

Softwareentwicklung (SW)

APIs, Web Services, RESTFul APIs

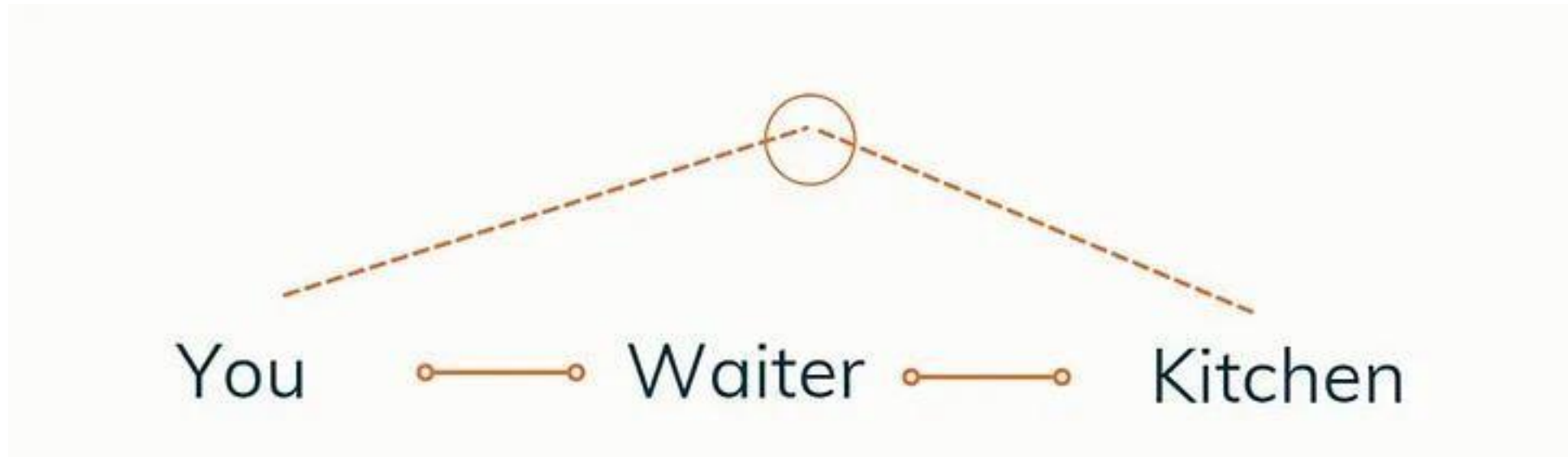
Prof. Dr. Alixandre Santana
alixandre.santana@oth-regensburg.de

Wintersemester
2024/2025

- die Konzepte von API, Webservices und RESTFul APIs zu verstehen
- die Merkmale einer typischen RESTFul-API zu identifizieren
- eine CRUD-RESTFul-API zu implementieren

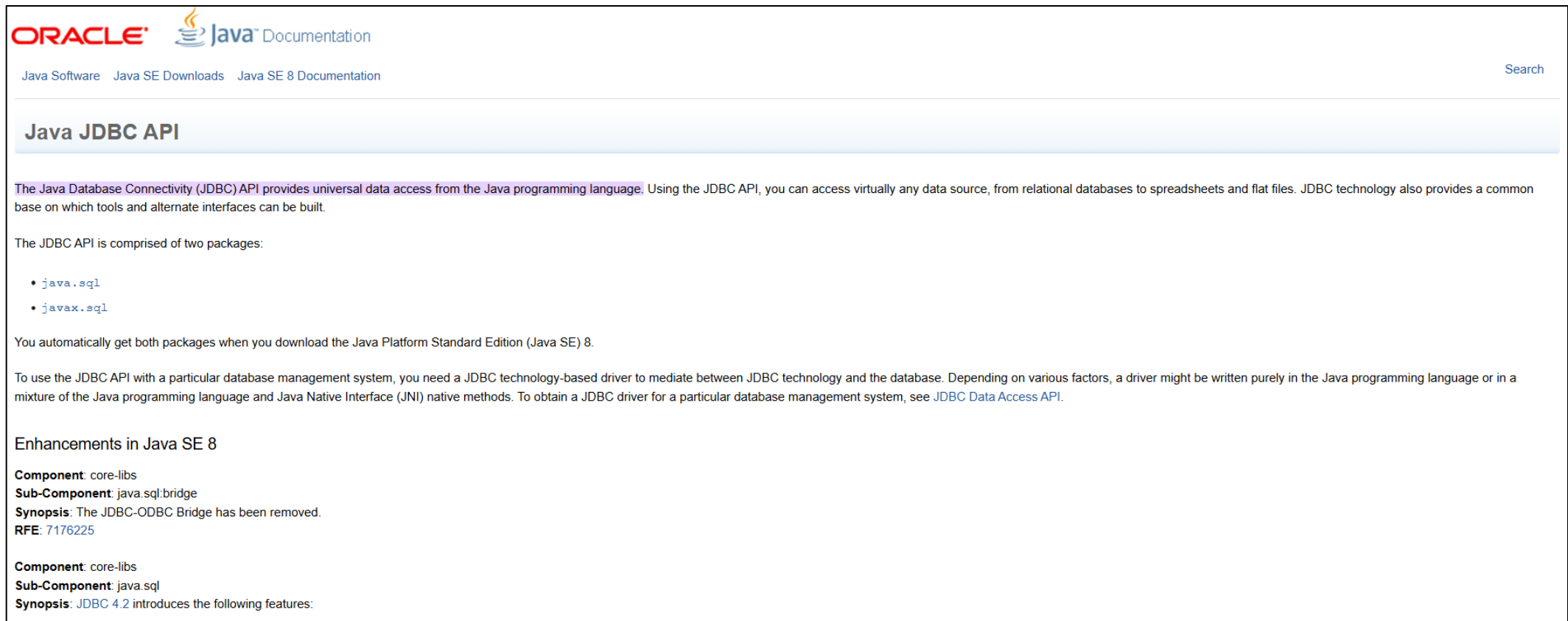
1. APIs und Microservices
2. Web-Services
3. RESTFul APIs
4. Beispiel

