

Verteilte Systeme Master Lab

christian.zirpins@hs-karlsruhe.de

Microservice Basistechnologien



Hochschule Karlsruhe
Technik und Wirtschaft

UNIVERSITY OF APPLIED SCIENCES



Zeitplan

VERTEILTE SYSTEME MASTER LABOR - SOMMERSEMESTER 2020

				Präsenztermine (Onlinetreffen) jeweils Mi 8:00 Uhr		
#	KW	Tag	Laboraufgabe	Termin	Form	Abgabe
1	12					
2	13					
3	14		A1+A2 Analyse der Legacy Anwendung und Microservice Architekturentwurf	Seminar: Microservice Architekturen	Mediacast	
4	15			Seminar: Web Shop Basistechnologien	Mediacast	
5	16			Seminar: Microservice Basistechnologien	Mediacast	
6	17	22.4.		Lab: Analyse Legacy App / Microservice Entwurf	Onlinetreffen	
7	18	29.4.		Meetup: Abgabe von Analysen und Entwürfen	Onlinetreffen	<u>System- und REST-Modelle</u>
8	19		A3 Microservice Implementierung	Seminar: Microservice Middleware	Mediacast	
9	20	13.5.		Lab: Web Shop REST Microservices	Onlinetreffen	
10	21	20.5.		Lab: Microservice Infrastruktur mit Netflix OSS	Onlinetreffen	
11	22	27.5.		Meetup: Web Shop Microservices	Onlinetreffen	<u>Web Shop V2</u>
	23			<i>Pfingsten</i>		
12	24		A4 Web Shop Implementierung	Seminar: Microservice Security	Mediacast	
13	25	17.6.		Lab: Microservice Security mit OAuth	Onlinetreffen	
14	26	24.6.		Lab: Frontend integration	Onlinetreffen	
15	27	1.7.		Meetup: Web Shop V3	Onlinetreffen	<u>Web Shop V3</u>

Struktur — Microservice Basistechnologien

Service Enablement

REST API Spezifikation

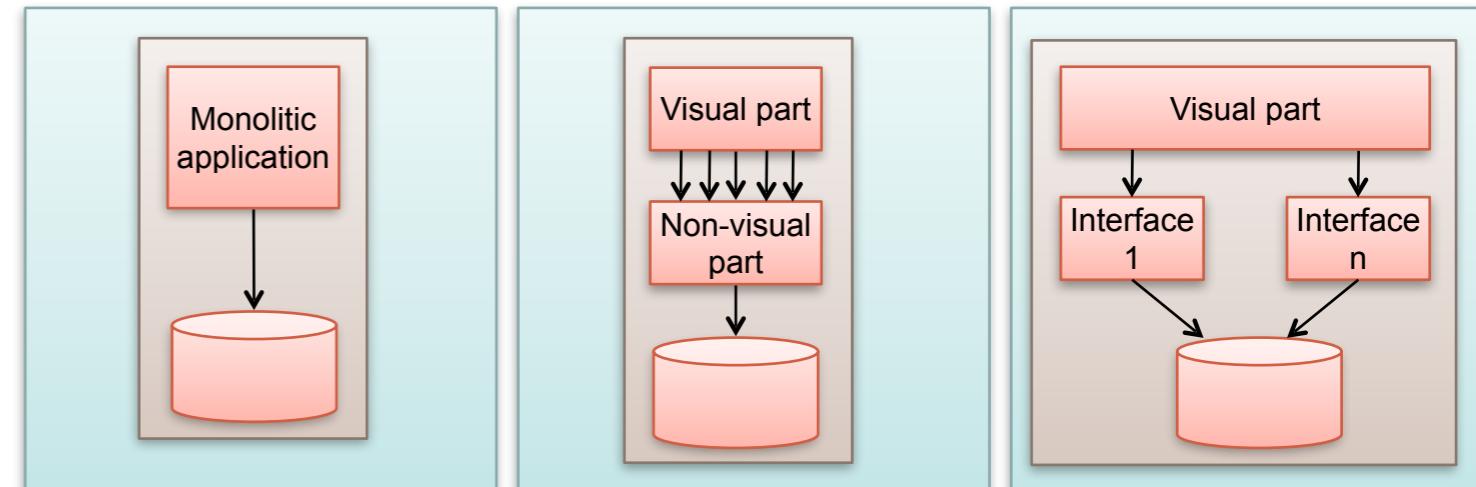
Spring REST Framework

Service Enablement

- Unter *Service Enablement* versteht man einen *Prozess zur Konstruktion von Diensten*, um die Funktionalität einer bestehenden Anwendung zu kapseln.
- Anwendungen können auf beliebigen Technologien basieren, z.B.
 - *monolithische Client/Server Systeme* oder
 - *verteilte Objektsysteme* z.B. basierend auf CORBA
- Service Enablement umfasst mehr als nur eine bestehende Programmierschnittstelle *zu verpacken und bereitzustellen*.

Service Enablement für monolithische Anwendungen

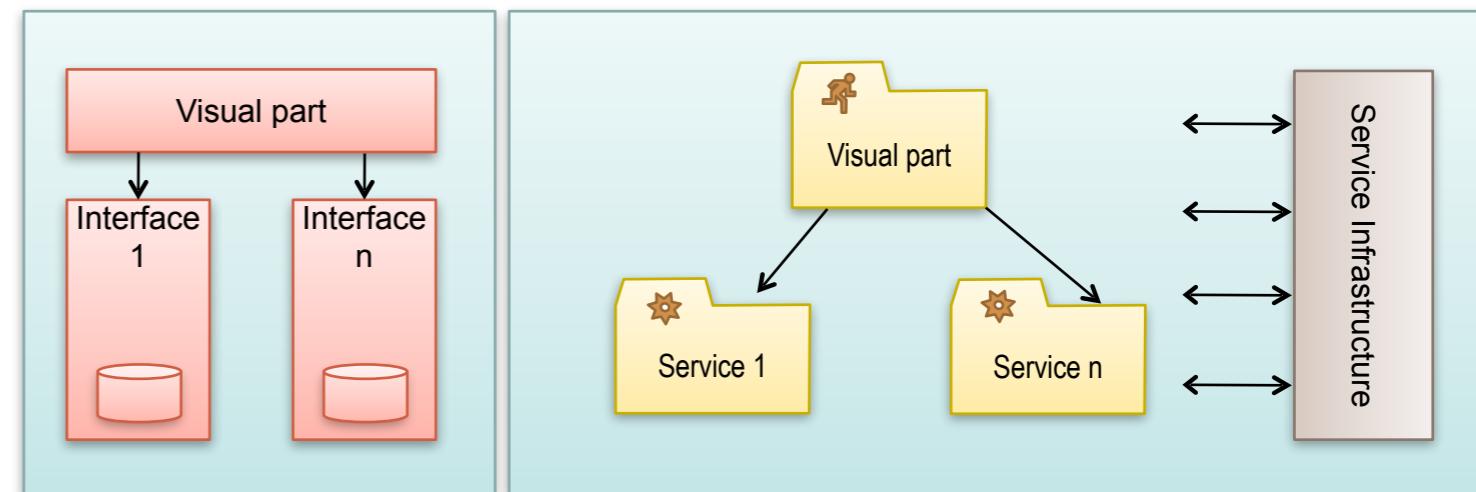
- Zunächst, *separiere visuelle und nicht-visuelle Anteile*.
- Dann *gruppiere die Interaktionen* beider Teile i.B.a. geschäftliche Begriffe und definiere Interfaces für nicht-visuelle Anteile.
- *Separiere Implementierung und Daten* pro Interface.
- *Migriere* die Anwendung in die Service Infrastruktur.
- Suche möglichst *wieder verwendbare* Dienste
 - Nutze diese in möglichst vielen Anwendungen.
 - Nutze *Dienstverzeichnisse* um gleiche Dienste zu finden und zu verwalten.



