shape3 + "://" + shape4);
    }
}
```



Was ändert sich jetzt?



other singleton pitfalls



```
public class SingletonSurprise {
    public static void main(String[] args) {
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");

        Shape shape1 = ctx.getBean("scopeTest", Shape.class);
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);

        System.out.println(shape1 == shape2);          false
        System.out.println(shape1 + " :: " + shape2);  Shape [color=blue]::Shape [color=red]

        Shape shape3 = ctx.getBean("scopeTest", Shape.class);
        Shape shape4 = ctx.getBean("scopeTestDuplicate", Shape.class);
        System.out.println(shape1 == shape3);          true
        System.out.println(shape2 == shape4);          true
        System.out.println(shape3 + " :: " + shape4);  Shape [color=blue]::Shape [color=red]
    }
}
```

Fazit Singleton vs Prototype



Singleton vs Prototype

- Singleton => Stateless beans
- Prototype => Stateful beans



Fazit Singleton vs Prototype



Bean Scope

- The default one is Singleton. It is easy to change to Prototype





Life Cycle





Lifecycle Callbacks

- **Initialization Callbacks**

- @PostConstruct
- InitializingBean
- initMethod in @Bean



Präzedenz

- **Destruction Callback**

- @PreDestroy
- DisposableBean
- destroyMethod in @Bean



@PostConstruct



```
@Component
public class VideoPlayer {

    private AsmiqAcademyStream asmiqAcademyStream;
    private YoutubeStream youtubeStream;

    private String youtubeApiKey = "y7k8nd9d92d12m";
    private String asmiqAcademyApiKey = "y7k8nd9d92d12m";

    @PostConstruct
    private void initStreams() {
        asmiqAcademyStream =
            AsmiqAcademyStream.builder().withApiKey(asmiqAcademyApiKey).build();
        youtubeStream = YoutubeStream.builder().withApiKey(youtubeApiKey).build();
    }
}

<dependency>
<groupId>javax.annotation</groupId>
<artifactId>javax.annotation-api</artifactId>
<version>1.3.2</version>
</dependency>
```

InitializingBean



```
@Component
public class VideoPlayer implements InitializingBean {

    private AsmiqAcademyStream asmiqAcademyStream;
    private YoutubeStream youtubeStream;

    private String youtubeApiKey = "y7k8nd9d92d12m";
    private String asmiqAcademyApiKey = "y7k8nd9d92d12m";

    @Override
    public void afterPropertiesSet() throws Exception {
        initStreams();
    }

    private void initStreams() {
        asmiqAcademyStream =
            AsmiqAcademyStream.builder().withApiKey(asmiqAcademyApiKey).build();
        youtubeStream = YoutubeStream.builder().withApiKey(youtubeApiKey).build();
    }
}
```

initMethod in @Bean



```
@Component
public class VideoPlayer {

    private AsmiqAcademyStream asmiqAcademyStream;
    private YoutubeStream youtubeStream;

    private String youtubeApiKey = "y7k8nd9d92dl2m";
    private String asmiqAcademyApiKey = "y7k8nd9d92dl2m“;

    private void initStreams() {
        asmiqAcademyStream =
            AsmiqAcademyStream.builder().withApiKey(asmiqAcademyApiKey).build();
        youtubeStream = YoutubeStream.builder().withApiKey(youtubeApiKey).build();
    }
}
```

```
@Configuration
public class AppConfig {
    @Bean(initMethod = "initStreams")
    public VideoPlayer videoPlayer() {
        return new VideoPlayer();
    }
}
```

@Predestroy



```
@Component
public class VideoPlayer {

    private AsmiqAcademyStream asmiqAcademyStream;
    private YoutubeStream youtubeStream;

    private String youtubeApiKey = "y7k8nd9d92d12m";
    private String asmiqAcademyApiKey = "y7k8nd9d92d12m";

    @PreDestroy
    private void signOffStreams() {
        asmiqAcademyStream.signOff();
        youtubeStream.signOff();
    }

    public void run() {
        // start any of the stream
    }
}
```

DisposableBean



```
@Component
public class VideoPlayer implements DisposableBean {

    private AsmiqAcademyStream asmiqAcademyStream;
    private YoutubeStream youtubeStream;

    private String youtubeApiKey = "y7k8nd9d92d12m";
    private String asmiqAcademyApiKey = "y7k8nd9d92d12m";

    @Override
    public void destroy(){
        signOffStreams ();
    }

    private void signOffStreams() {
        asmiqAcademyStream.signOff();
        youtubeStream.signOff();
    }
}
```

destroyMethod in @Bean



```
@Component
public class VideoPlayer {

    private AsmiqAcademyStream asmiqAcademyStream;
    private YoutubeStream youtubeStream;

    private String youtubeApiKey = "y7k8nd9d92dl2m";
    private String asmiqAcademyApiKey = "y7k8nd9d92dl2m";

    private void signOffStreams() {
        asmiqAcademyStream.signOff();
        youtubeStream.signOff();
    }

    public void run() {
        // start any of the stream
    }
}
```

```
@Configuration
public class AppConfig {
    @Bean(destroyMethod = "signOffStreams")
    public VideoPlayer videoPlayer() {
        return new VideoPlayer();
    }
}
```



*Aware-Interfaces für besondere Anwendungsfälle

- Eine weitere Möglichkeit, in den Lebenszyklus einzusteigen, ist das Implementieren von XyzAware-Schnittstellen:

```
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.BeanNameAware;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Component;
```

```
@Component
class MySpringBean implements BeanNameAware, ApplicationContextAware {