Application Programming Interface (API)



<https://medium.com/analytics-vidhya/what-is-an-api-api-for-beginners-4854f34153e>

„APIs sind Mechanismen, die es **zwei Software-Komponenten** ermöglichen, über eine Reihe von Definitionen und Protokollen miteinander **zu kommunizieren**.“



The screenshot shows the Oracle Java Documentation page for the Java JDBC API. The page header includes the Oracle logo and the text "Java SE 8 Documentation". Below the header, there is a search bar and a navigation menu with links to "Java Software", "Java SE Downloads", and "Java SE 8 Documentation". The main heading is "Java JDBC API". The text describes the JDBC API as providing universal data access from the Java programming language. It lists two packages: `java.sql` and `javax.sql`. It also mentions that the JDBC API is comprised of two packages and that you automatically get both packages when you download the Java Platform Standard Edition (Java SE) 8. The page further explains that to use the JDBC API with a particular database management system, you need a JDBC technology-based driver to mediate between JDBC technology and the database. It also mentions that the JDBC-ODBC Bridge has been removed and provides a link to the JDBC Data Access API. The page is divided into sections for "Enhancements in Java SE 8" and "Component: core-libs".

ORACLE Java[™] Documentation

Java Software Java SE Downloads Java SE 8 Documentation Search

Java JDBC API

The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which tools and alternate interfaces can be built.

The JDBC API is comprised of two packages:

- `java.sql`
- `javax.sql`

You automatically get both packages when you download the Java Platform Standard Edition (Java SE) 8.

To use the JDBC API with a particular database management system, you need a JDBC technology-based driver to mediate between JDBC technology and the database. Depending on various factors, a driver might be written purely in the Java programming language or in a mixture of the Java programming language and Java Native Interface (JNI) native methods. To obtain a JDBC driver for a particular database management system, see [JDBC Data Access API](#).

Enhancements in Java SE 8

Component: core-libs
Sub-Component: java.sql:bridge
Synopsis: The JDBC-ODBC Bridge has been removed.
RFE: 7176225

Component: core-libs
Sub-Component: java.sql
Synopsis: JDBC 4.2 introduces the following features:

<https://aws.amazon.com/what-is/api>

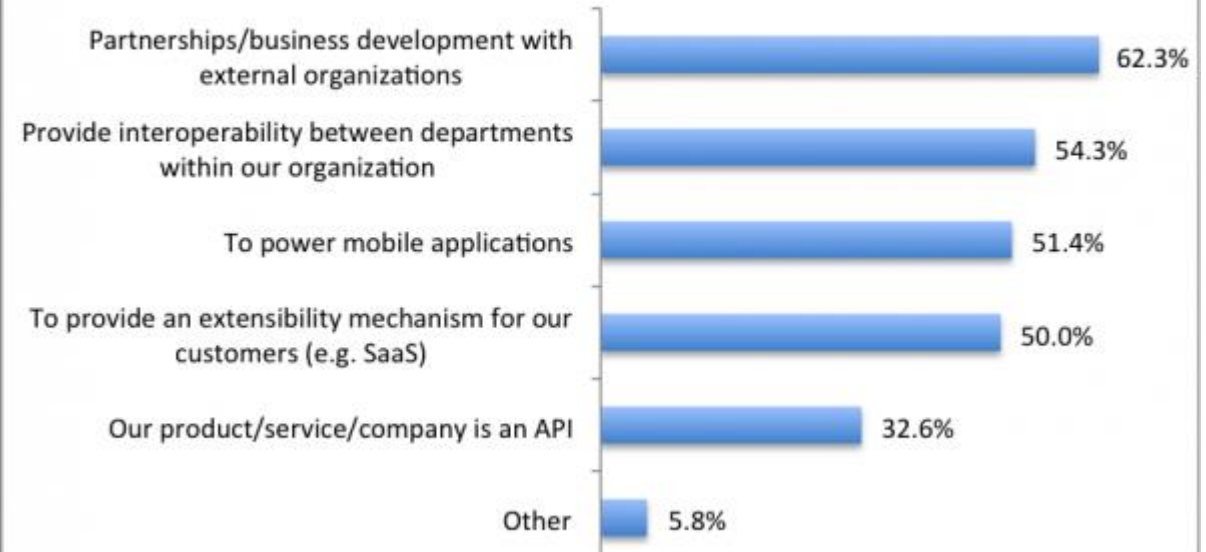
[https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/#:~:text=The%20Java%20Database%20Connectivity%20\(JDBC,from%20the%20Java%20programming%20language.](https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/#:~:text=The%20Java%20Database%20Connectivity%20(JDBC,from%20the%20Java%20programming%20language.)

APIs - Wofür?

Why do you consume APIs?