a) Startpunkt ist eine monolithische Anwendung

b) Der visuelle Teil wird vom nicht-visuellen Teil der Anwendung getrennt

c) Vertikale Schnitte führen zu mehreren Schnittstellen

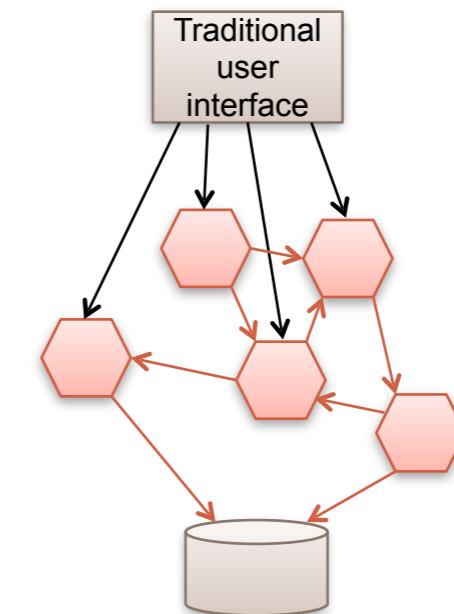


d) Das Datenmodell wird in separate Submodelle zerteilt

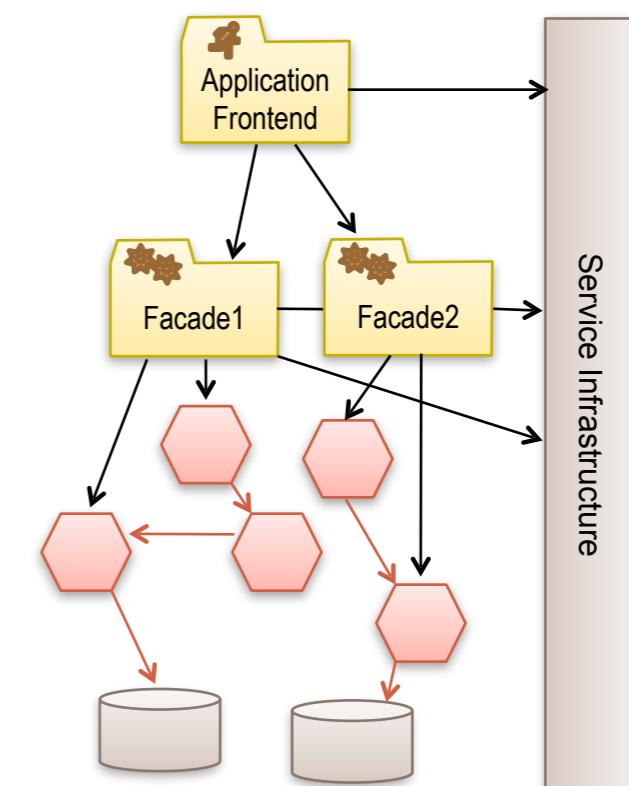
e) Die Anwendung wird in Anwendungsfrontend(s) und -dienste getrennt. Die Service Infrastruktur entkoppelt die Anwendungskomponenten von der darunterliegenden Verteilungstechnik.

Service Enablement für verteilte Objektsysteme

- Verteilte Objektsysteme beruhen oft auf *feingranularen Interaktionsmustern*.
- Dann können *Fassaden* genutzt werden.
- Anwendungen werden so geändert, dass sie *ausschließlich Fassaden* nutzen.
- Dann werden Objektbeziehungen durch *loose Dienstkopplung* ersetzt.
- Die Implementierungen der Dienste sollten möglichst *lokal gebündelt* werden.
- Die Fassaden sollten ihre *Daten jeweils exklusiv* nutzen.



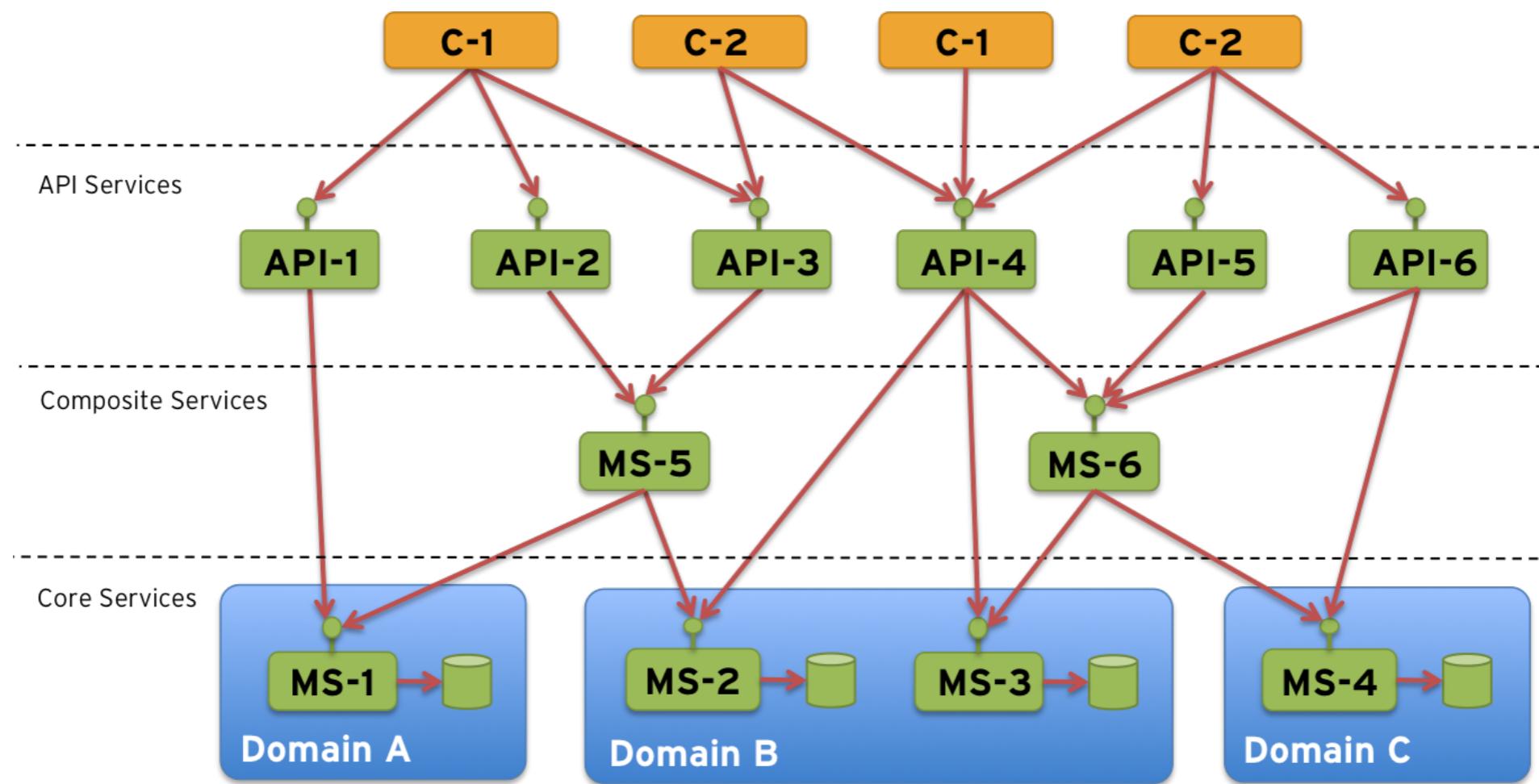
a) Feingranulare Interaktionsmuster



b) Grobgranulare Interaktionsmuster

Service Enablement mit API Gateways

- Sollen Anwendungen nach dem Microservice Architekturstil entstehen, dann wird eine zusätzliche **API-Ebene** eingezogen.
- **API-Gateways** aggregieren **Microservice APIs** zur Optimierung der Effizienz und Kontrolle des Zugriffs für verschiedene Clienttypen.