    @Override
    public void setBeanName(String name) {
        // ...
    }

    @Override
    public void setApplicationContext(ApplicationContext applicationContext)
        throws BeansException {
        // ...
    }
}
```



Exercises 9 – 11

https://github.com/Michaeli71/Spring_ Intro _3d





PART 4:

Spring MVC





Servlet

- Java program that runs on server.

Servlet container

- A software that provides runtime for the servlets.
(e.g.) Tomcat, Jetty



Simple Servlet – An example

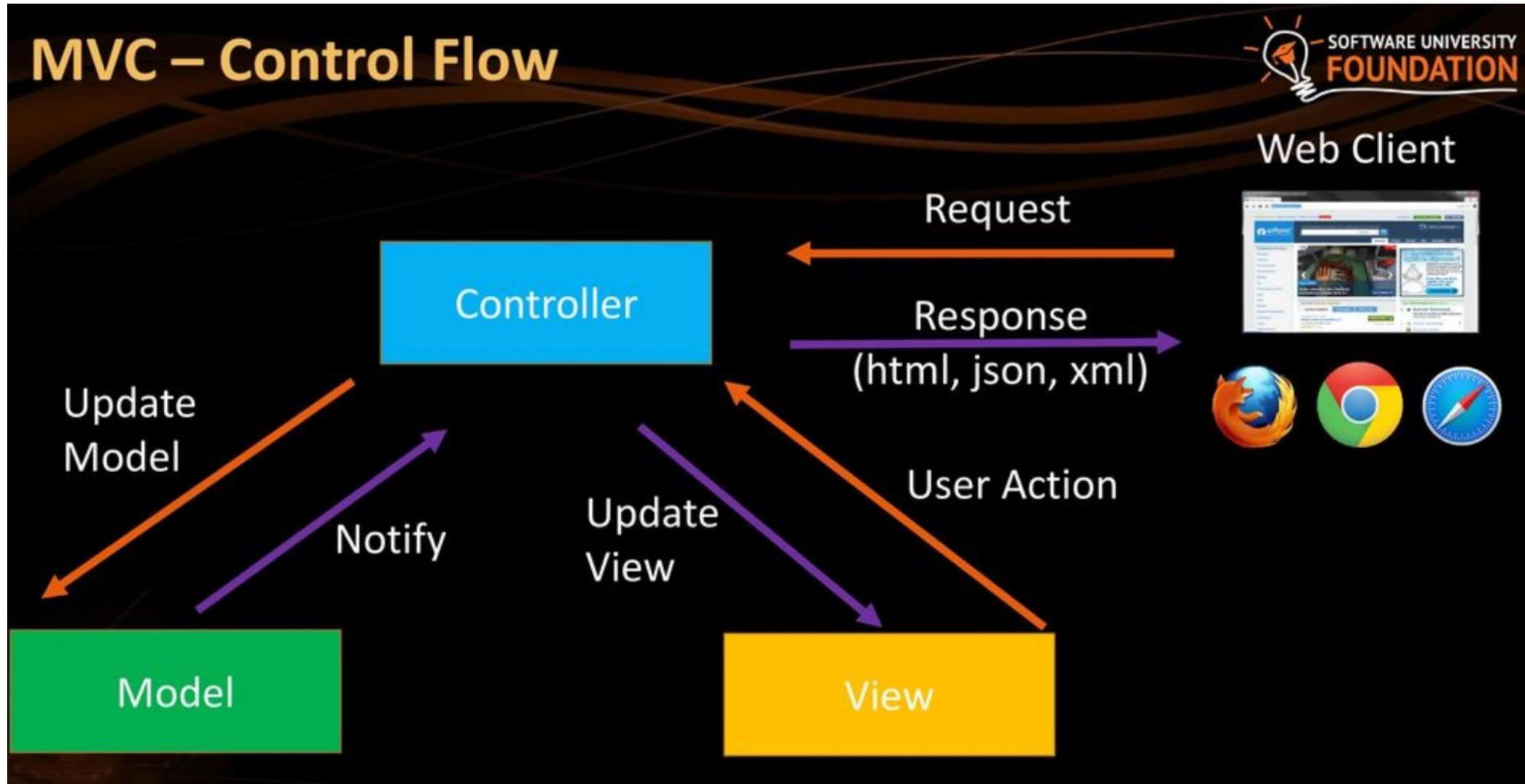
```
@Override  
protected void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
    throws ServletException, IOException {  
    response.setContentType("text/html");  
    request.authenticate(response);  
    PrintWriter writer = response.getWriter();  
    writer.println("<html>");  
    writer.println("<head>");  
    writer.println("this is simple serlvet example");  
    writer.println("</head>");  
    writer.println("<body>");  
    writer.println("Hello HBT :-)");  
    writer.println("</body>");  
    writer.println("</html>");  
}
```



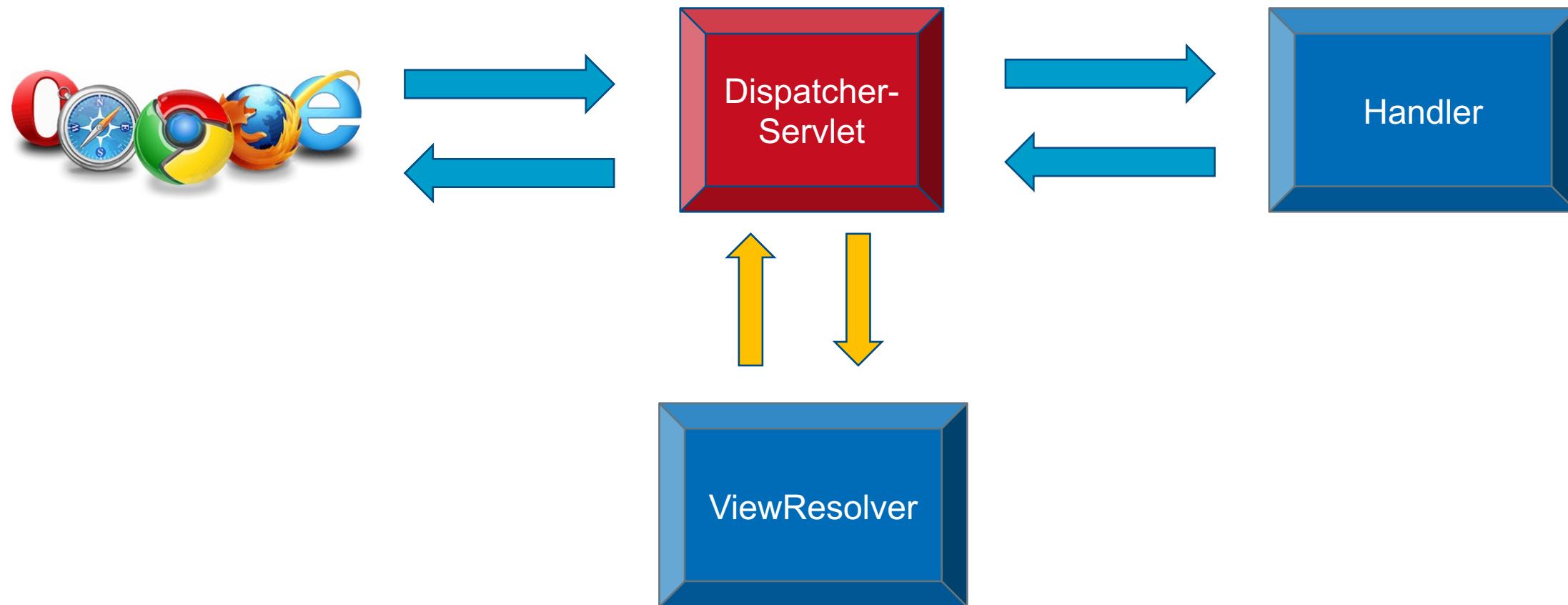
Simple Servlet – An example with methods extracted

```
@Override  
protected void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
throws ServletException, IOException {  
  
    authenticate(response);  
  