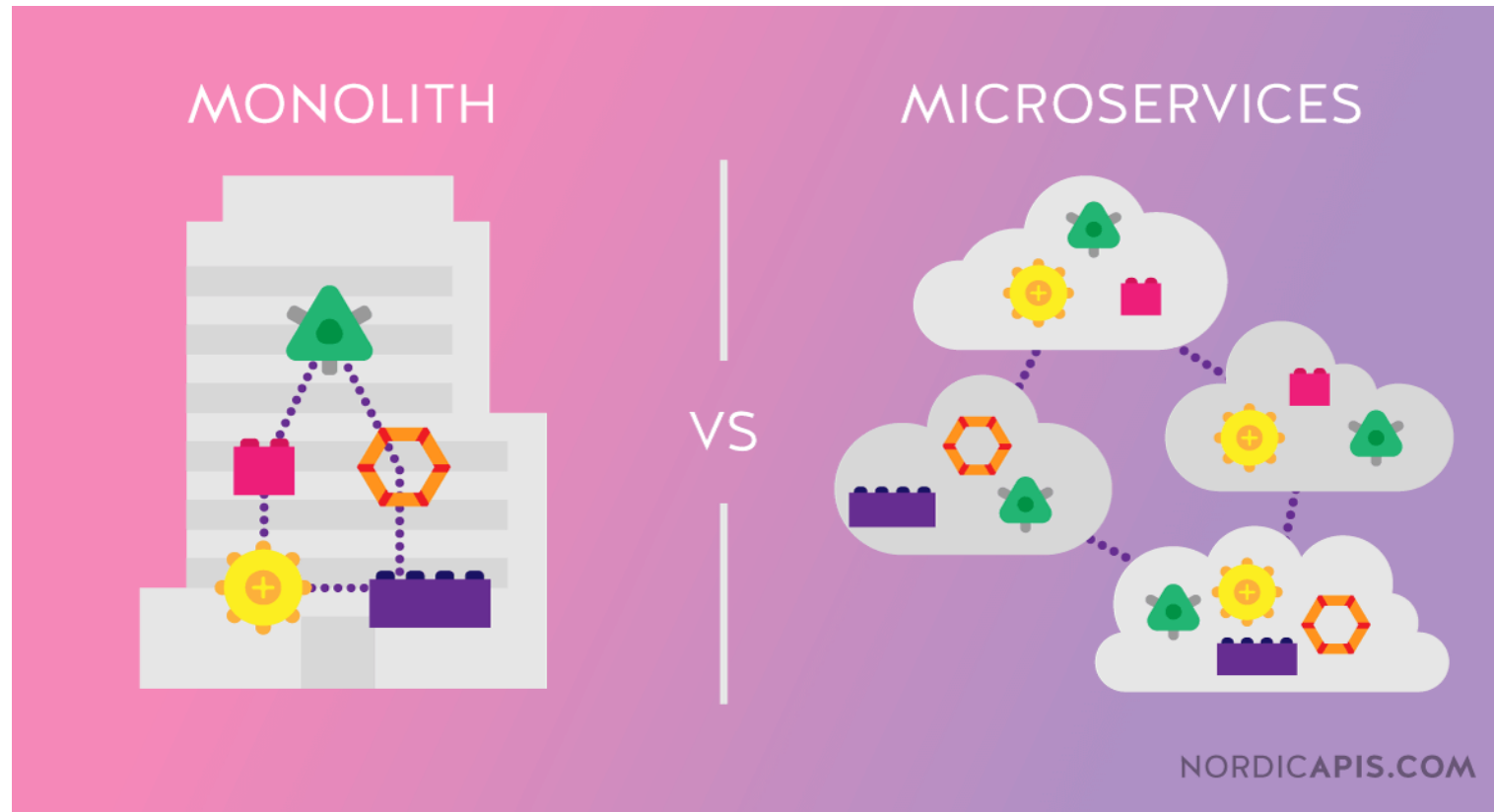


Why produce APIs?



<https://www.programmableweb.com/news/api-consumers-want-reliability-documentation-and-community/2013/01/07>