Struktur — Microservice Basistechnologien

Service Enablement

REST API Spezifikation

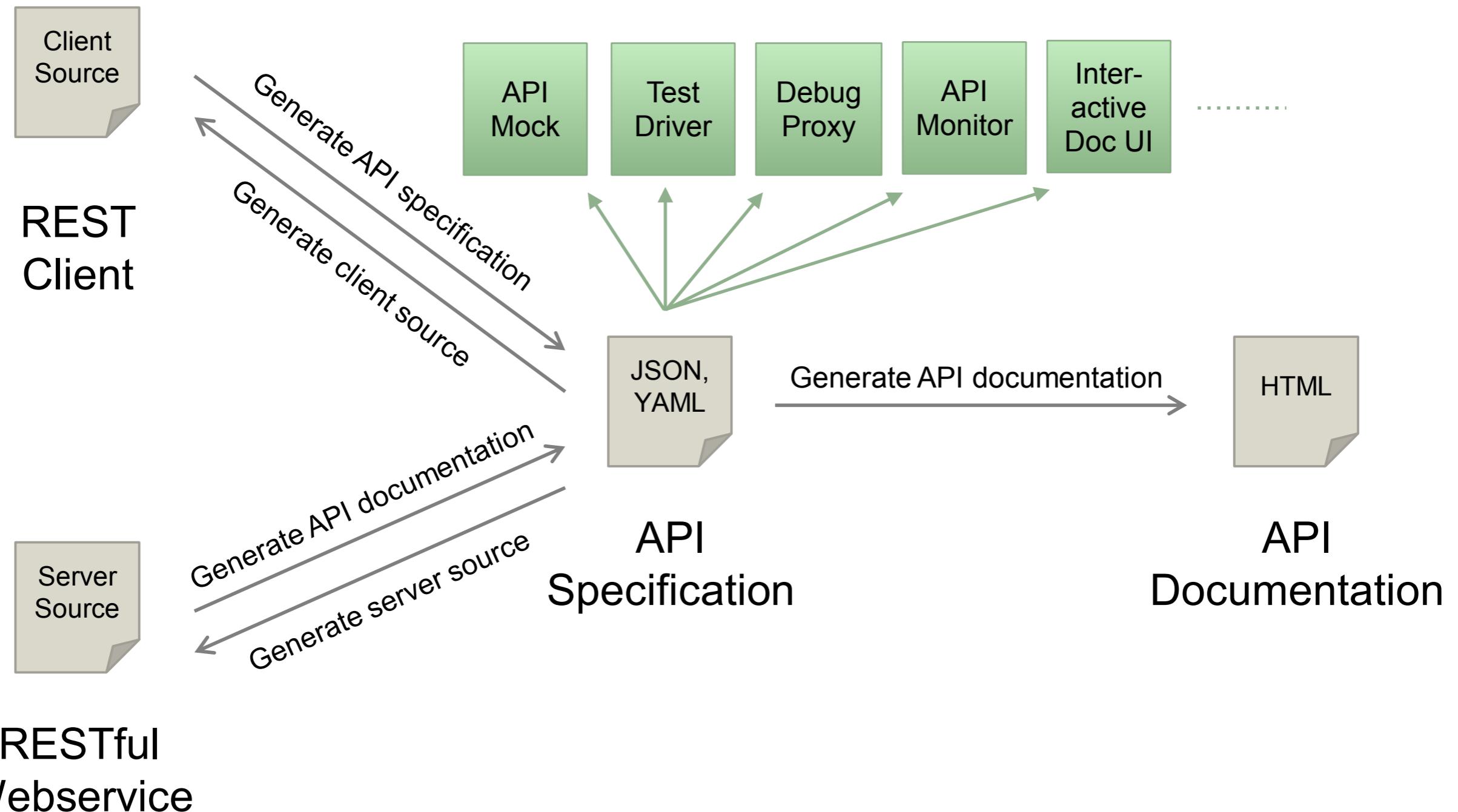
- Spezifikation von REST APIs
- Übersicht von Ansätzen zur API Spezifikation
- Beispiel: RAML

Spring REST Framework

Spezifikation von REST APIs

- Der **Entwurf** einer API wird in einer **Spezifikation** erfasst.
- Die Spezifikation dient auch als **Dokumentation** für Entwickler von **Client Anwendungen**, die eine API verwenden möchten.
- Dokumente und dynamische Dienste bilden ein **gemeinsames Modell**:
 - Im Web haben die **Dokumente**, die die Ressourcen beschreiben, selbst den Status eigenständiger **Ressourcen**.
 - Die **Navigation** innerhalb der Dokumentation, innerhalb der dynamischen Ressourcen und dazwischen kann durch **Hypermedia** erfolgen.
 - Die **Schnittstelle** für die Bearbeitung und das Erzeugen von Ressourcen kann ebenfalls eine eigene **Ressource** sein.

API Spezifikation als Systemkomponente



Abgeleitet von René Milzarek, „Swagger and RAML – API centering dev“, Webtech Seminar, sebis, TUM, 13.04.2015

Übersicht von Ansätzen zur API Spezifikation

Sprache	Entwickler
<u>Web Services Description Language (WSDL)</u>	(W3C)
<u>Web Application Description Language (WADL)</u>	(W3C)
<u>CloudRail</u>	licobo GmbH
<u>Google Cloud Endpoints</u>	Google
<u>Open Data Protocol (OData)</u>	OASIS Standard, Microsoft
<u>GraphQL</u>	Facebook
<u>OpenAPI Specification</u>	Open API Initiative (OAI), ehem. "Swagger"
<u>RESTful Service Description Language (RSDL)</u>	Michael Pasternak
<u>Hydra Core Vocabulary (Hydra)</u>	Hydra W3C Community Group
<u>RESTful API Modeling Language (RAML)</u>	Mulesoft
<u>API Blueprint</u>	Apiary
<u>API Builder</u>	HBC, Flow Commerce
<u>I/O Docs</u>	Mashery
<u>Apache Avro</u>	
<u>Barrister</u>	James Cooper
<u>SERIN - Semantic Restful Interfaces</u>	Bruno Muniz et al.