    createView(response);  
}
```

MVC – Model View Controller – Logisches Model



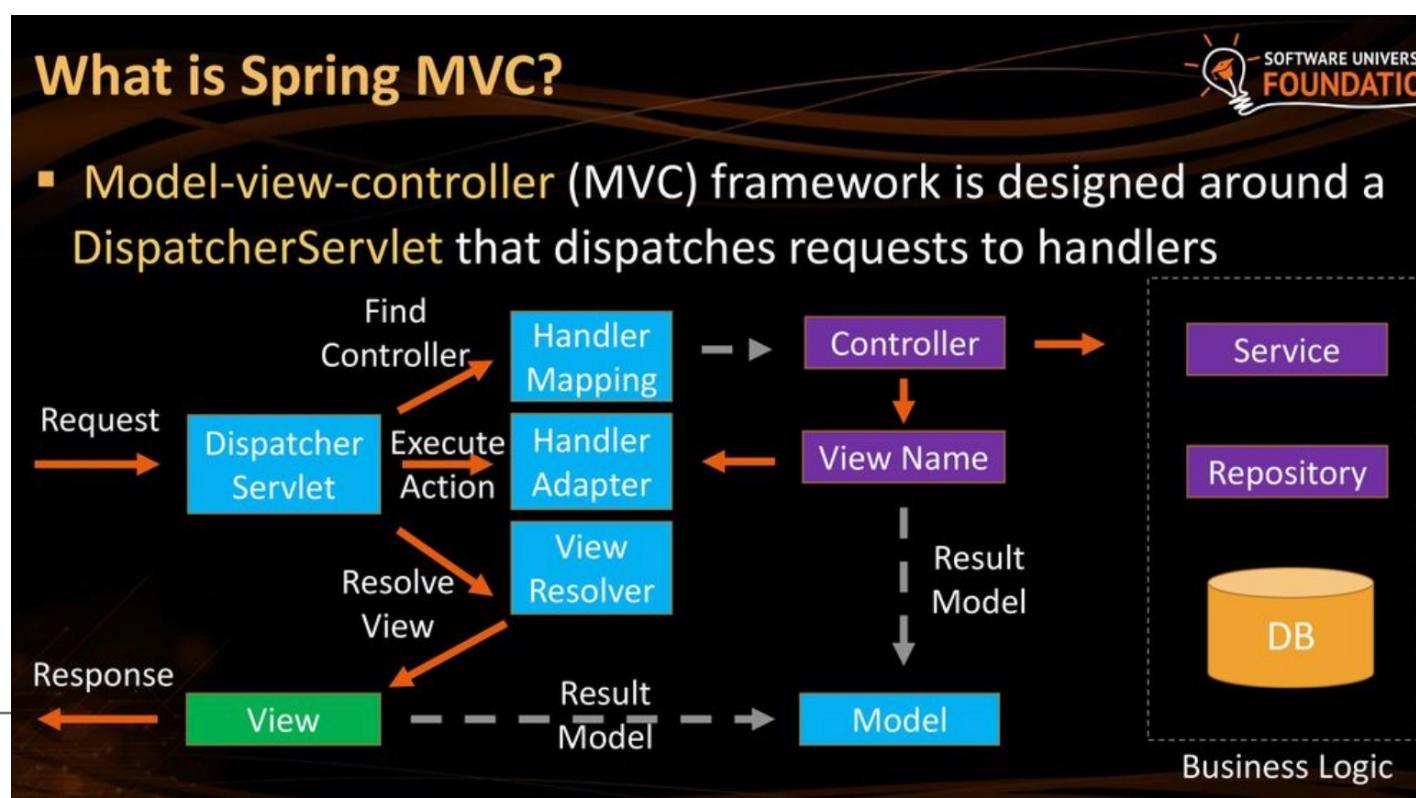
Spring Web-MVC – FrontController Pattern





Blick im Detail

1. Request geht an DispatcherServlet
2. Mithilfe des HandlerMappings wird die URL ausgewertet und es erfolgt die Zuordnung zu Controller und passender Controller-Methode
3. Aufruf vom Controller mit Model-Objekt
4. Ausführung der Controller-Methode und Befüllung des Models, Rückgabe eines View-Namens
5. View-Name wird auf HTML-Seite abgebildet. Das ist Aufgabe vom View-Resolver
6. Schließlich müssen die Model-Daten noch passend ins HTML eingefügt werden.
7. Response wird an Browser geliefert



@Controller



```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Controller {

    /**
     * The value may indicate a suggestion for a logical component name,
     * to be turned into a Spring bean in case of an autodetected component.
     * @return the suggested component name, if any (or empty String otherwise)
     */
    @AliasFor(annotation = Component.class)
    String value() default "";

}
```



@Controller

@Controller

```
public class ShoppingBasket {
```

```
    @RequestMapping(name="/items", method=RequestMethod.GET)
```

```
    public String items() {
```

```
        // return the view name for the shopping lists
```

```
}
```

```
<<More request mapping methods>>
```

```
}
```

```
    public enum RequestMethod {  
        GET, HEAD, POST, PUT, DELETE, ...  
    }
```





Thymeleaf Mini-Intro





Thymeleaf Überblick

- Template Engine zur Gestaltung von Webseiten
- Erlaubt Layouts und Forms
- Basiert auf HTML 5
- Ersatz für JSP / JSF
- Nutzt SpEL
- Kann sowohl im Servlet als auch nicht Web-Umfeld genutzt werden
- Eine Dependency, sofern in Spring Boot:

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



Einfaches Beispiel

```
@Controller  
public class ThymeleafController { Bereitstellung des Models  
    @GetMapping("/home")  
    public String index(Model model) {  
        // Befüllung des Models  
        model.addAttribute("msg", "Greetings from Spring Boot to thymeleaf!");  
  
        // Rückgabe des View-Namens  
        return "index";  
    }  
}
```

A blue arrow points from the text "**Bereitstellung des Models**" to the line "model.addAttribute("msg", "Greetings from Spring Boot to thymeleaf!");".

Verzeichnis und einfache Template-Datei



```
✓  src/main/resources
    static
    ✓ templates
        index.html
```

Standardordner für Templates / View-Fragmente

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <body>
        <h1 th:text="${msg}" />
    </body>
</html>
```

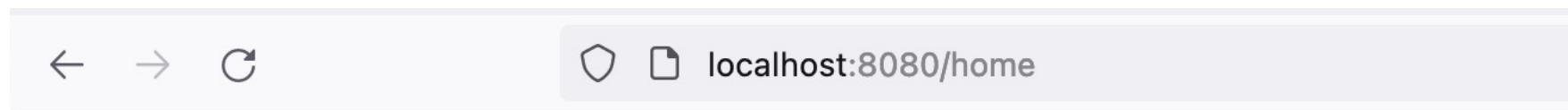
Einfache Auszeichnungssprache

Vorgriff: Spring Boot App



```
@SpringBootApplication
public class ExampleThymeleafApp {

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

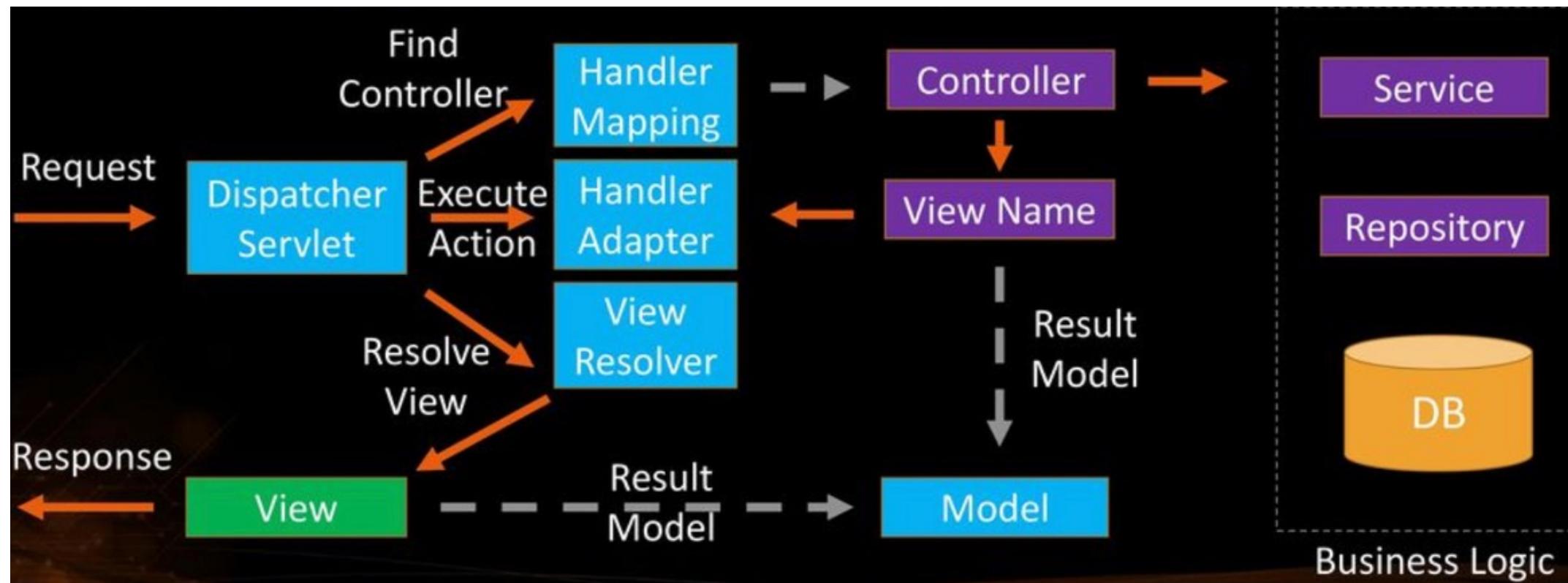


Greetings from Spring Boot to thymeleaf!

Spring Web-MVC – FrontController RECAP