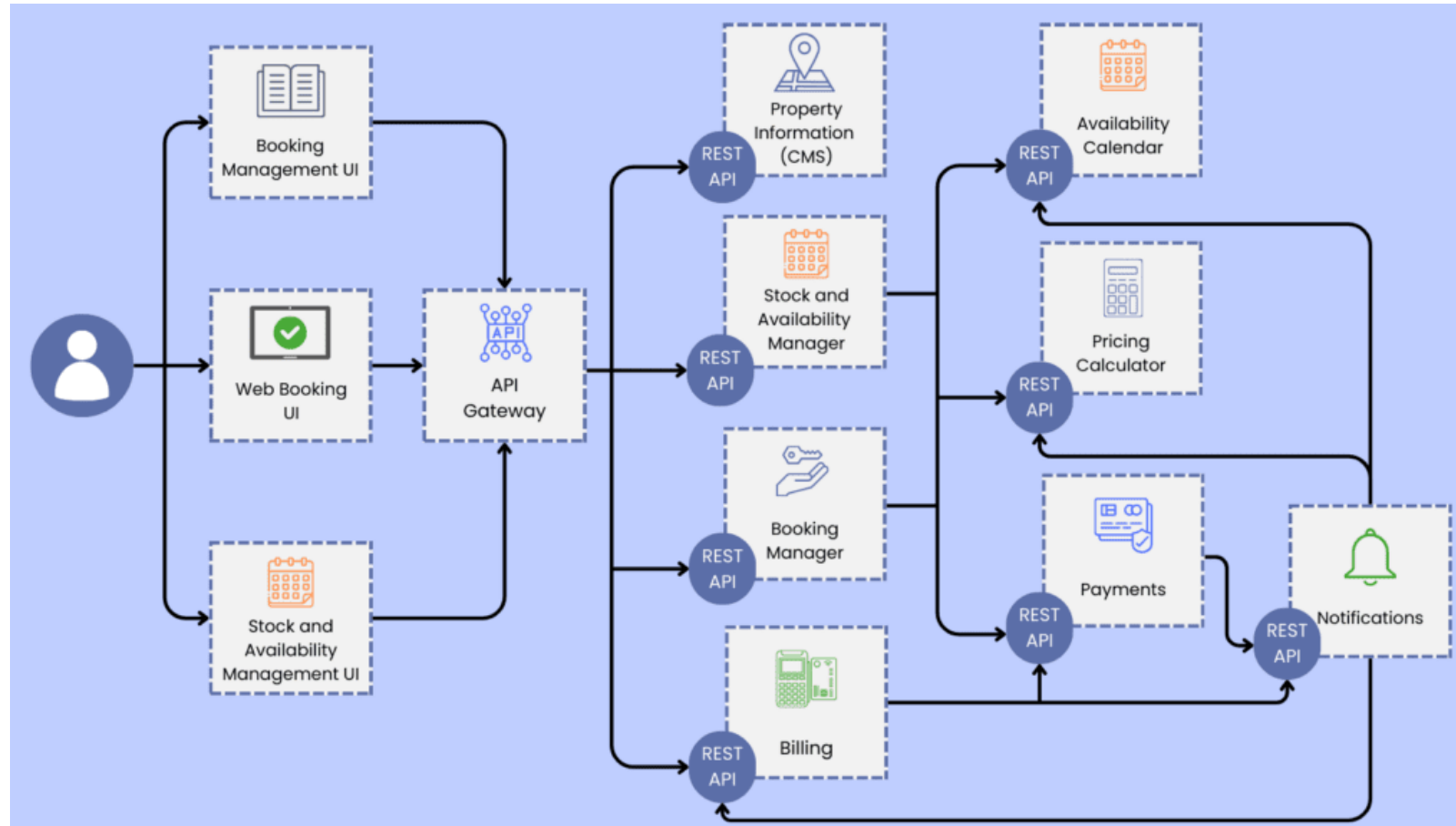
APIs und Microservices



Ein
Architekturstil,
bei dem eine
Anwendung in
kleine,
unabhängige
Dienste
unterteilt ist

<https://nordicapis.com/should-you-start-with-a-monolith-or-microservices/>,

APIs und Microservices



Ein Architekturstil, bei dem eine Anwendung in kleine, unabhängige Dienste unterteilt ist

1. APIs und Microservices
2. Web-Services
3. RESTFul APIs
4. Beispiel

Webservices

- Webservices sind **Softwarekomponenten**, die eine Maschine-zu-Maschine-Kommunikation technisch umsetzen
- Sie dienen zur Umsetzung einer serviceorientierten Architektur (SOA)
- Gibt es zwei „Klassen“ von Webservices:
 - **Service-orientierte Webservices:** Beliebige Definition von Service-Operationen XML- basierend auf WS-Standards wie WSDL u. SOAP (Simple Object Access Protocol)
 - **Resource-orientierte Webservice:** RESTful Webservices zur Manipulation von „Ressourcen“ nach dem CRUD-Paradigma

Was sind Webservices?¹

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

¹<http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>

Webservices

Web-Services sind unabhängig. Auf der Clientseite ist keine zusätzliche Software erforderlich. Eine Programmiersprache mit XML- und HTTP-Clientunterstützung ist für die ersten Schritte ausreichend. Auf der Serverseite sind ein Web-Server und eine Servlet-Steuerkomponente erforderlich. Client und Server können in verschiedenen Umgebungen implementiert werden. Es ist möglich, eine vorhandene Anwendung über einen Web-Service zu aktivieren, ohne eine einzige Zeile Code zu schreiben.

Web-Services sind selbst beschreibend. Client und Server müssen lediglich Format und Inhalt von Anforderungen und Antwortnachrichten erkennen. Die Definition des Nachrichtenformats wird mit der Nachricht transportiert, es sind keine externen Metadatenrepositories oder Codegenerierungstools erforderlich.

Web-Services sind modular. Einfache Web-Services können zu komplexeren Web-Services zusammengefasst (modular aufgebaut) werden; entweder durch die Verwendung von Workflow-Verfahren, oder durch das Aufrufen von Web-Services einer niedrigeren Schicht aus Web-Service-Implementierungen.

Web-Services sind plattformunabhängig. Web-Services basieren auf einer präzisen Gruppe offener, XML-basierter Standards, die der Förderung der Interoperabilität zwischen einem Web-Service und Clients über eine Vielzahl von IT-Plattformen und Programmiersprachen hinweg dienen.

