Java EE

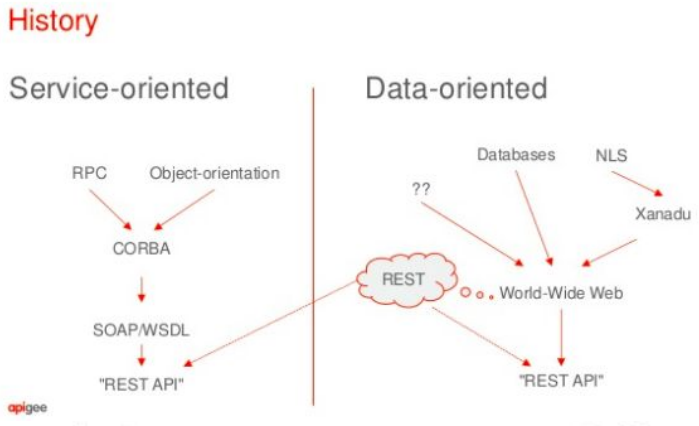
REST, SOAP, Beispiele & Architekturen

API

Grundlagen

API - Definition

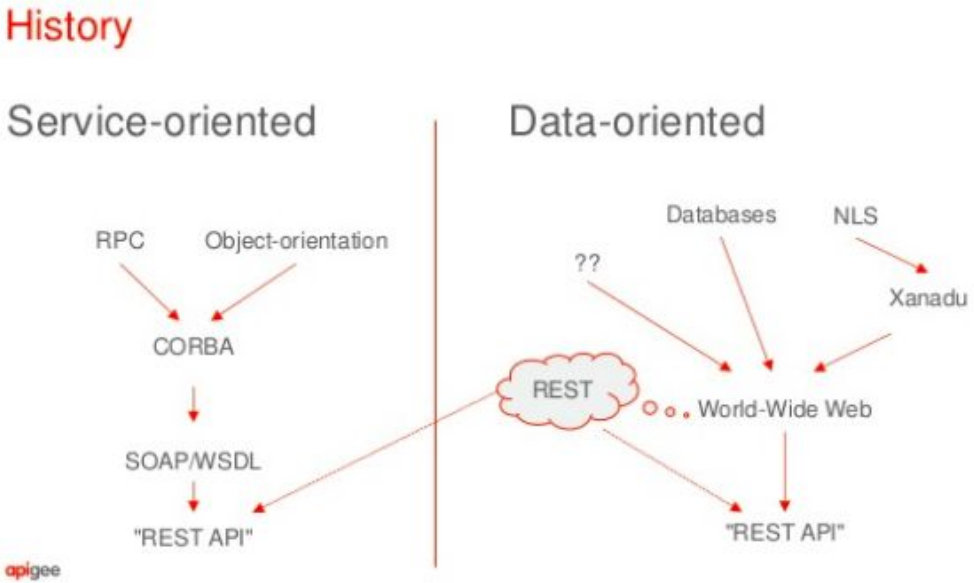
“Die API ist eine Schnittstelle, die ein Softwaresystem bereitstellt, um dieses in andere Programme einzubinden.” aus Gründerszene