```
@GetMapping("/home")
public String index(Model model) {
    model.addAttribute("msg", "Greetings from Spring Boot to thymeleaf!");
    return "index";
}
```





Controller mit 2 Endpoints

```
@Controller
public class ThymeleafController {

    @Autowired
    PersonDAO personDao;

    @GetMapping("/home")
    public String index(Model model) {
        model.addAttribute("msg", "Greetings from Spring Boot to thymeleaf!");
        return "index";
    }

    @GetMapping(value="/personsList")
    public String getAllPersonsForModel(Model model){
        model.addAttribute("persons", personDao.getAllPersons());
        return "personsList";
    }
}
```



HTML-Datei für Personenliste

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
    <body>
        <h1>Person List</h1>
        <div>
            <u><br></u>
            <u><nobr></nobr></u>
                <a th:href="@{/persons/add}">Add Contact</a> | 
                <a th:href="@{/home}">Back to Home</a>
            <u></nobr></u>
        </div>

        <br/><br/>
        <div>
            <table border="1">
                <tr>
                    <th>Id</th>
                    <th>Firstname</th>
                    <th>Lastname</th>
                    <th>Age</th>
                    <th>Edit</th>
                </tr>
        
```



HTML-Datei für Personenliste

```
...
    <tr th:each ="person : ${persons}">
        <td><a th:href="@{/persons/{personId}(personId=${person.id})}" th:utext="${person.id}">...</a></td>
        <td><a th:href="@{/persons/{personId}(personId=${person.id})}" th:utext="${person.firstName}">...</a></td>
        <td><a th:href="@{/persons/{personId}(personId=${person.id})}" th:utext="${person.lastName}">...</a></td>
        <td><a th:href="@{/persons/{personId}(personId=${person.id})}" th:utext="${person.age}">...</a></td>

        <td><a
th:href="@{/persons/{personId}/edit(personId=${person.id})}">Edit</a></td>
    </tr>
</table>
</div>
</body>
</html>
```



← → ⌂



localhost:8080/personsList

Person List

[Add Contact](#) | [Back to Home](#)

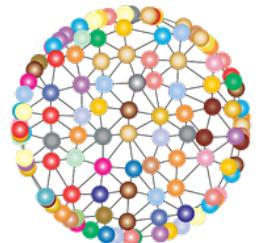
Id	Firstname	Lastname	Age	Edit
1	Michael	Inden	50	Edit
2	Tim	Bötzmeyer	50	Edit
3	Heinz	Mustermann	32	Edit
4	James	Bond	44	Edit

Alternative JSP

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
    <head>
        <title>View Books</title>
        <link href="" rel="stylesheet" type="text/css">
    </head>
    <body>
        <table>
            <thead>
                <tr>
                    <th>ISBN</th>
                    <th>Name</th>
                    <th>Author</th>
                </tr>
            </thead>
            <tbody>
                <c:forEach items="${books}" var="book">
                    <tr>
                        <td>${book.isbn}</td>
                        <td>${book.name}</td>
                        <td>${book.author}</td>
                    </tr>
                </c:forEach>
            </tbody>
        </table>
    </body>
</html>
```



Demo



Wie sieht es denn
mit REST aus?



REST-Controller basierend auf Controller

```
@Controller
public class ShoppingBasket{

    @RequestMapping(name="/items", method=RequestMethod.GET)
    @ResponseBody
    public List<ShoppingItem> items(){
        //return the shopping lists
    }

}
```



@RestController

Das Bildelement mit der Beziehungs-ID 102 wurde in der Datei nicht gefunden.

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Controller

@ResponseBody

public @interface RestController {

/**

* The value may indicate a suggestion for a logical component name,

* to be turned into a Spring bean in case of an autodetected component.

* @return the suggested component name, if any (or empty String otherwise)

* @since 4.0.1

*/

@AliasFor(annotation = Controller.class)

String value() default "";

}

@RestController



```
@Controller      @RestController
public class ShoppingBasket{

    @RequestMapping(name="/items", method=RequestMethod.GET)
    @ResponseBody
    public List<ShoppingItem> items(){
        //return the shopping lists
    }

}
```

@RestController



@RestController

```
public class ShoppingBasket{
```

```
    @GetMapping("/items")
```

```
    @RequestMapping(name="/items", method=RequestMethod.GET)
```

```
    public List<ShoppingItem> items(){
```

```
        //return the shopping lists
```

```
}
```

```
}
```



Shortcut Handler Methods

<code>@RequestMapping(... , method=RequestMethod.GET)</code>	<code>@GetMapping(..).</code>
<code>@RequestMapping(... , method=RequestMethod.POST)</code>	<code>@PostMapping(...).</code>
<code>@RequestMapping(... , method=RequestMethod.PUT)</code>	<code>@PutMapping(...).</code>
<code>@RequestMapping(... , method=RequestMethod.DELETE)</code>	<code>@DeleteMapping(...)</code>



produces => accept header

```
curl http://localhost:8080/items
```

```
@GetMapping("/items")
public List<ShoppingItem> items(){
    return items;
}
```

```
curl -H "accept: application/json" http://localhost:8080/items
```

```
@GetMapping(value="/items", produces=MediaType.APPLICATION_JSON_VALUE)
public List<ShoppingItem> itemsInJSON(){
    return items;
}
```

produces => accept header



```
curl -H "accept: application/xml" http://localhost:8080/items
```

```
@GetMapping(value="/courses", produces= { MediaType.APPLICATION_JSON_VALUE,  
                                         MediaType.APPLICATION_XML_VALUE })  
public List<ShoppingItem> getItems(){  
    return items;  
}  
  
<dependency>  
    <groupId>com.fasterxml.jackson.core</groupId>  
    <artifactId>jackson-core</artifactId>  
    <version>2.12.5</version>  
</dependency>  
<dependency>  
    <groupId>com.fasterxml.jackson.core</groupId>  
    <artifactId>jackson-databind</artifactId>  
    <version>2.12.5</version>  
</dependency>  
<dependency>  
    <groupId>com.fasterxml.jackson.dataformat</groupId>  
    <artifactId>jackson-dataformat-xml</artifactId>  
    <version>2.12.5</version>  
</dependency>
```

MEDIA TYPES



Field Summary

Fields

Modifier and Type	Field and Description
static MediaType	ALL Public constant media type that includes all media ranges (i.e.
static java.lang.String	ALL_VALUE A String equivalent of ALL .
static MediaType	APPLICATION_ATOM_XML Public constant media type for application/atom+xml.
static java.lang.String	APPLICATION_ATOM_XML_VALUE A String equivalent of APPLICATION_ATOM_XML .
static MediaType	APPLICATION_FORM_URL_ENCODED Public constant media type for application/x-www-form-urlencoded.
static java.lang.String	APPLICATION_FORM_URL_ENCODED_VALUE A String equivalent of APPLICATION_FORM_URL_ENCODED .
static MediaType	APPLICATION_JSON Public constant media type for application/json.
static MediaType	APPLICATION_JSON_UTF8 Public constant media type for application/json; charset=UTF-8.
static java.lang.String	APPLICATION_JSON_UTF8_VALUE A String equivalent of APPLICATION_JSON_UTF8 .
static java.lang.String	APPLICATION_JSON_VALUE A String equivalent of APPLICATION_JSON .
static MediaType	APPLICATION_OCTET_STREAM Public constant media type for application/octet-stream.

XML

JSON



consumes => content-type header