Webservices

- Datenaustausch (Attribute) meist in XML oder **JSON**
- Jeder Service hat eine eindeutige **URI** (Uniform Resource Identifier)
- Maschinenlesbare Schnittstellenbeschreibung (z. B. **WS-WSDL**)
- Nutzung verbreiteter Internet-Protokolle (z. B. **HTTP**)

```
<definitions>

<types>
  data type definitions.....
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations.....
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Was sind die Ähnlichkeiten zwischen SOAP und REST?

Um Anwendungen zu erstellen, können Sie viele verschiedene Programmiersprachen, Architekturen und Plattformen verwenden. Es ist eine Herausforderung, Daten zwischen so unterschiedlichen Technologien auszutauschen, da sie unterschiedliche Datenformate haben. **Sowohl SOAP als auch REST wurden entwickelt**, um dieses Problem zu lösen.

Sie können SOAP und REST verwenden, um APIs oder Kommunikationspunkte zwischen verschiedenen Anwendungen zu erstellen.

APIs sind jedoch die umfassendere Kategorie. Webservices sind eine spezielle Art von API.

https://aws.amazon.com/de/compare/the-difference-between-soap-rest/?nc1=h_ls

Umsetzungen von Webservices

- SOAP/WSDL
- **RESTful Webservices**
- Remote Procedure Call (RPC)



1. APIs und Microservices
2. Web-Services
3. RESTFul APIs
4. Beispiel

- REST ist die Abkürzung für **RE**presentational **S**tate **T**ransfer
- Es handelt sich um einen **Architekturstil** für verteilte Hypermedia-Systeme
- Der REST-Architekturstil bestimmt, wie der Representational State Transfer (REST) durchgeführt werden soll, d. h. die Darstellung, die dem Wertesatz entspricht, der eine **bestimmte Entität** zu einem bestimmten Zeitpunkt darstellt.

- Die wichtigste Informationsabstraktion in REST ist eine **Ressource**.
- Jede benennbare Information kann eine Ressource sein: **ein Dokument oder Bild, ein Dienst, eine Collection anderer Ressourcen**.
- Der Zustand der Ressource zu einem bestimmten Zeitstempel wird als bezeichnet die „**Resource Representation**“.

- Einsatz von **URIs (Uniform Resource Identifiers or endpoints)**, in der Regel Webadressen, die sowohl den **Server** identifizieren, auf dem die Anwendung gehostet wird, als auch die **Anwendung** selbst und welche angebotenen Features angeboten werden.
- Diese Informationen oder Darstellungen werden in einem von mehreren Formaten über HTTP bereitgestellt: **JSON (Javascript Object Notation)**, HTML, XLT, Python, PHP oder einfacher Text.

Was macht eine API zu einer RESTFul-API?

- Schöne URLs wie „/employees/3“ sind kein REST.
- Die bloße Verwendung von GET, POST usw. ist kein REST.
- Die Anordnung aller CRUD-Operationen ist kein REST.

Was macht eine API zu einer RESTFul-API?

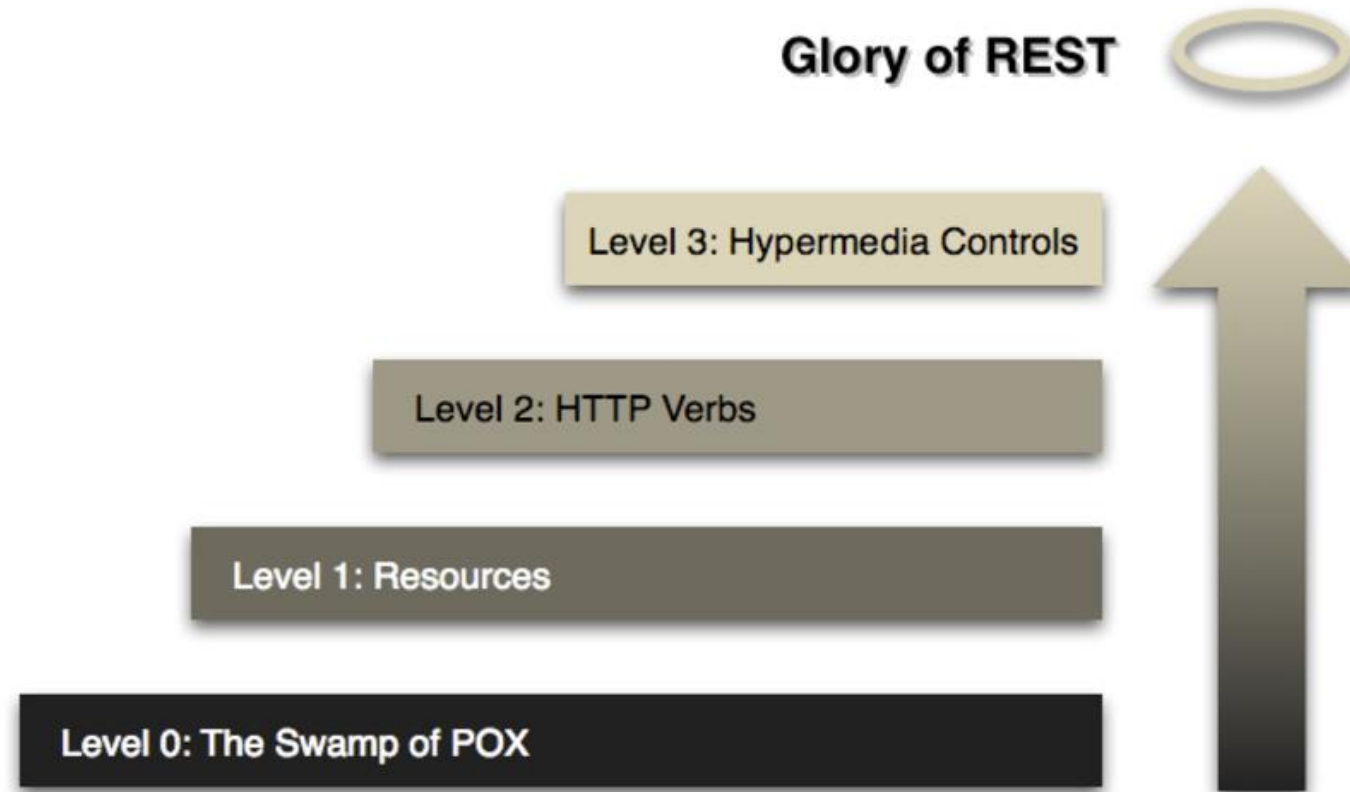


Figure 1: Steps toward REST

- Verwenden Sie Substantive im Plural: Die Pluraloption erleichtert die Identifizierung von Ressourcen in den URIs, vermeidet Fehler und vereinfacht das Debuggen der Kommunikation zwischen Client und Server.
- Vermeiden Sie die Verwendung von Leerzeichen: Verwenden Sie den Unterstrich (_) oder den Bindestrich (-), wenn Sie lange Namen verwenden, die aus mehr als einem Wort bestehen.
- Verwenden Sie Kleinbuchstaben: Obwohl bei URIs zwischen Groß- und Kleinschreibung unterschieden wird, ist es eine gute Praxis, alle Ihre Buchstaben klein zu schreiben, um Fehler bei der Bildung von URIs zu vermeiden, wie im Fall der Verwendung von Leerzeichen;