Abgeleitet von https://en.wikipedia.org/wiki/Overview_of_RESTful_API_Description_Languages

Übersicht von Ansätzen zur API Spezifikation

Sprache	Entwickler
<u>Web Services Description Language (WSDL)</u>	(W3C)
<u>Web Application Description Language (WADL)</u>	(W3C)

Die z.Z. wichtigsten Kandidaten

	Sponsor	Initial commit	Latest stable release	Stable release date	Software license	Format	Open Source	Code generation (client)	Code generation (server)
RAML	MuleSoft	September, 2013	1.0	May 16, 2016	Apache 2.0	YAML	Yes	Yes	Yes
API Blueprint	Oracle Apiary	April, 2013	1A9	Jun 8, 2015	MIT	Mark down	Yes	limited	Yes
OpenAPI	Open API Initiative (OAI)	July, 2011	3.0.2	Oct 8, 2018	Apache 2.0	JSON or YAML	Yes	Yes	Yes

<u>I/O Docs</u>	<u>Mashery</u>
<u>Apache Avro</u>	
<u>Barrister</u>	James Cooper
<u>SERIN - Semantic Restful Interfaces</u>	Bruno Muniz et al.

Abgeleitet von https://en.wikipedia.org/wiki/Overview_of_RESTful_API_Description_Languages

Übersicht von Ansätzen zur API Spezifikation

Sprache	Entwickler
<u>Web Services Description Language (WSDL)</u>	(W3C)
<u>Web Application Description Language (WADL)</u>	(W3C)

Wichtiger Aspekt: Daten-Beschreibungssprachen

- **XML Schema** (<https://www.w3.org/XML/Schema>)
 - "Klassische" Schema Sprache für Web Services (aber auch RAML)
- **JSON Schema / JSON Hyper-Schema** (<http://json-schema.org>)
 - Schema Sprache für JSON Daten / JSON Hypermedia
 - Verwendet in allen wichtigen API Sprachen (OpenAPI, RAML, API Blueprint)
- **json:api** (<http://jsonapi.org/>), **HAL** ([Hypertext Application Language](#))
 - Formatvorgaben für JSON-basierte APIs (z.B. HATEOAS "Relationen")

I/O Docs	Mashery
Apache Avro	
Barrister	James Cooper
SERIN - Semantic Restful Interfaces	Bruno Muniz et al.

Abgeleitet von https://en.wikipedia.org/wiki/Overview_of_RESTful_API_Description_Languages

API Dokumentation mit OpenAPI (Swagger)

- Zu jedem URI-Muster erfolgt die **Spezifikation** von...
 - unterstützten HTTP-Verben,
 - einer (prosaischen) **Beschreibung**,
 - der **Parameter** (aus Query-Parametern und URL Pfad),
 - unterstützten **MIME Types**,
 - möglichen **Antworten** mit HTTP-Status und
 - ggf. den **JSON-Schemas** für die Anfrage und Antwort.
- **Beschreibungen** können **aus vielen integrierten REST Frameworks generiert** werden.
 - Ein Schwerpunkt liegt auf der **Dokumentation bestehender APIs**.
- Ein **Editor/UI** erlaubt die **API-Beschreibung in YAML** und zu jedem Verb kann ein Request direkt aus der Dokumentation abgesendet werden.
- Es gibt **Codegeneratoren** für Clients und Server.

swaggerhub.com

Search All APIs My APIs zich0001

VALID { }

Collaboration +

A sample swagger spec

created: tue, 27 oct 2015 18:16:46 gmt, modified: wed, 28 oct 2015 14:27:02 gmt

Generate: Client Server Download...
Publish Save 1 Reset

Editor Version Info Interactive API Docs

```

1  swagger: '2.0'
2  info:
3    version: '1'
4    title: User API
5    description: A sample swagger spec
6  basePath: /v1
7  schemes:
8    - http
9  consumes:
10   - application/json
11  produces:
12   - application/json
13  paths:
14    /users/:
15      get: →
16      post: →
17    /users/{id}:
18      delete: →
19      get:
20        description: get user details
21        tags:
22          - user
23        parameters:
24          - name: id
25            in: path
26            type: integer
27            format: int64
28            required: true
29            description: Id of user
30        responses:
31          '200':
32            description: User found
33            schema:
34              $ref: '#/definitions/User'
35  definitions:
36    User: →
37    Users: →
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87

```

/users/

GET /users/

POST /users/

/users/{id}

DELETE /users/{id}

GET /users/{id}

user

Description

get user details

Parameters

Name	Located in	Description	Required	Schema
id	path	Id of user	Yes	integer (int64)

Responses

Code	Description	Schema
200	User found	►User { }

Try this operation



Beispiel einer API Spezifikation mit Editor und Dokumentation von SwaggerHub

API Design mit RAML

- Über das Generieren von Dokumentation hinaus fördert RAML das **Teilen von guten Lösungen**, um so Wiederholung zu vermeiden.
 - **Ressource Types** und **Traits** erlauben die **Wiederverwendung** von Struktur und **Verhaltensmustern** ähnlich wie bei OO-Vererbung.
 - Die Struktur der Beschreibung ist sonst **ähnlich zu OAI/Swagger**, unterstützt aber **verschachtelte Ressourcen** (z.B. Container und ihre Elemente).
 - Insgesamt liegt der Schwerpunkt von RAML eher auf dem **API-Design**.
- RAML setzt als Beschreibungsformat auf **YAML** und definiert einen eigenen Medientyp **application/raml+yaml**.
 - Für die Beschreibung kann Markdown zur Formatierung genutzt werden.
- RAML ermöglicht auch die **Generierung** von Client- und Servercode, zum Beispiel als Mock-Server.
- Für RAML gibt es eigene **Editoren** (oder IDE/Editor Plugins) sowie interaktive **Dokumentationsformate** (API Console/Notebook).

RAML (RESTful API Modeling Language)



RESTful API Modeling Language

raml.org

REST Spezifikation: Web Ressourcen (04/2019)

RAML

- Home: <http://raml.org>
- SaaS Tools (Anypoint/MuleSoft): <https://anypoint.mulesoft.com/apiplatform/>
- Sources: <https://github.com/raml-org/raml-spec>
 - Beispiele: <https://github.com/raml-org/raml-tutorial-200>

OpenAPI (Swagger)

- Home: <https://www.openapis.org/>
- SaaS Tools (SwaggerHub/SmartBear): <https://swaggerhub.com>
- Sources: <https://github.com/OAI/OpenAPI-Specification>
 - Beispiele: <https://github.com/OAI/OpenAPI-Specification/tree/master/examples/v3.0>

API Blueprint

- Home: <https://apiblueprint.org>
- SaaS Tools (Oracle Apiary): <https://apiary.io>
- Sources: <https://github.com/apiaryio/api-blueprint>
 - Beispiele: <https://github.com/apiaryio/api-blueprint/tree/master/examples>