```
curl -H "Content-Type: application/json" -X PUT -d {"id":1, "price":32}  
http://localhost:8080/items
```

```
@PutMapping(value="/items", consumes= {MediaType.APPLICATION_JSON_VALUE})  
public List<ShoppingItem> updateItem(@RequestBody ShoppingItem item){  
    //update Item  
}
```



headers

```
curl -H "quantity=3" http://localhost:8080/items
```

```
@GetMapping(value="/items", headers="quantity=3")
public List<ShoppingItem> itemsWithHeaderFiltered(){
    //return items
}
```



params

```
curl http://localhost:8080/items?quantity=2
```

```
@GetMapping(value="/items", params="quantity=2")
public List<ShoppingItem> itemsWithParamterFiltered(){
    //return items
}
```



Get a particular item

```
@GetMapping("/items/{itemId}")
public ShoppingItem getItemById(@PathVariable String itemId){
    //return item with itemId
}
```

<http://host:port/items/1>

```
@GetMapping("/items")
public ShoppingItem getItemById(@RequestParam String itemId){
    //return item with itemId
}
```

<http://host:port/items?itemId=1>

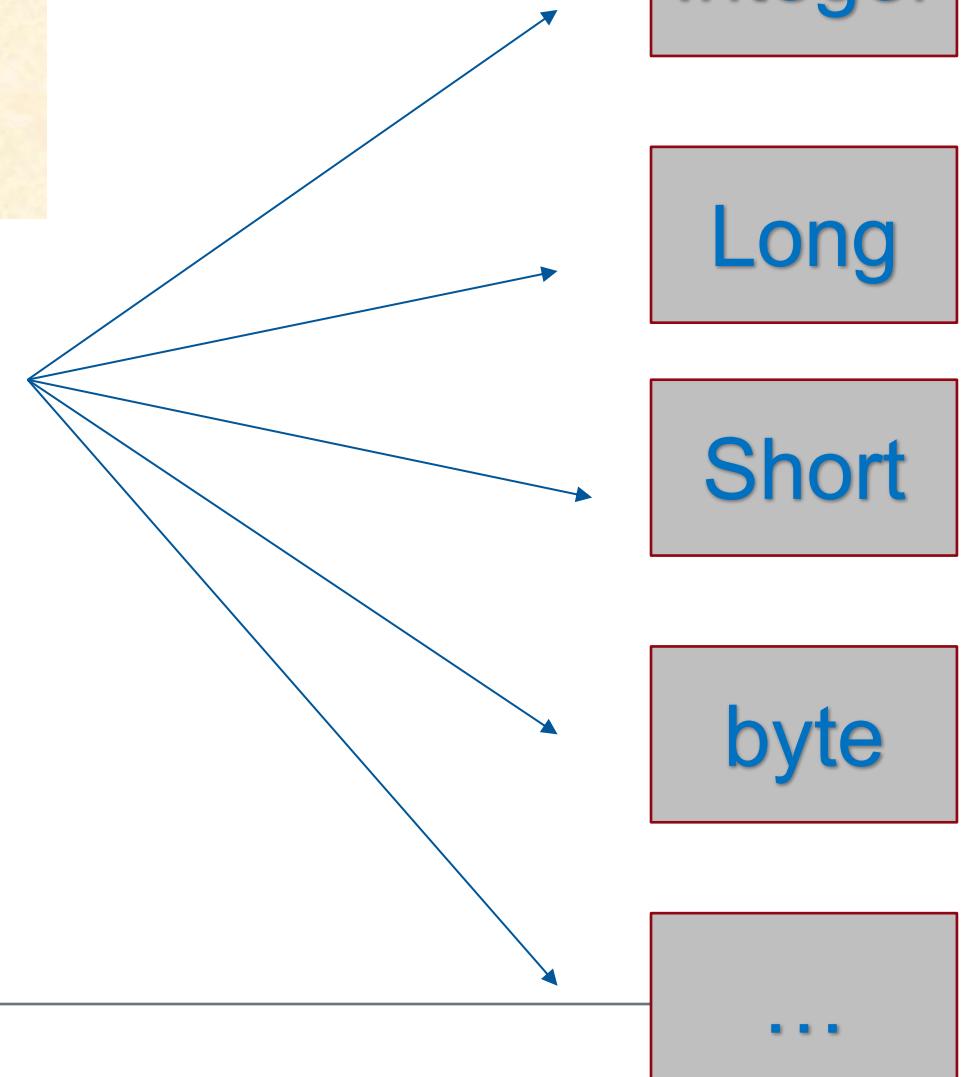


Type Conversion (1)

```
@GetMapping("/items/{itemId}")
public void items(@PathVariable String itemId){
    ...
}
```

<http://host:port/items/1>

String



Type Conversion (2)



```
curl -H "Content-Type: application/json" -X POST -d {"id":1, "price":3}
http://localhost:8080/items
```

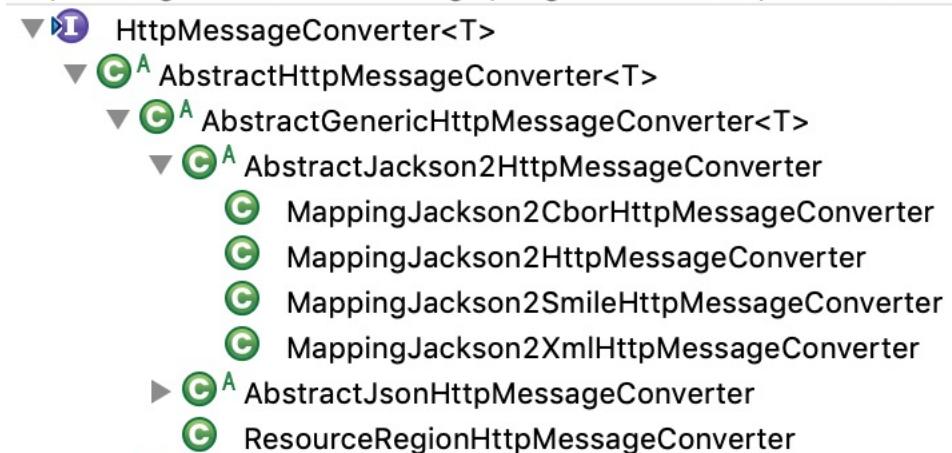
```
@PostMapping("/items")
public ShoppingItem createItem(@RequestBody ShoppingItem item){
    //create the item
}
```

<<Interface>>
HttpMessageConverter

Type Conversion (2)



<<Interface>>
HttpMessageConverter



```
@Configuration  
@EnableWebMvc  
@ComponentScan("ch.karthi")  
public class WebAppConfig {  
  