- Bewahren Sie die „Backwards Compatibility“: Da Webdienste öffentliche Dienste sind, sollte eine URI, sobald sie offengelegt wurde, immer verfügbar sein. Wenn eine URI aktualisiert werden muss, sollte die nicht verwendete URI mit dem Rückgabecode 300 des HTTP-Protokolls an die neue Adresse umgeleitet werden;
- Verwenden von HTTP-Befehlen: Wenn Sie Vorgänge an Ressourcen ausführen, verwenden Sie die GET-, PUT-, POST- und DELETE-Nachrichten des HTTP-Protokolls. Es empfiehlt sich nicht, Vorgangsnamen in URIs zu verwenden.

Konventionen für RESTful-API

- Je (Root) Entity/Aggregate ein Service mit CRUD-Operationen
- Keine individuellen Service-Operations-Namen nach Außen
 - POST, PUT, GET, DELETE, OPTIONS, TRACE, HEAD, CONNECT, PATCH
 - POST Neuanlage (CREATE)
 - PUT Änderung (UPDATE, komplettes Objekt überschreiben)
 - PATCH Änderung (UPDATE, einzelne, übergebene Attribute)
 - DELETE Löschen einer (Root-)Entity
 - GET Abfragen der Entity-Daten
- Ressource-Pfad enthält (Root-)Entity-Typ (Englisch und im Plural) und Primärschlüsselwert
- Angemessene Verwendung von **HTTP-Headern und Statuscode** für die Antwort auf Anfragen.

Resource (HTTP) methods: POST

Das POST-Verb wird am häufigsten zum ****Erstellen**** neuer Ressourcen verwendet.

Bei erfolgreicher Erstellung wird der **HTTP-Status 201** zurückgegeben und ein Location-Header mit einem Link zur neu erstellten Ressource mit dem HTTP-Status 201 zurückgegeben.

<http://200.212.34.56/clients/345>

Resource (HTTP) methods: GET

Die HTTP-GET-Methode wird verwendet, um eine Darstellung einer Ressource zu ****lesen**** (oder abzurufen).

Im „glücklichen“ (oder fehlerfreien) Pfad gibt GET eine Darstellung in XML oder JSON und einen HTTP-Antwortcode **von 200 (OK)** zurück. Im Fehlerfall wird meistens 404 (NOT FOUND) oder 400 (BAD REQUEST) zurückgegeben.

Gemäß dem Design der HTTP-Spezifikation werden GET-Anfragen (zusammen mit HEAD-Anfragen) nur zum Lesen von Daten und nicht zum Ändern dieser Daten verwendet. Daher gelten sie bei dieser Verwendung als sicher.

Resource (HTTP) methods: PUT

PUT wird am häufigsten für ****Aktualisierungsfunktionen**** verwendet, wobei PUT an einen bekannten Ressourcen-URI gesendet wird, wobei der Anforderungstext die neu aktualisierte Darstellung der ursprünglichen Ressource enthält.

Bei erfolgreicher Aktualisierung wird **200** (oder **204**, wenn kein Inhalt im Hauptteil zurückgegeben wird) von einem PUT zurückgegeben. Wenn Sie PUT zum Erstellen verwenden, geben Sie bei erfolgreicher Erstellung den HTTP-Status **201** zurück. Ein Text in der Antwort ist optional – vorausgesetzt, er verbraucht mehr Bandbreite.

Resource (HTTP) methods: DELETE

DELETE ist ziemlich einfach zu verstehen. Es wird verwendet, um eine durch einen URI identifizierte Ressource zu ****löschen****.

Bei erfolgreichem Löschvorgang wird der HTTP-Status **200 (OK)** zusammen mit **einem Antworttext zurückgegeben**, möglicherweise der Darstellung des gelöschten Elements (erfordert häufig zu viel Bandbreite) oder einer verpackten Antwort (siehe Rückgabewerte unten). Entweder das oder es wird der HTTP-Status **204 (NO CONTENT)** ohne Antworttext zurückgegeben.

Kurze Wiederholung

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.


Agenda

1. APIs und Microservices
2. Web-Services
3. RESTFul APIs
4. Beispiel

Resource Representation with JSON

JSON FORMATTER & VALIDATOR

AboutLearnBookmarkletFAQChangelogContact



Paste in JSON or a URL, drop a file, browse, or load an example to begin.

JSON Data/URL