Struktur — Microservice Basistechnologien

Serviece Enablement

REST Spezifikation

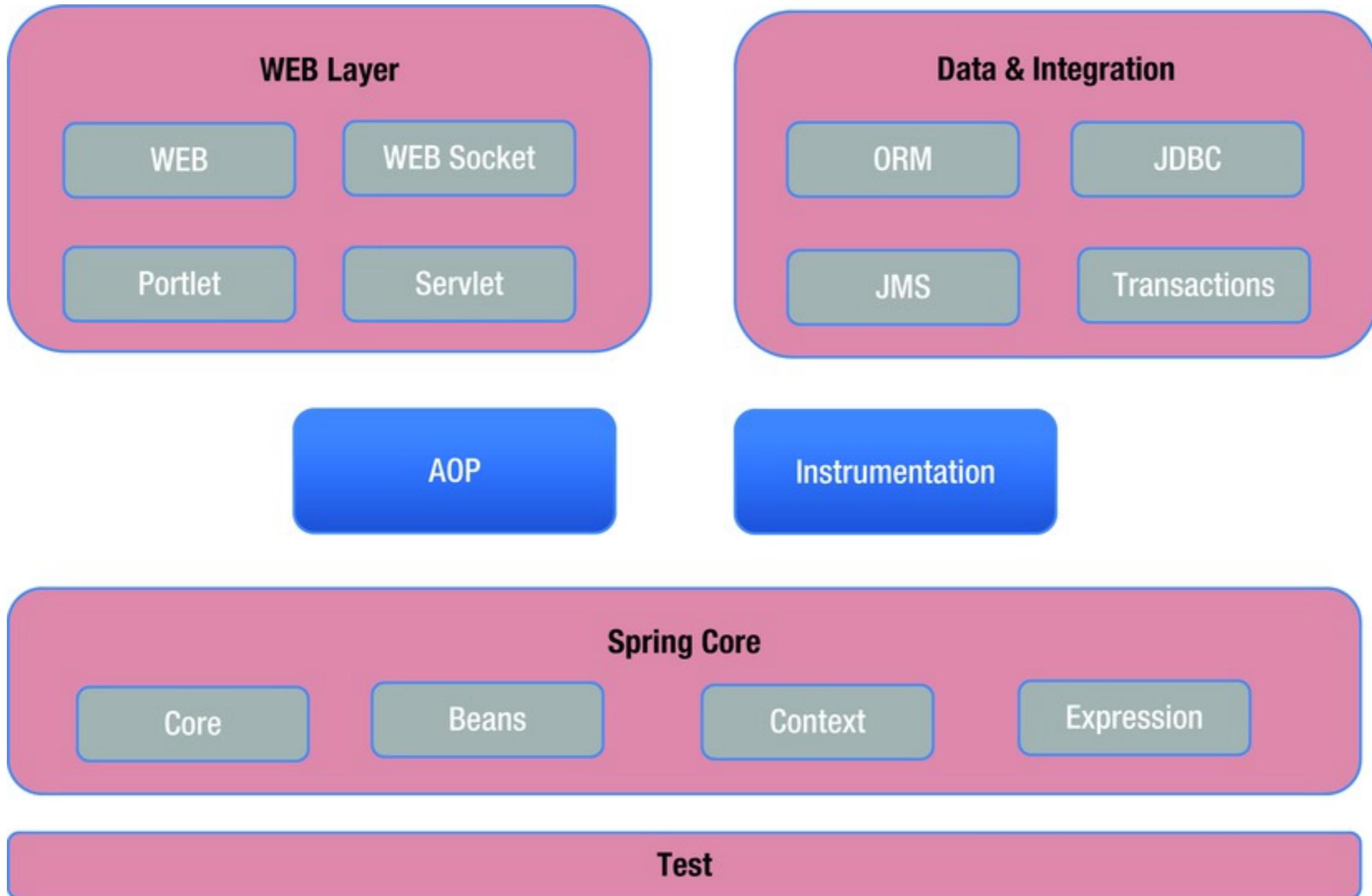
Spring REST Framework

- Spring Web MVC
- REST Services Implementieren

Spring in a Nutshell

- Das Spring Framework ist der de facto Standard zur Entwicklung von Java/Java EE-basierten Enterprise Anwendungen.
 - Das Framework wurde erstmals 2002 von Rod Johnson vorgestellt.
- Das Spring Framework ...
 - ...enthält ein **Dependency Injection** (DI) Model zur **Reduktion von schematischem Code** bei der Anwendungsentwicklung,
 - ...unterstützt **Aspect Oriented Programming** (AOP) zur Implementierung von **Crosscutting Concerns (CCC)**, and
 - ...vereinfacht die **Integration** mit anderen **Frameworks** und Technologien.
- Das Spring Framework beinhaltet **Module**, die spezifische Dienste anbieten, u.a. für **Datenzugriff**, **Instrumentation**, **Messaging**, **Testen** und **Web Integration**.
 - Durch die Modularität von Spring können beliebige Module (und nur diese) je nach individuellen Anforderungen gewählt werden.

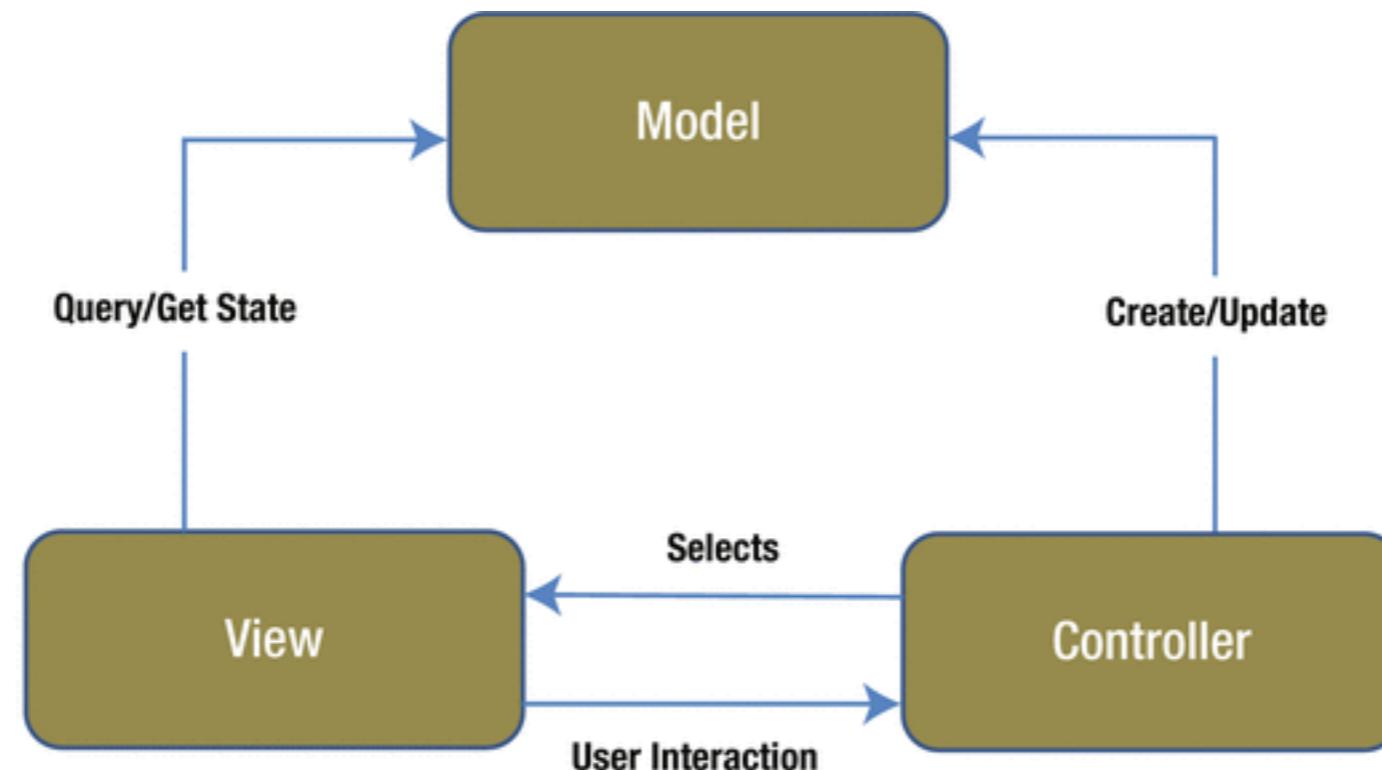
Spring Framework Module



Aus [Varanasi 2015]