aus Computersciencewiki

APIs - Historie

von Plattformabhängigen zu Plattformneutralen APIs

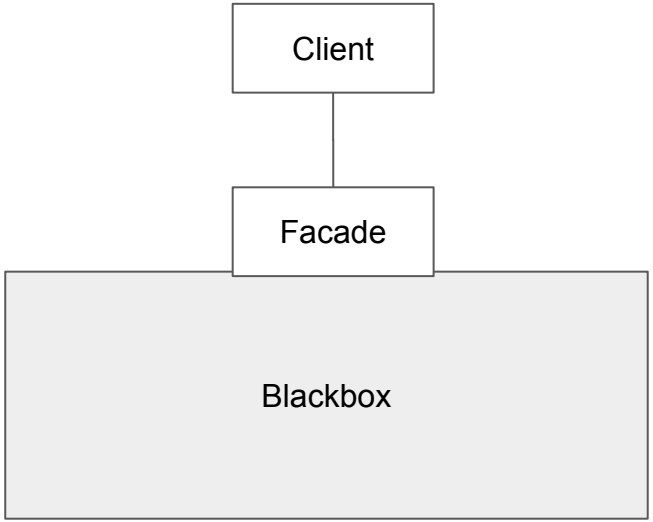


aus Slideshare

Service-Architekturen

Grundlagen

Prinzip - Information Hiding & Domäne

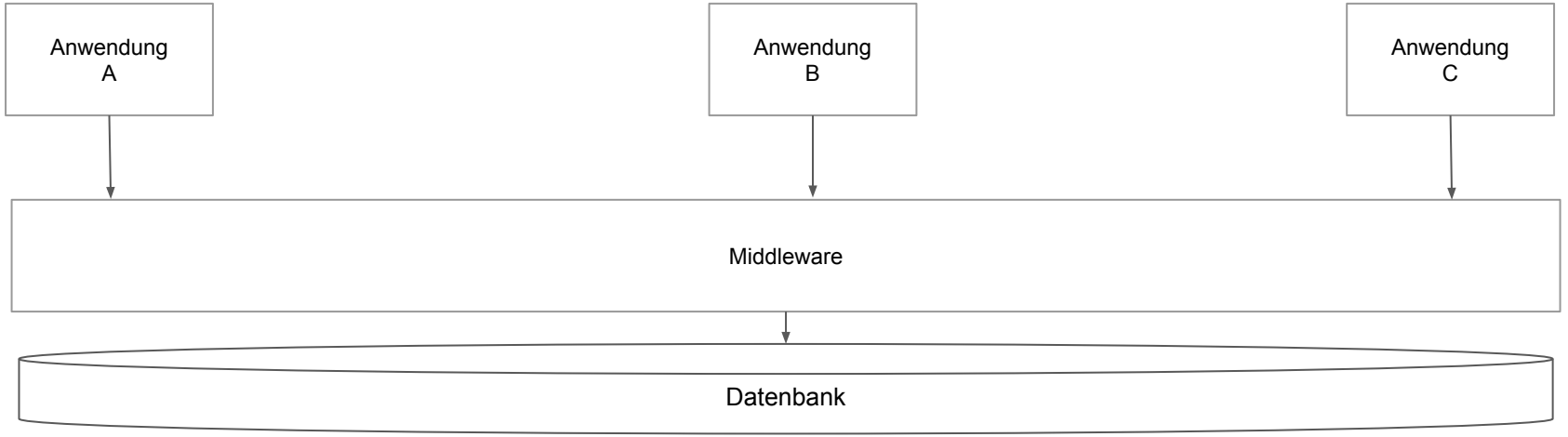


Vorteile:

- Verbergung von Komplexität
- Bedingung für Modularisierung

Architekturkonzepte - Middlewarearchitektur

Vermittlung zwischen Anwendungen
Ziel: Komplexität soll verborgen werden

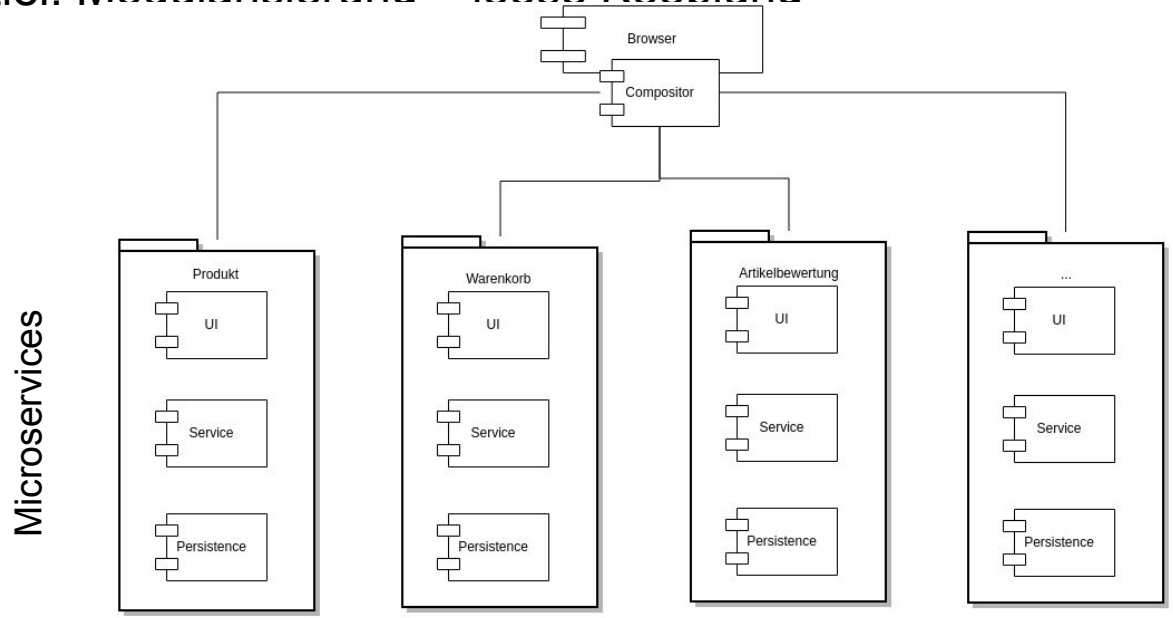


Anwendungsorientiert Middlewares: JEE o. .NET
Kommunikationsorientierte Middlewares: RMI o. SOAP
Nachrichtenorientierte Middlewares: JMS o.

Architekturkonzepte - Microservice-Architekturen

Vertikale Decomposition

Ziel: Modularisierung + loose Kopplung



Ziel: Kleinsten Nenner z.B. durch Ressourcenorientierung
Existenzebenen: Organisation, Entwicklungsebene, Boot-Ebene und Laufzeitebene
Horizontale Skalierung oft zur Laufzeit jeder Komponente möglich

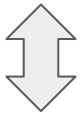
REST Architektur

Grundlagen

Client-Server-Modell

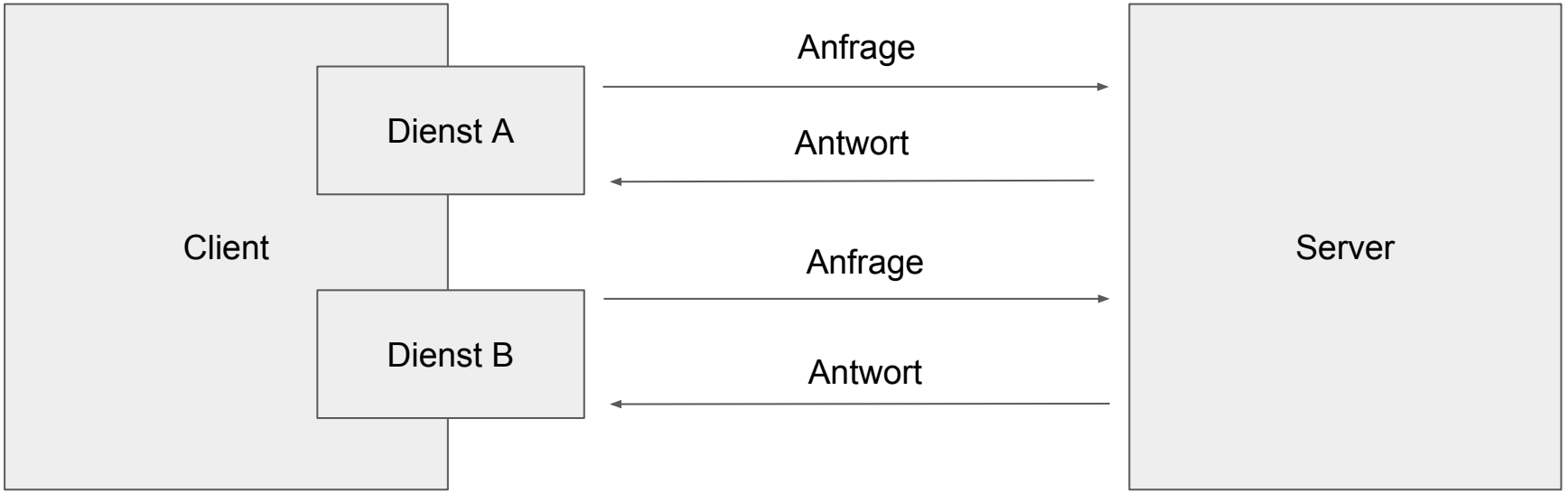
“Das Client-Server-Modell beschreibt eine Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. “
aus Wikipedia