    @Bean  
    public MappingJackson2HttpMessageConverter jsonMessageConverter() {  
        return new MappingJackson2HttpMessageConverter();  
    }  
}
```



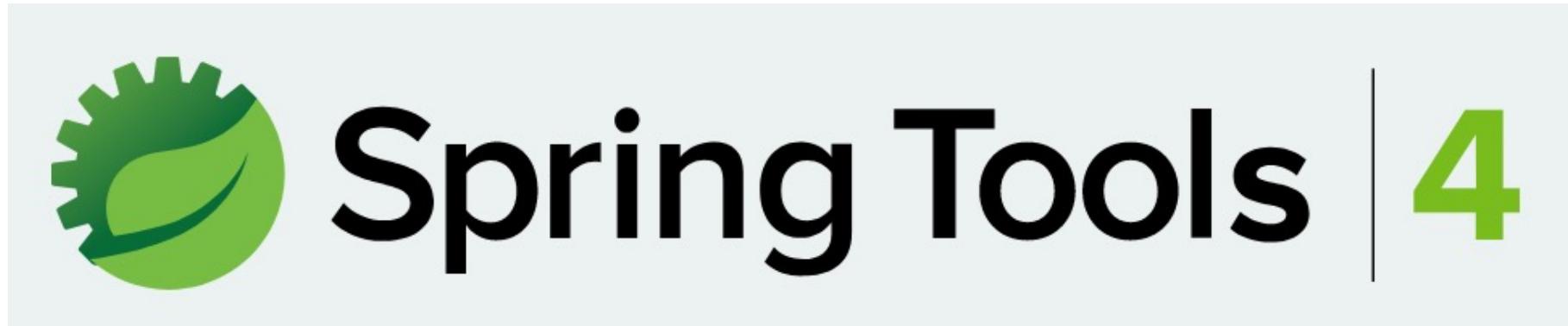
Exercises 13 – 15

https://github.com/Michaeli71/Spring_Intro_3d





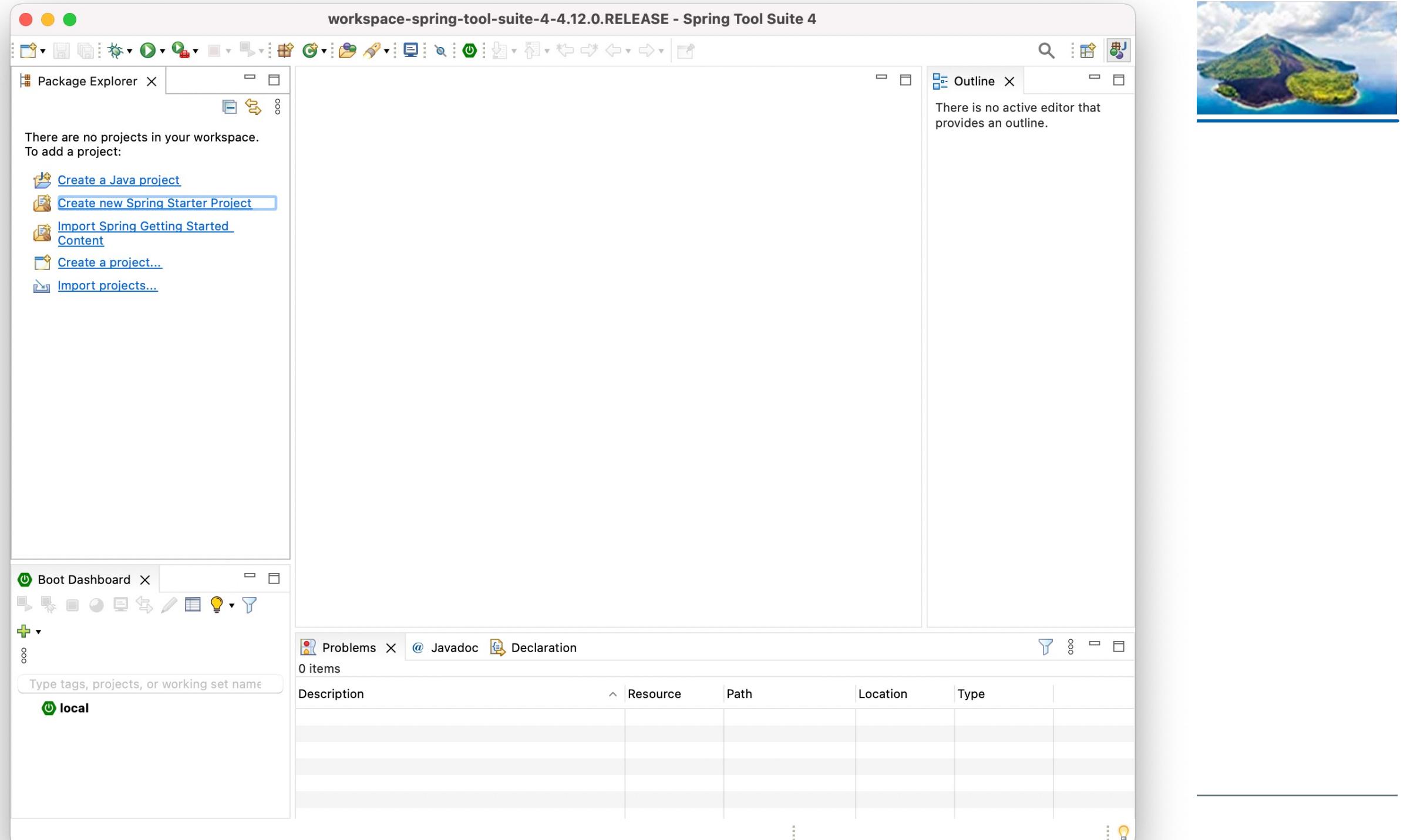
PART 5: STS & Tooling

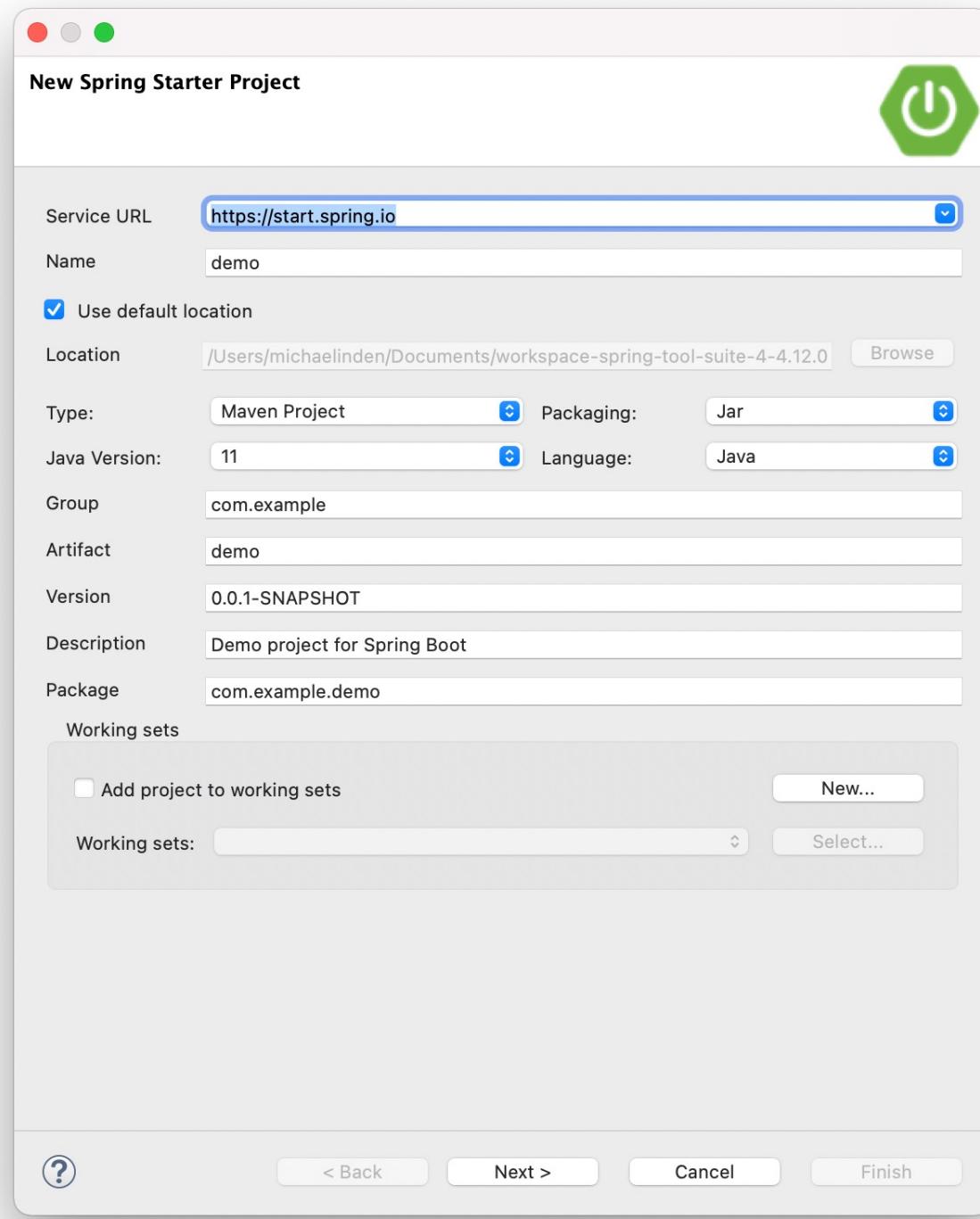




Spring Tool Suite







New Spring Starter Project Dependencies

Spring Boot Version: 2.5.5

Available: Selected:

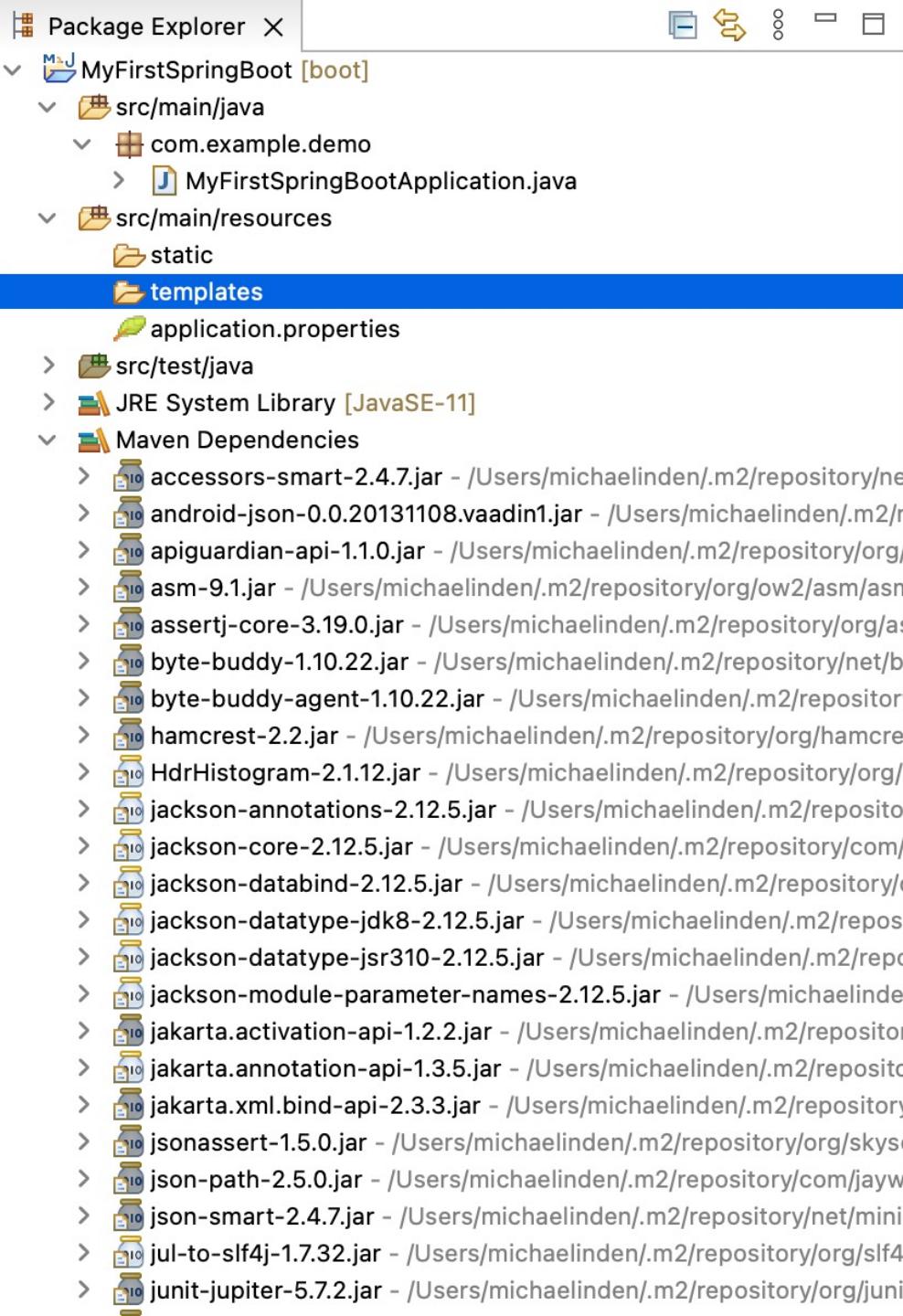
Type to search dependencies

- ▶ Developer Tools
- ▶ Google Cloud Platform
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud
- ▶ Spring Cloud Circuit Breaker
- ▶ Spring Cloud Config
- ▶ Spring Cloud Discovery
- ▶ Spring Cloud Messaging
- ▶ Spring Cloud Routing
- ▶ Spring Cloud Tools
- ▶ Template Engines
- ▶ Testing
- ▶ VMware Tanzu Application Service
- ▶ Web

Make Default Clear Selection

? < Back Next > Cancel Finish







- Run As Java Application

The Spring Boot logo consists of a series of nested and overlapping brackets (curly braces) in various sizes and orientations, creating a complex, organic shape. Below the logo, the text ":: Spring Boot ::" is centered in a bold, black font. To the right of the main logo, the text "(v2.5.5)" is enclosed in a pair of parentheses.

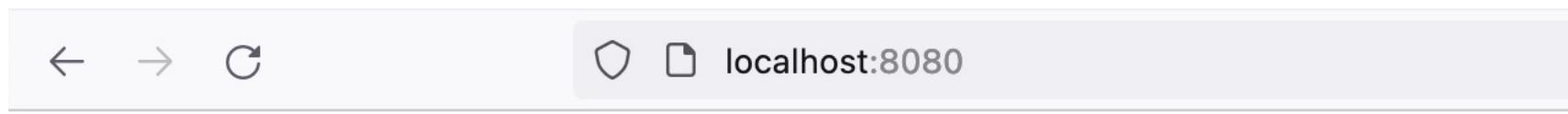
```
2021-09-23 16:47:12.627 INFO 57546 --- [           main] c.e.demo.MyFirstSpringBootApplication : Starting MyFirstSpringBootApplication using Java 11.0.11 on Air-von-Micha
2021-09-23 16:47:12.629 INFO 57546 --- [           main] c.e.demo.MyFirstSpringBootApplication : No active profile set, falling back to default profiles: default
2021-09-23 16:47:14.818 INFO 57546 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-09-23 16:47:14.831 INFO 57546 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-09-23 16:47:14.831 INFO 57546 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.53]
2021-09-23 16:47:14.931 INFO 57546 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2021-09-23 16:47:14.932 INFO 57546 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2181 ms
2021-09-23 16:47:15.894 INFO 57546 --- [           main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actuator'
2021-09-23 16:47:15.980 INFO 57546 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-09-23 16:47:16.001 INFO 57546 --- [           main] c.e.demo.MyFirstSpringBootApplication : Started MyFirstSpringBootApplication in 4.239 seconds (JVM running for 4.239)
2021-09-23 16:47:56.296 INFO 57546 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-09-23 16:47:56.296 INFO 57546 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-09-23 16:47:56.301 INFO 57546 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 5 ms
```



- Run As Spring Boot

MyFirstSpringBoot - MyFirstSpringBootApplication [Spring Boot App] /Library/Java/JavaVirtualMachines/jdk-11.0.11.jdk/Contents/Home/bin/java (23.09.2021, 16:49:18)