Spring Web MVC

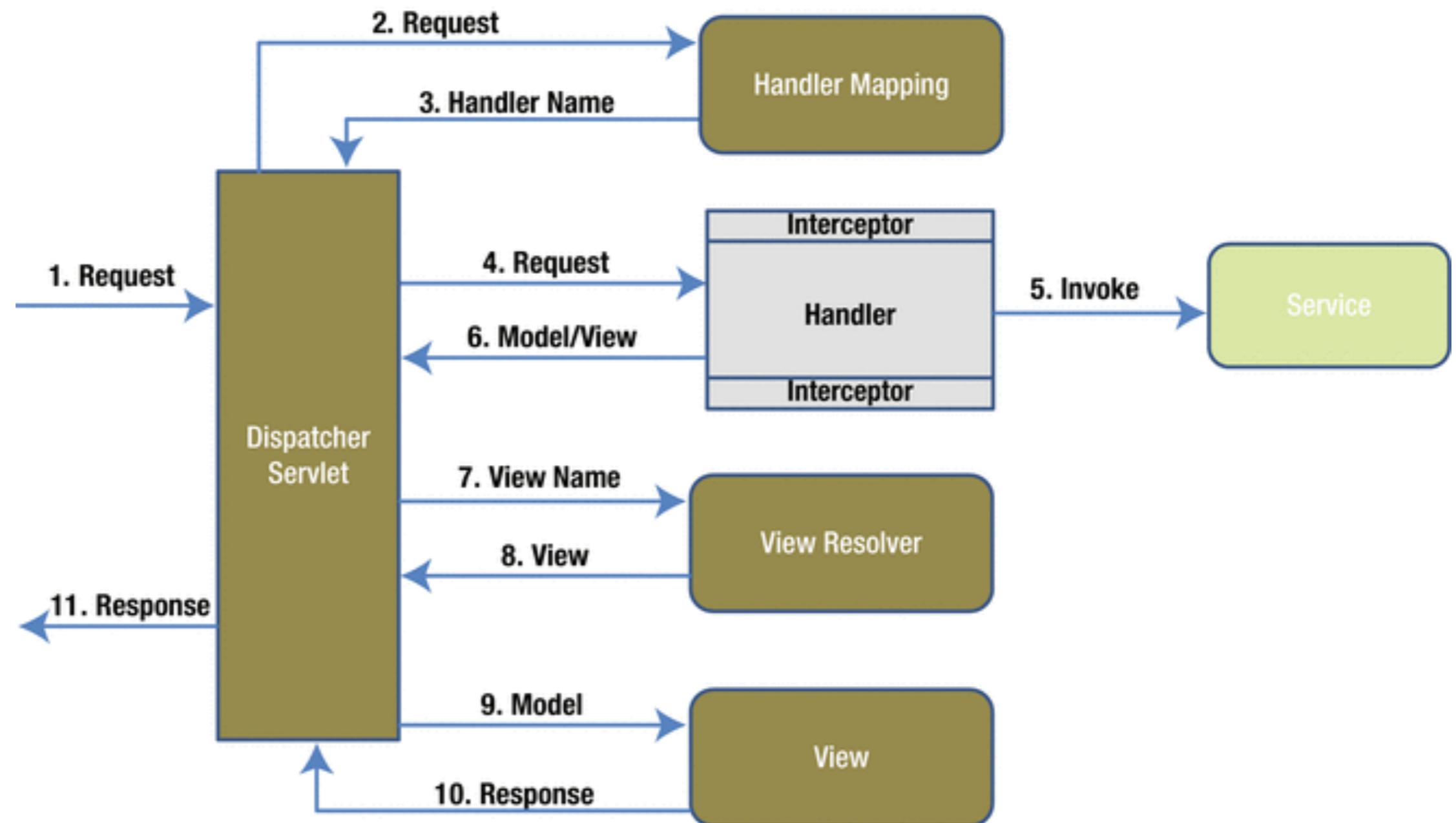
- **Spring Web MVC** ist ein Teil des **Spring Web Modules** zur Entwicklung Web-basierter Anwendungen.
- Das Modul basiert auf der **Model-View-Controller** Architektur.
 - Ein **Modell** repräsentiert Daten oder Zustand.
 - Ein **View** bildet eine visuelle Repräsentation des Modells.
 - Ein **Controller** wickelt Benutzeraktivitäten ab.



Aus [Varanasi 2015]

Spring Web MVC Architektur

- Die Spring Web MVC Implementierung basiert auf dem Dispatcher Servlet, das als Einstiegspunkt für die Bearbeitung von Anfragen dient.



Web MVC Komponenten: *Controller*

- Controller werden mit dem Stereotype `org.springframework.stereotype.Controller` deklariert.
 - **Stereotypes** kennzeichnen Rollen oder Verantwortlichkeiten von Klassen oder Schnittstellen.

```
@Controller
public class HomeController {
    @RequestMapping("/home.html")
    public String showHomePage() {
        return "home";
    }
}
```

- **@Controller** kennzeichnet `HomeController` als MVC Controller.
- **@RequestMapping** bildet Web Requests auf Handler Klassen und Methoden ab.

Web MVC Komponenten: *Model*

- Das `org.springframework.ui.Model` Interface dient als Container für Modellattribute.

```
public interface Model {  
    Model addAttribute(String attributeName, Object attributeValue);  
    Model addAttribute(Object attributeValue);  
    Model addAllAttributes(Collection<?> attributeValues);  
    Model addAllAttributes(Map<String, ?> attributes);  
    Model mergeAttributes(Map<String, ?> attributes);  
    boolean containsAttribute(String attributeName);  
    Map<String, Object> asMap();  
}
```

- Die Methoden `addAttribute` und `addAllAttributes` Methoden fügen Attribute zum Model Objekt hinzu.

Web MVC Komponenten: *Model*

- Ein Controller kann mit einem Model arbeiten, indem dieses als Methodenparameter deklariert wird.

```
@RequestMapping("/home.html")
public String showHomePage(Model model) {
    model.addAttribute("currentDate", new Date());
    return "home";
}
```

- Alternativ kann java.util.Map verwendet werden.

```
@RequestMapping("/home.html")
public String showHomePage(Map model) {
    model.put("currentDate", new Date());
    return "home";
}
```

Web MVC Komponenten: View

- Web MVC unterstützt verschiedene View Techniken (JSP, Velocity, Freemarker, XSLT) mit dem org.springframework.web.servlet.View Interface.

```
public interface View {  
    String getContentType();  
    void render(  
        Map<String, ?> model,  
        HttpServletRequest request,  
        HttpServletResponse response) throws Exception;  
}
```

- Konkrete Implementierungen des View Interfaces realisieren das Rendering der Response. Beispielsweise:

org.springframework.web.servlet.view.json.MappingJackson2JsonView	View Implementierung zur Kodierung von ModellAttributen in JSON .
org.springframework.web.servlet.view.InternalResourceView	View Implementierung zur Delegierung von Requests an eine JSP Seite.