Der **Client fordert Betriebsmittel** beim Server ein



Der **Server stellt** diese **Betriebsmittel** dem **Client zur Verfügung**

Client-Server-Modell // Anfragen, Antwort, Dienst

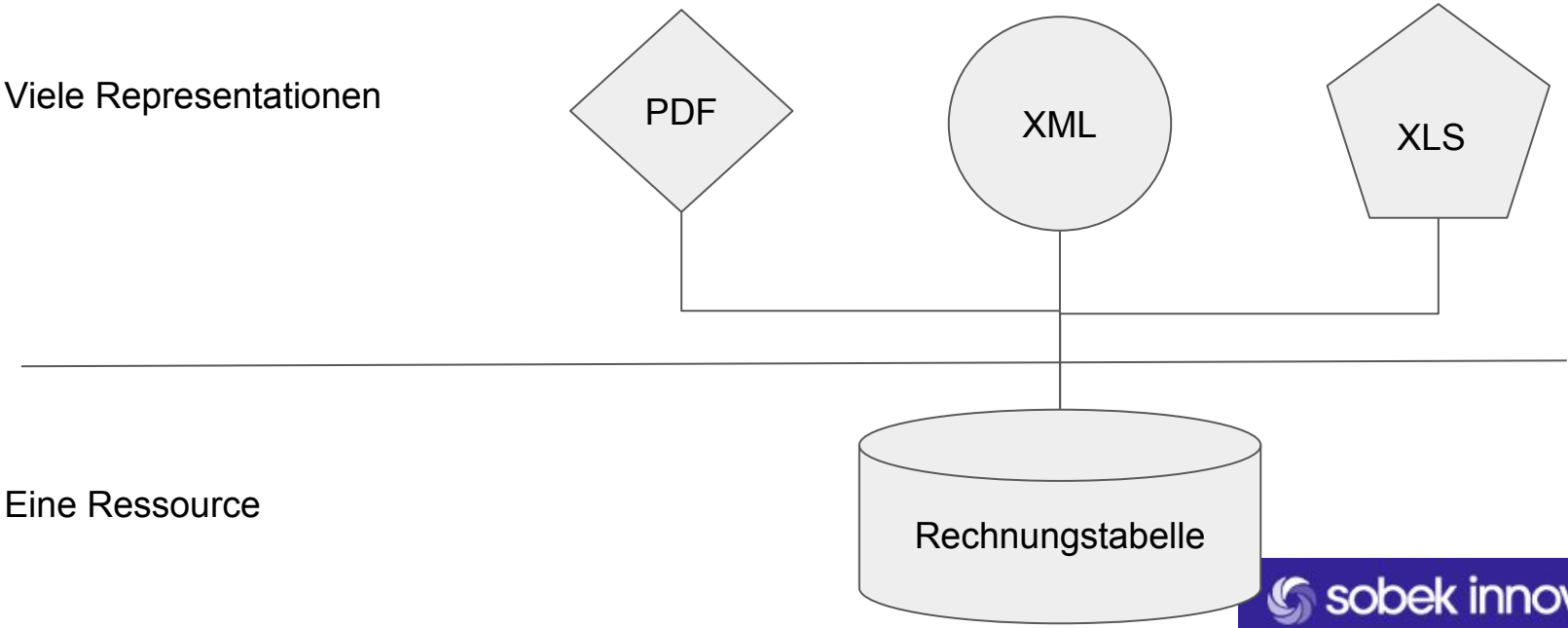


APIs - REST (Representational State Transfer)