```
2021-09-23 16:49:20.541 INFO 57608 --- [           main] c.e.demo.MyFirstSpringBootApplication : Starting MyFirstSpringBootApplication using Java 11.0.11 on Air-von-Micha
2021-09-23 16:49:20.543 INFO 57608 --- [           main] c.e.demo.MyFirstSpringBootApplication : No active profile set, falling back to default profiles: default
2021-09-23 16:49:21.629 INFO 57608 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-09-23 16:49:21.637 INFO 57608 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-09-23 16:49:21.638 INFO 57608 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.53]
2021-09-23 16:49:21.698 INFO 57608 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2021-09-23 16:49:21.698 INFO 57608 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1105 ms
2021-09-23 16:49:22.297 INFO 57608 --- [           main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actuator'
2021-09-23 16:49:22.350 INFO 57608 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-09-23 16:49:22.366 INFO 57608 --- [           main] c.e.demo.MyFirstSpringBootApplication : Started MyFirstSpringBootApplication in 2.312 seconds (JVM running for 3
```



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Sep 23 16:47:56 CEST 2021

There was an unexpected error (type=Not Found, status=404).

Simplen Controller hinzufügen



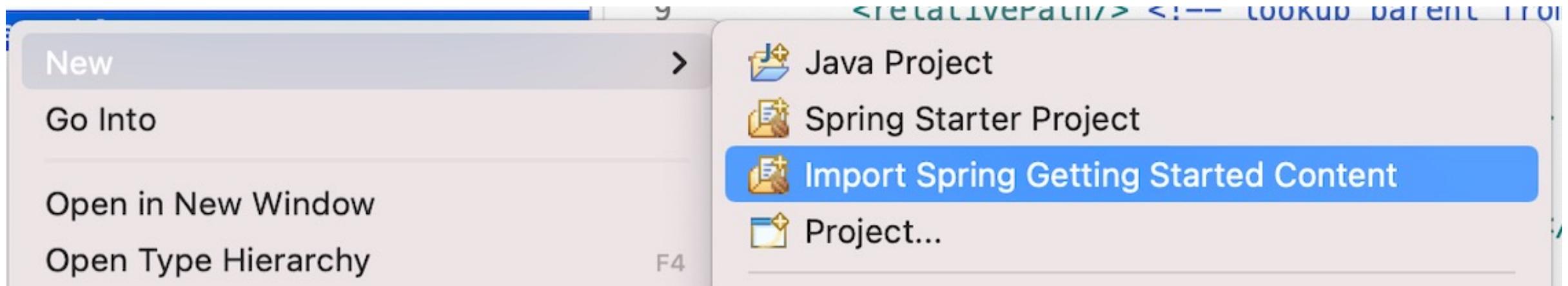
```
package com.example.demo;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorldController
{
    @RequestMapping("/")
    public String hello()
    {
        return "Hello Spring Workshop Participants";
    }
}
```



Cooles Feature





Simple Extensions ... Dev Tools + Actuator



Health Check



localhost:8080/actuator/health

JSON Rohdaten Kopfzeilen

Speichern Kopieren Alle einklappen Alle ausklappen | JSON durchsuchen

status: "UP"



Spring Initializr



**Project** Maven Project Gradle Project**Language** Java Kotlin Groovy**Spring Boot** 2.6.0 (SNAPSHOT) 2.6.0 (M2) 2.5.6 (SNAPSHOT) 2.5.5 2.4.12 (SNAPSHOT) 2.4.11**Project Metadata**

Group com.example

Artifact demo

Name demo

Description Demo project for Spring Boot

Package name com.example.demo

Packaging Jar WarJava 17 11 8**Dependencies**

No dependency selected

ADD DEPENDENCIES... ⌘ + B**GENERATE** ⌘ + ↵**EXPLORE** CTRL + SPACE**SHARE...**



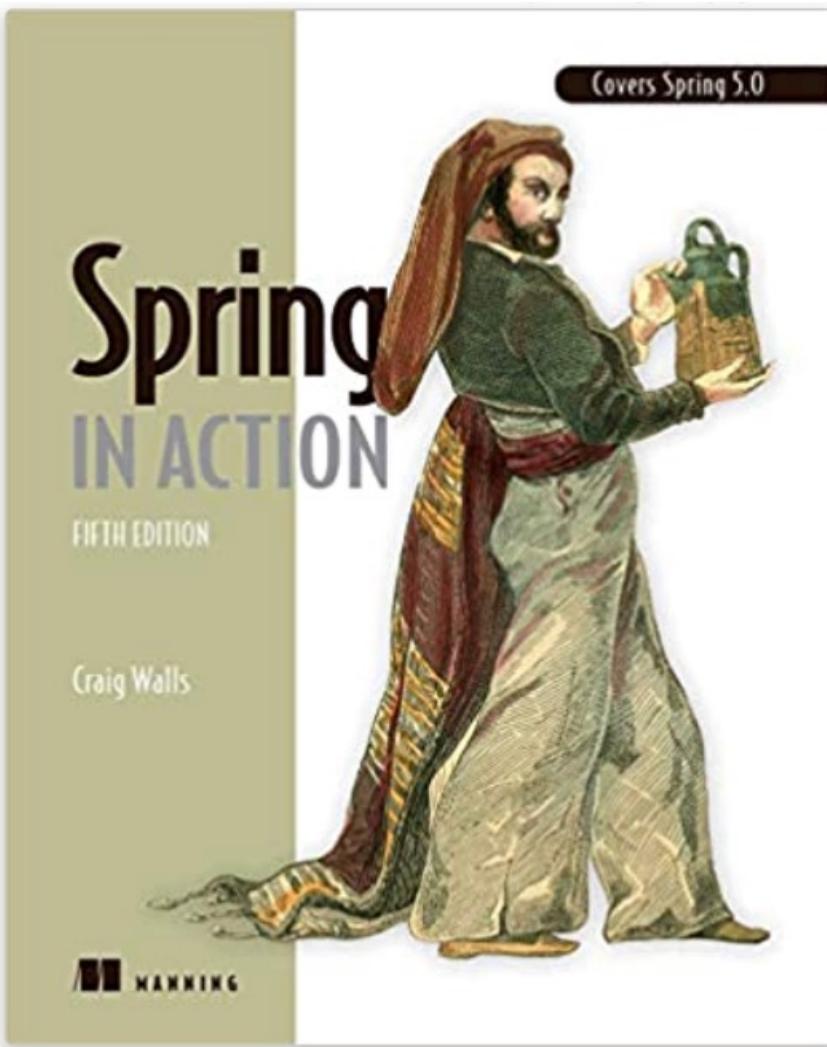
Demo



Questions?



Empfehlenswerte Bücher





Thank You