Web MVC Komponenten: **@RequestMapping**

- Die **@RequestMapping** Annotation bildet einen Web Request auf eine Handler Klasse oder Handler Methode ab.
- **@RequestMapping** besitzt verschiedene Attribute, die die Abbildung einschränken:
 - **method** - Einschränkung auf HTTP Methoden wie GET, PUT etc.
 - **produces** - Einschränkung auf einen erzeugten Mediatype
 - **consumes** - Einschränkung auf einen empfangenen Mediatype
 - **headers** - Einschränkung auf bestimmte Header
 - **name** - Benennung einer Abbildung
 - **params** - Einschränkung auf Parameternamen und Wert

```
@RequestMapping(value="/saveuser.html", method=RequestMethod.POST)
public String saveUser(@RequestParam String username,
                      @RequestParam String password) {
    // Save User logic
    return "success";
}
```

Flexible Handler Parameter und Return Typen

- Mögliche **Argumente** von Handler Methoden
 - HttpServletRequest/HttpServletResponse - rohe Nutzerdaten
 - HttpSession - zugehörige Session
 - Command Object - POJO gefüllt mit übertragenen Daten
 - HttpEntity<?> - Repräsentation des HTTP Requests
 - Principal - Repräsentation des Nutzers (Java.security)
- Mögliche **Return Typen** von Handler Methoden
 - String - View Name
 - View - View Repräsentation
 - HttpEntity<?> - HTTP Response Repräsentation
 - HttpHeaders - Rückgabe Header, Body bleibt leer
 - Pojo - Java Objekt als Model Attribut

Web MVC Komponenten: `@RequestParam`

- Die `@RequestParam` Annotation bindet Servlet Request Parameter an Handler/Controller Methodenparameter.
- Der Request Parameterwert wird automatisch in den spezifizierten Methodenparameter konvertiert.

```
@RequestMapping("/search.html")
public String search(@RequestParam String query,
    @RequestParam("page") int pageNumber) {
    ...
    return "search";
}
```

- Falls der Parameter nicht im Request enthalten ist, wird eine Exception geworfen.

Web MVC Komponenten: *Pfad Variablen*

- Die `@RequestMapping` Annotation unterstützt dynamische URIs mit URI Templates.
- **URI Templates** sind URIs mit Platzhaltern/Variablen.
- Durch die `@PathVariable` Annotation können die Platzhalter als Methodenparameter genutzt werden.

```
@RequestMapping("/users/{username}")
public User getUser(@PathVariable("username") String username) {
    User user = null;
    // Code to construct user object using username
    return user;
}
```

Struktur — Microservice Basistechnologien

Serviece Enablement

REST Spezifikation

Spring REST Framework

- Spring Web MVC
- REST Services Implementieren

Spring IO Platform



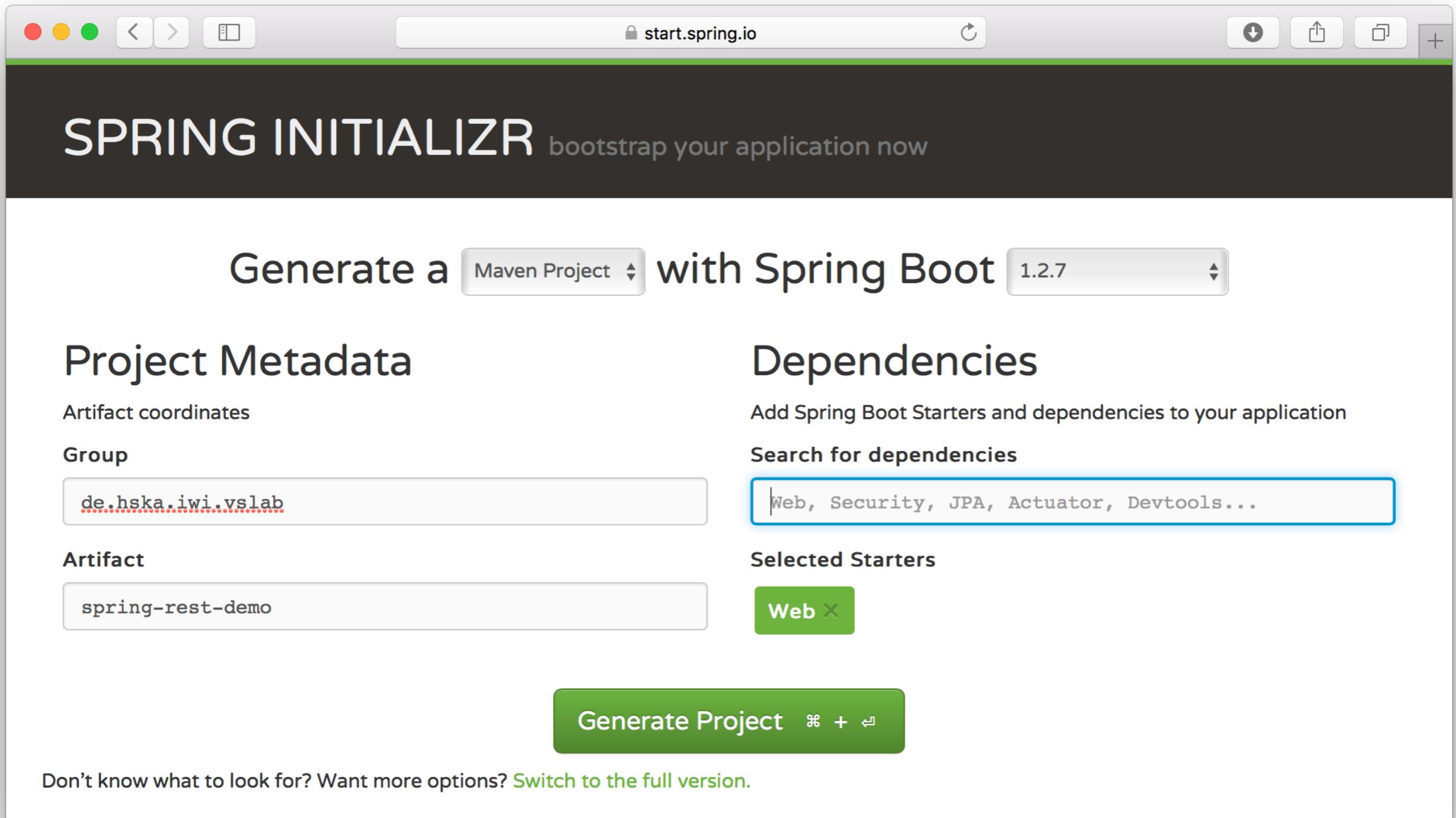
Spring Boot Runtime

- **Spring Boot** vereinfacht Bootstrapping ("zum Laufen bringen") von Spring Anwendungen durch **Starter Project Templates**.
 - Enthalten geordnete Abhängigkeiten für spezifischen Projekt **Features**.
 - Z.B. **JPA Feature**: JPA, Hibernate und Spring JARs werden hinzugefügt.
- Spring Boot **Standardkonfigurationen** erleichtern Entwicklung.
 - Wenn Spring Boot z.B. JPA und MySQL JARs im Klassenpfad findet, würde es automatisch eine JPA Persistence Unit konfigurieren.
- Spring Boot unterstützt **Standalone Anwendungen** mit eingebetteten **Jetty/Tomcat Servern**, die auf jedem Rechner mit Java laufen.
- Produktivfunktionen für **Metriken** und **Health Checks**.