Abbildung von Ressourcenorientierten APIs

Eine Ressource (intern) unterschiedliche Representationen z.B. Rechnung

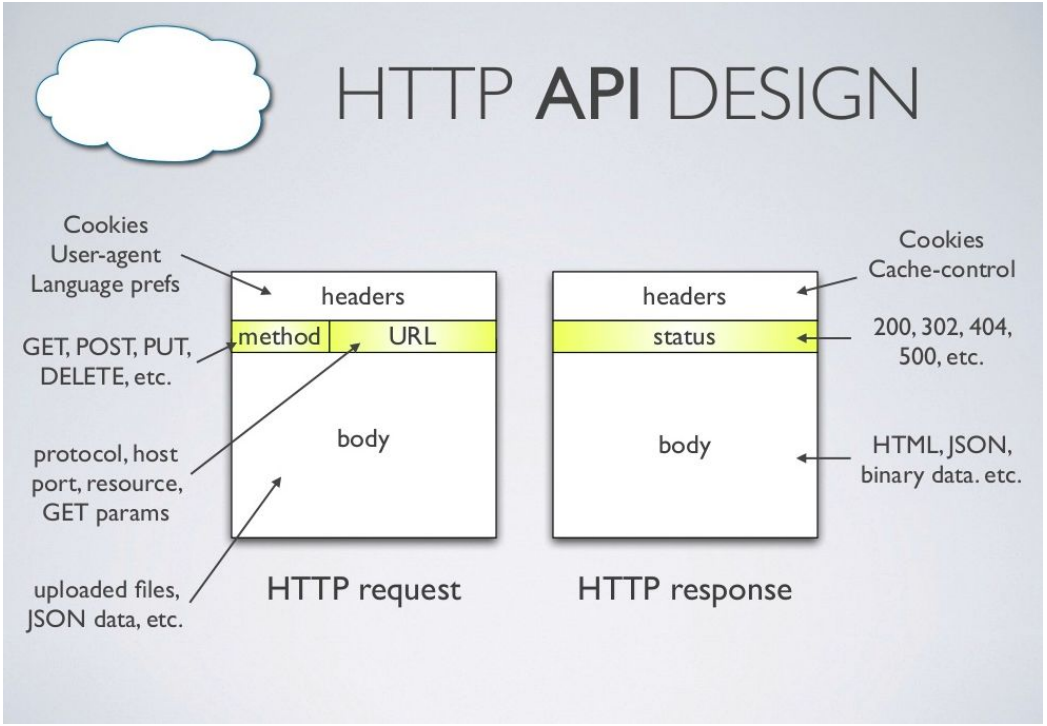
Viele Representationen



Eine Ressource

HTTP Basics

Webservices - APIs - HTTP = Plattformneutralität



aus Slideshare

sobek innovations

REST HTTP Headers

REST - HTTP Status Codes

Name	Beschreibung	Syntax
Accept	Der Accept-Header sagt dem Server, welche Mime-Types er interpretieren kann. Im Umfeld von REST teilt mit diesen Header der Client dem Server mit, welche Repräsentation der Ressource er anfordert. Bspw: Accept: application/json	Accept: <MIME_type>/<MIME_subtype> Accept: <MIME_type>/* Accept: */*
Content-Type	Der Content-Type beschreibt das Format der übermittelten Nachricht sowahl im Response (Server-Antwort) also auch im Request (Client-Anfrage). Bspw: Content-Type: application/json bei POST http://localhost/customers	Content-Type: text/html; charset=utf-8 Content-Type: multipart/form-data