```
{  
  "nome": "Marcelino Avelino",  
  "email": "marcelino.avelino@gmail.com",  
  "endereco": "Rua M",  
  "cpf" : "12345678910"  
}
```

Process

JSON Template

3 Space Tab

JSON Specification

RFC 8259

Fix JSON ☒ ?

<https://jsonformatter.curiousconcept.com/>

RESTful-Webservices- Implementierung mit Spring MVC



- RESTful-Webservices werden über den Web-Container und den zugehörigen Webserver zur Verfügung gestellt
- Sie werden über eine **Controller-Klasse** und zugehörigen Request-Mappings zur Verfügung gestellt (obwohl sie „Service-Schnittstellen“ sind)
- Die Controller-Implementierungen können entsprechend die empfangenen Daten validieren
- Die Controller-Klasse wird mit der Annotation **@RestController** annotiert

@RestController

- @RestController ist eine spezielle Version des Controllers.
- Es enthält die Annotationen @Controller und @ResponseBody und vereinfacht dadurch die Controller-Implementierung.

Implementierung der Service-Operationen

```
@RestController
public class KundenServiceRestController {

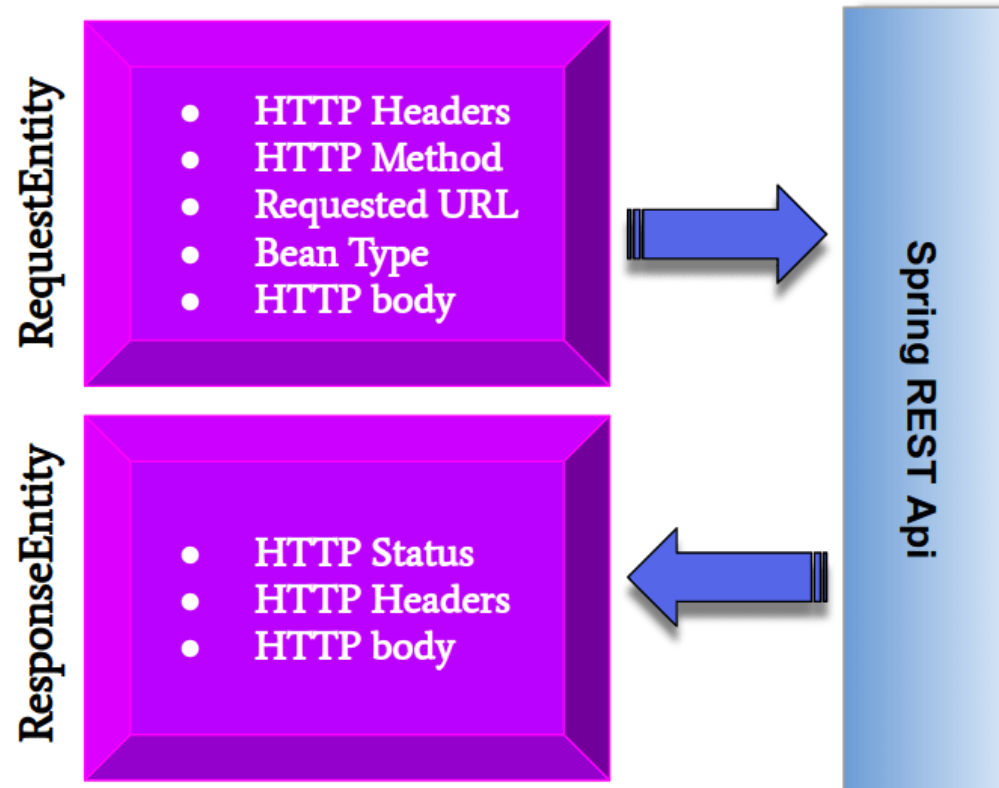
    @Autowired
    private KundenService kundenService;

    @RequestMapping(value="/api/customers", method = RequestMethod.POST)
    public Kunde kundeErzeugen(@RequestBody Kunde kunde) {
        return kundenService.erzeugeNeuenKunden(kunde);
    }

    @RequestMapping(value="/api/customers/{knr}", method = RequestMethod.GET)
    public Kunde kundeLesen(@PathVariable("knr") String kundenId) {
        return kundenService.sucheKunde(kundenId);
    }

    @RequestMapping(value="/api/customers/", method = RequestMethod.GET)
    public List<Kunde> kundenSuchen(@PathParam("category") String kategorie,
                                   @PathParam("size") int seitenLänge,
                                   @PathParam("offset") int nrStartSeite) {
        /* kategorie, seitenLänge und/oder nrStartSeite können auch null bzw. 0
           sein (nicht vorhanden)
           wie auch immer die Suche im Service dargestellt ist :-)
        */
    }
}
```


ResponseEntity & RequestEntity parts



@ResponseBody

Es stellt eine HTTP-Antwort dar mit **status, headers and body**.

Mit ResponseEntity können Sie die Antwort mit optionalen Header und Statuscode ändern.

```
@GetMapping(value = "/customers/{id}")
public ResponseEntity<Customer> read(@PathVariable(name = "id") int id) {
    final Customer customer = customerService.findById(id);

    return customer != null
        ? new ResponseEntity<>(customer, HttpStatus.OK)
        : new ResponseEntity<>(HttpStatus.NOT_FOUND);
}
```

HATEOAS - Library

- Spring HATEOAS (**H**yper**m**edia as the **E**ngine of **A**pplication **S**tate) bietet einige APIs, um die Erstellung von REST-Darstellungen zu erleichtern, die dem HATEOAS-Prinzip folgen, wenn mit Spring und insbesondere Spring MVC gearbeitet wird.

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-hateoas</artifactId>  
  <version>2.6.4</version>  
</dependency>
```

Es bietet:

- **Modell-klassen** für Link- und Ressourcendarstellungsmodelle
- **Link Builder-API** zum Erstellen von Links, die auf Spring MVC-Controller-Methoden verweisen

HATEOAS - Library

- Spring HATEOAS (**H**yper**m**edia as the **E**ngine of **A**pplication **S**tate) bietet einige APIs, um die Erstellung von REST-Darstellungen zu erleichtern, die dem HATEOAS-Prinzip folgen, wenn mit Spring und insbesondere Spring MVC gearbeitet wird.

```
@PostMapping("/api/workshops/")
public ResponseEntity<EntityModel<Workshop>> post(@RequestBody Workshop
workshopFromRequest) {
    Workshop workshop = workshopService.saveWorkshop(workshopFromRequest);
    EntityModel<Workshop> entityModel = EntityModel.of(workshop);
    Link workshopLink =
    WebMvcLinkBuilder.LinkTo(WebMvcLinkBuilder.methodOn(WorkshopRestController.class)
    .getWorkshopById(workshop.getId())).withSelfRel();
    entityModel.add(workshopLink);
    return new ResponseEntity<>(entityModel, HttpStatus.CREATED);
}
```

HATEOAS - Library

- Spring HATEOAS (**H**yper**m**edia as the **E**ngine of **A**pplication **S**tate) bietet einige APIs, um die Erstellung von REST-Darstellungen zu erleichtern, die dem HATEOAS-Prinzip folgen, wenn mit Spring und insbesondere Spring MVC gearbeitet wird.