Spring Tools

- Spring Boot unterstützt zwei Build Systeme: **Maven** und **Gradle**.
- **Spring Tool Suite** bzw. **STS**
 - Eclipse-basierte Entwicklungsumgebung mit Spring Tooling
 - STS kann von Pivotal's Website bezogen werden:
<https://spring.io/tools/sts/all>
- Drei Möglichkeiten um neue Projekte zu generieren:
 - Spring Boot's **Starter Website** (<http://start.spring.io>)
 - **Spring Tool Suite (STS)** IDE
 - Boot **command line interface (CLI)**

SPRING INITIALIZR



The screenshot shows the Spring Initializr web application at start.spring.io. The title "SPRING INITIALIZR" is displayed with the subtitle "bootstrap your application now". The main heading "Generate a **Maven Project** with Spring Boot **1.2.7**" is centered.

Project Metadata

- Artifact coordinates: `de.hska.iwi.vslab`
- Artifact: `spring-rest-demo`

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies: `Web, Security, JPA, Actuator, Devtools...`

Selected Starters: `Web`

Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)

Spring Boot Projekt Struktur

```
.  
├── mvnw  
├── mvnw.cmd  
├── pom.xml  
└── src  
    ├── main  
    │   ├── java  
    │   │   └── de  
    │   │       └── hska  
    │   │           └── iwi  
    │   │               └── vslab  
    │   │                   └── SpringRestDemoApplication.java  
    │   └── resources  
    │       ├── application.properties  
    │       ├── static  
    │       └── templates  
    └── test  
        └── java  
            └── de  
                └── hska  
                    └── iwi  
                        └── vslab  
                            └── SpringRestDemoApplicationTests.java
```

Maven POM

```
<project ... >
  <groupId>de.hska.iwi.vslab</groupId>
  <artifactId>spring-rest-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>spring-rest-demo</name>
  ...
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    ...
  </parent>
  ...
  <dependencies> <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency> <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency> </dependencies>

  <build> <plugins> <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin> </plugins> </build>
</project>
```

SpringRestDemoApplication.java

- SpringRestDemoApplication.java ist die **Basisklasse** der Anwendung und enthält die main() Methode.
- @SpringBootApplication ist equivalent zu @Configuration, @ComponentScan @EnableAutoConfiguration

```
package de.hska.iwi.vslab;

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

@SpringBootApplication
public class SpringRestDemoApplication {

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

SpringRestDemoController.java

- Die `@RestController` Annotation weist darauf hin, dass `SpringRestDemoController` REST Endpunkte hat.

```
package de.hska.iwi.vslab;

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

@RestController
public class SpringRestDemoController {
    @RequestMapping(value = "/hello")
    public String helloWorld() {
        return "Hello World!";
    }
}
```

User Repository mit Spring JPA Modul

```
package de.hska.iwi.vslab.usrv;

@javax.persistence.Entity
public class User {
    @javax.persistence.Id
    @javax.persistence.GeneratedValue
    @javax.persistence.Column(name="USER_ID")
    private Long id;

    @javax.persistence.Column(name="USER_NAME")
    private String name;

    @javax.persistence.Column(name="USER_PWD")
    private String passwd;

    // getter und setter nicht aufgeführt
}
```

```
package de.hska.iwi.vslab.usrv;
public interface UserRepo extends
    org.springframework.data.repository.CrudRepository<User, Long> { }
```

Maven POM mit JPA Modul und Datenbank

```
<project ...>
...
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
...
</project>
```

User Service Controller

```
package de.hska.iwi.vslab.usrv;  
...  
@RestController  
public class UserController {  
    @Autowired  
    private UserRepo repo;  
  
    @RequestMapping(value = "/users", method = RequestMethod.GET)  
    public ResponseEntity<Iterable<User>> getUsers() {  
        Iterable<User> allPolls = repo.findAll();  
        return new ResponseEntity<>(allPolls, HttpStatus.OK);  
    }  
    @RequestMapping(value = "/users", method = RequestMethod.POST)  
    public ResponseEntity<?> addUser(@RequestBody User user) {  
        user = repo.save(user);  
        return new ResponseEntity<>(null, HttpStatus.CREATED);  
    }  
    @RequestMapping(value = "/users/{userId}", method = RequestMethod.GET)  
    public ResponseEntity<User> getUser(@PathVariable Long userId) {  
        User user = repo.findOne(userId);  
        return new ResponseEntity<>(user, HttpStatus.OK);  
    }  
}
```

REST Service Clients mit Spring RestTemplate

- Spring Unterstützung zur Entwicklung von REST Clients basiert auf dem `org.springframework.web.client.RestTemplate`
- Das **RestTemplate** kümmert sich um die Anbindung eines REST Service und führt automatisch die Konvertierung der Daten in HTTP Requests und Responses durch.
- RestTemplate ist Teil von `spring-web.jar`. Um einen Standalone Client zu entwickeln, muss die `spring-web Dependency` zur Maven POM hinzugefügt werden.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
```

RestTemplate API

- Die wichtigsten Template-Methoden entsprechen den HTTP Methoden.
- Das Template wickelt die HTTP Verbindungen ab; Anwendungen stellen URLs mit Template-Variablen bereit und verarbeiten das Ergebnis.

```
public interface RestOperations {  
    ...  
    public <T> T getForObject(String url, Class<T> responseType, Object... urlVariables)  
        throws RestClientException {}  
    public <T> T getForObject(URI url, Class<T> responseType) throws RestClientException  
    ...  
    public URI postForLocation(String url, Object request, Object... urlVariables)  
        throws RestClientException  
    public <T> T postForObject(String url, Object request, Class<T> responseType,  
        Object... uriVariables) throws RestClientException  
    ...  
    public void delete(String url, Object... urlVariables) throws RestClientException  
    ...  
    public void put(String url, Object request, Object... urlVariables)  
        throws RestClientException  
    ...  
}
```

RestTemplate Beispiel

```
package de.hska.iwi.vslab.usrv;

import java.net.URI;
import org.springframework.web.client.RestTemplate;

public class UserClient {
    private static final String USER_URI = „http://localhost:8080/users”;
    // Thread save
    private static RestTemplate restTemplate = new RestTemplate();

    public static void main(String[] args) {
        URI uri = restTemplate.postForLocation(USER_URI, new User(1L, "Alice", „pwd“));
        User user = restTemplate.getForObject(USER_URI + "/{id}", User.class, 1L);
        System.out.println(user.getName() + " has the URI " + uri);
    }
}
```

Spring REST: Literatur und Web Ressourcen

Literatur

[Tilkov 2015] Stefan Tilkov, Martin Eigenbrodt, Silvia Schreier et al. "REST und HTTP",
3. Auflage, dpunkt, 2015

[Varanasi 2015] Balaji Varanasi, Sudha Belida, "Spring REST", Apress, 2015

Aus dem Web (abgerufen 11/2019)

[Spring 2019a] Spring Team, „Getting Started: Building a RESTful Web Service“
<https://spring.io/guides/gs/rest-service/>

[Spring 2019b] Spring Team, „Getting Started: Consuming a RESTful Web Service“
<https://spring.io/guides/gs/consuming-rest/>

[Spring 2019c] Spring Team, „Building REST services with Spring“,
<http://spring.io/guides/tutorials/bookmarks/>

[Spring 2019d] Spring Team, „Spring Reference Documentation“,
<https://spring.io/docs/reference>