REST - HTTP Status Codes

Name	Beschreibung	Syntax
Authorization	Mit dem Authorization-Header werden Authentifizierungsinformationen übertragen (Basic Auth o. Digest).	Authorization: <type> <credentials>
Cookie	In Cookies oder im eigenen Request-Headern (bspw: x-oauth-token => Problem Bilder) werden authorisierungstokens, welche dem Server eine Validierung erlauben, übertragen.	Cookie: <cookie-list> Cookie: name=value Cookie: name=value; name2=value2

REST // Wichtige HTTP Status Codes

REST - HTTP Status Codes

Status Code	Nachricht	Beschreibung
200	Success	Erfolgreiche Verarbeitung.
201	Created	Erfolgreiche Erstellung.
204	Success	Erfolgreiche Verarbeitung ohne Antwort-Inhalt.
400	Bad Request	Ressourcen-URI unbekannt oder falsche Headerinformationen z.B. falsche Accept-, Content-Types oder Authentication-Headers

REST - HTTP Status Codes

Status Code	Nachricht	Beschreibung
401	Unauthorized	Der Benutzer ist nicht autorisiert die API zu verwenden.
403	Forbidden	Der Benutzer darf nicht die angefragte Operation durchführen. Auch weiterführende Geschäftsregeln mgl. z.B. die Änderung einer Adresse obwohl dies über ACL-Regeln verboten worden ist.
404	Not found	Die Ressource existiert nicht.
405	Method not allowed	Der Benutzer darf die angefragt Method nicht durchführen (z.B. DELETE auf Rechnung). Oder die Methode ist nicht implementiert.

REST - HTTP Status Codes

Status Code	Nachricht	Beschreibung
406	Not acceptable	Der angefragte Datentyp (Accept-Type) wird von der API nicht unterstützt
415	Unsupported media type	Der versendete Datentyp (Content) wird von der API nicht utnerstützt.

REST & SOAP

Ein Vergleich

Webservices - APIs - Ressourcen- vs. Serviceorientierung

Ressourcenorientierte Architekturen

Datenorientierte Betrachtung.
Hier liegt der **Focus** auf **Daten** ohne Funktionen.

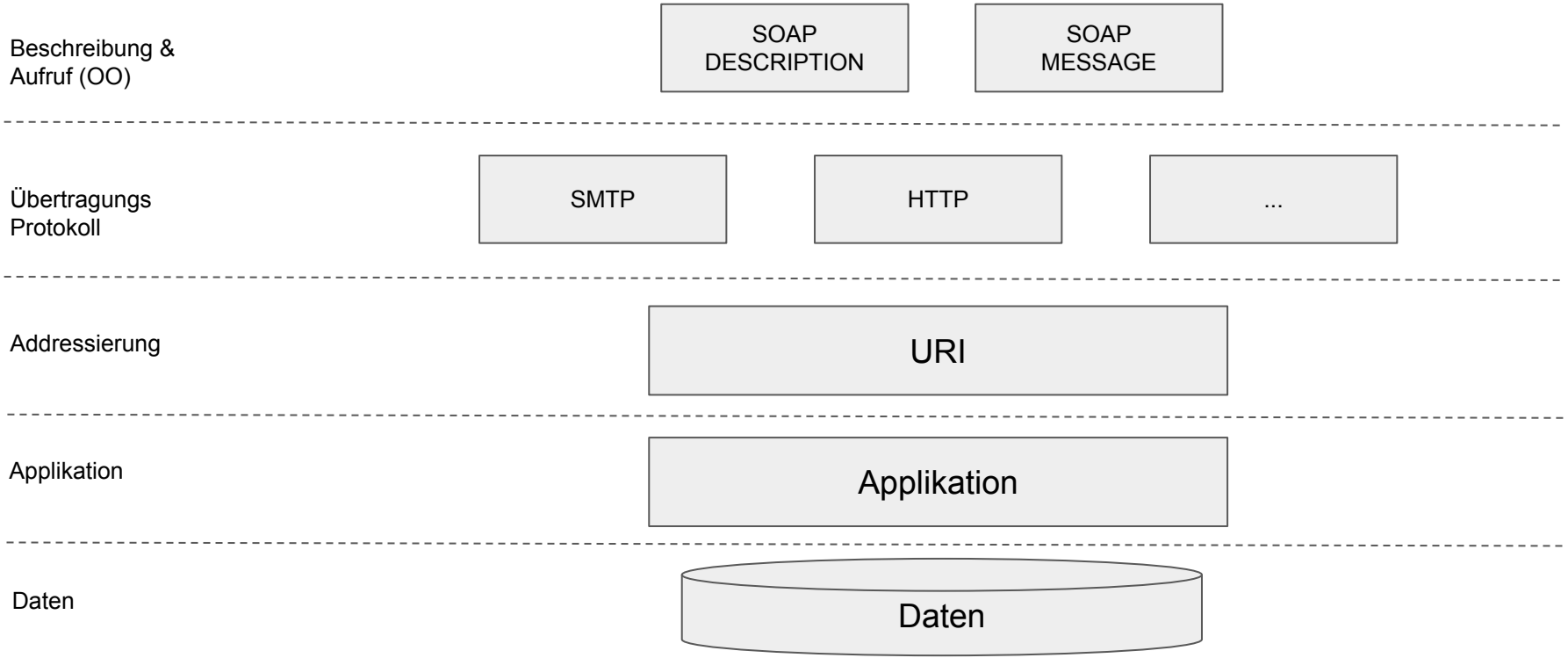
- Ressourcen (innen)
- Repräsentationen (aussen)
- CRUD auf HTTP
- URI-Design