GET <http://localhost:8080/api/workshops/1>

```
{
  "id": 1,
  "description": "IoT",
  "date": null,
  "_links": {
    "self": {
      "href": "http://localhost:8080/api/workshop/1"
    }
  }
}
```

HATEOAS - Library

- Spring HATEOAS (**H**yper**m**edia as the **E**ngine of **A**pplication **S**tate) bietet einige APIs, um die Erstellung von REST-Darstellungen zu erleichtern, die dem HATEOAS-Prinzip folgen, wenn mit Spring und insbesondere Spring MVC gearbeitet wird.

[GET http://localhost:8080/api/workshops/](http://localhost:8080/api/workshops/)

```
{
  "_embedded": {
    "workshopList": [
      {
        "id": 1,
        "description": "IoT",
        "date": null,
        "_links": {
          "self": {
            "href": "http://localhost:8080/api/workshop/1"
          }
        }
      },
      {
        "id": 2,
        "description": "ChatGPT",
        "date": null,
        "_links": {
          "self": {
            "href": "http://localhost:8080/api/workshop/2"
          }
        }
      },
      ...
    ]
  }
}
```

Wir benötigen die folgenden Komponenten, um eine externe API zu nutzen:

- WebClient (RestTemplate –VOR Spring 5.0)

WebClient ist eine Schnittstelle, die den Haupteinstiegspunkt für die Ausführung von Web Requests darstellt.

WebClient unterstützt sowohl synchrone als auch nicht-synchrone Operationen.

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

Aufrufen einer Drittanbieter-API

Das Senden einer HTTP-POST-Anfrage umfasst beispielsweise die folgenden Schritte:

- Create WebClient.UriSpec reference using prebuilt methods such as get(), put(), post() or delete().
- Set the request URI if not set already.
- Set the request headers and authentication details, if any.
- Set the request body, if any.
- Call the retrieve() or exchange() method.
 - The retrieve() method directly performs the HTTP request and retrieves the response body.
 - The exchange() method returns ClientResponse having the response status and headers. We can get the response body from ClientResponse instance.
- Handle the response returned from the server.

Aufrufen einer Drittanbieter-API

- WebClient

Flux<T> is useful when we need to handle zero to many or potentially infinite results. We can think of a Twitter feed as an example.

When we know that the results are returned all at once - as in our use case - we can use *Mono<T>*.

```
@ResponseBody
@GetMapping("/api/posts")
public List<PostJSON> demoPostRESTAPI() {
    Mono<PostJSON[]> response = webClient.get()
        .uri(GET_POSTS_ENDPOINT_URL)
        .accept(MediaType.APPLICATION_JSON)
        .retrieve()
        .bodyToMono(PostJSON[].class).log();
    PostJSON[] posts = response.block();
    return Arrays.asList(posts);
}
```

Aufrufen einer Drittanbieter-API

```
@ResponseBody
@GetMapping("/api/posts/{id}")
public ResponseEntity <EntityModel<PostJSON>> getPostByIDRESTAPI(@PathVariable String id) {

    EntityModel<PostJSON> entityModel = null;
    try {
        Mono<PostJSON> postMono= webClient.get()
            .uri(GET_POSTS_ENDPOINT_URL+"/{id}", id)
            .retrieve()
            .onStatus(httpStatus -> !httpStatus.is2xxSuccessful(), error -> Mono.error(new
                WebClientException("Post not found")))
            .bodyToMono(PostJSON.class);
        entityModel = EntityModel.of(postMono.block());
    }
    catch(WebClientException webClientException) {
        System.out.println(webClientException.getMessage());
        return ResponseEntity.notFound().build();
    }
    return new ResponseEntity<>(entityModel, HttpStatus.OK);
}
```

Hands On!

CRUD Beispiel mit Workshop

Methods	Urls	Actions
POST	/api/workshops	create new Workshop
GET	/api/workshops	retrieve all Workshop s
GET	/api/workshops/:id	retrieve a Workshop by :id
PUT	/api/workshops/:id	update a Workshop by :id
DELETE	/api/workshops/:id	delete a Workshop by :id
DELETE	/api/workshops	delete all Workshop s
GET	/api/workshops?description=[key word]	find all Workshops which description contains keyword

Empfehlungen für REST-Schnittstellen

Die Request-Pfade und die HTTP-Methoden einer REST-Schnittstelle müssen systematisch aufgebaut sein.

Inspirationen und Beispiele u. a. zu finden hier:

- <https://jsonapi.org/recommendations/>
(Insbesondere Abschnitte „Naming“, „URL Design“ und „related resource URL“)
- <https://www.martinfowler.com/articles/richardsonMaturityModel.html>
(Inklusive Level 2)
- <http://restcookbook.com>

Referenzen

- Classroom notes of Prof. Daniel Jobst
- https://aws.amazon.com/what-is/restful-api/?nc1=h_ls
- <https://martinfowler.com/articles/richardsonMaturityModel.html>
- <https://www.youtube.com/watch?v=9GWK9A79tEc>
- <https://www.baeldung.com/rest-with-spring-series>
- <https://restfulapi.net/rest-architectural-constraints/>
- <https://www.restapitutorial.com/lessons/httpmethods.html>
- <https://www.baeldung.com/java-dto-pattern#:~:text=2.,Fowler%20in%20his%20book%20EAA>
- <https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/>
- <https://www.javainuse.com/webseries/spring-security-jwt/chap6>
- <https://jsonplaceholder.typicode.com/>