Serviceorientierte Architekturen

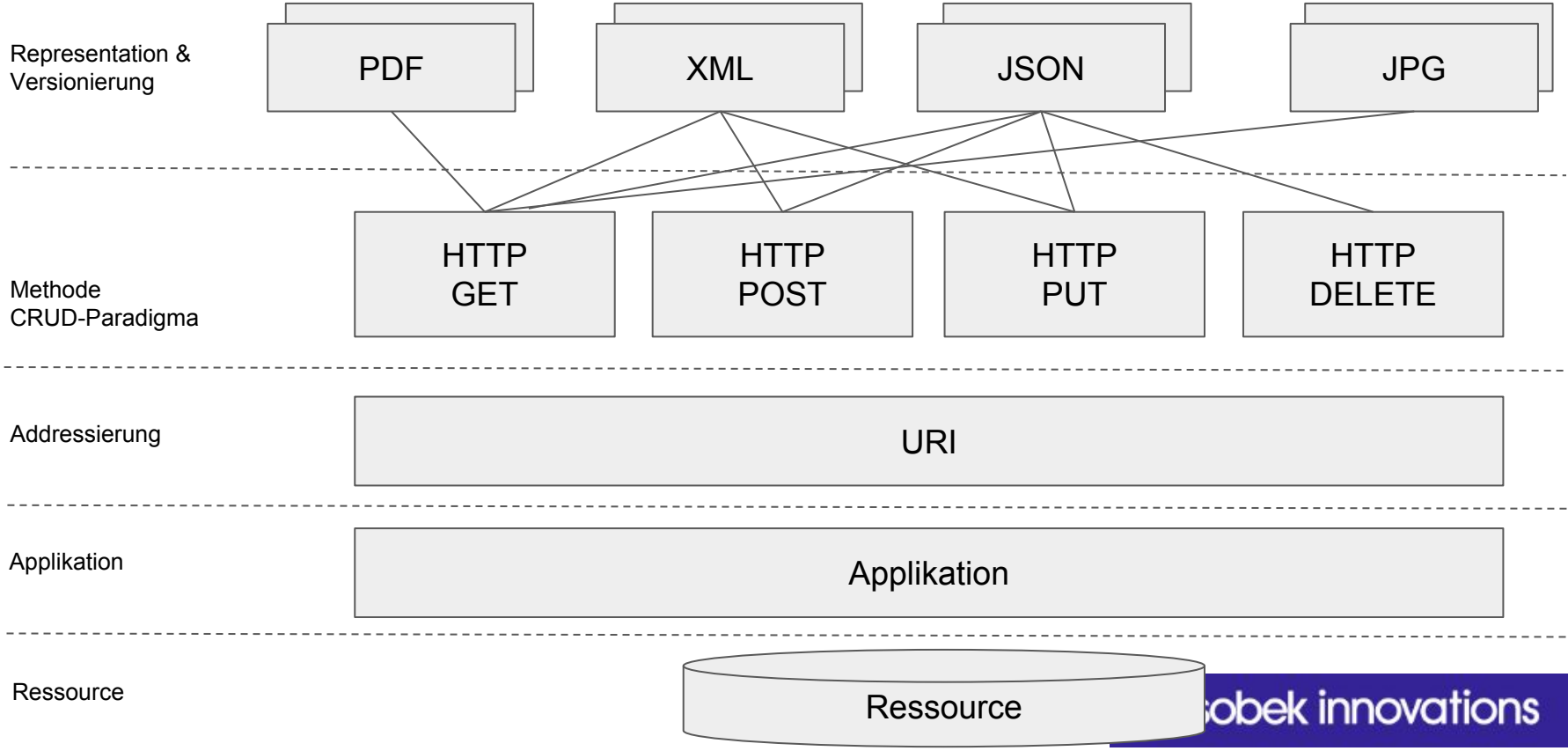
Objektorientierte Betrachtung.
Hier liegt der **Focus** auf Klassen (**Daten und Funktion**).

- Klasse bzw. deren Interface wird als Schnittstelle angeboten
- Funktionen wie Login sind hier ebenfalls abgebildet. Was bei Ressourceorientierten Architekturen so nicht der Fall ist.

Schichtenmodell bei SOAP



Schichtenmodell bei REST



APIs - REST (Representational State Transfer)

Abbildung von Ressourcenorientierten APIs

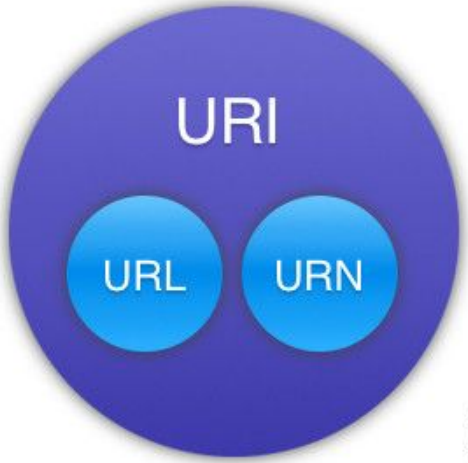
Angelehnt am CRUD-Paradigme (Create, Read, Update, Delete)

Create als HTTP POST Methode	Update als HTTP PUT Methode
Read als HTTP GET Methode	Delete als HTTP DELETE Methode

Was darf ich
als HTTP OPTIONS Methode

APIs - URI vs. URL

Was ist was?



URI = Uniform Resource Indentifier
URL = Uniform Resource Locator
URN = Uniform Resource Name

Eine URL (Unified Ressource Locator) zeigt an wo etwas ist.

z.B.
https://sobek-innovations.de/impressum

Eine URN (Unified Resource Name) zeigt an was etwas ist.

z.B. urn:isbn:0451450523
z.B. urn:ietf:rfc:2648

Beides ist eine URI
(Unified Resource Identifier)

sobek innovations

API Versionierung

Methoden

API - Versionierungsstrategien - Per Subpath

http://example.com/api/v0.1.0/invoice/1234

http://example.com/api/v0.2.alpha-1/invoice/1234

http://example.com/api/v1.0.0/customer/1234

Diese Variante stellt die häufigste Versionierungsform dar und ist mit geringsten Automatisierungsaufwänden verbunden.

API - Versionierungsstrategien - Accept- u. Vendortypes

GET /invoices/1234 HTTP/1.1

Host: api.example.com

Accept: application/json;**version=0.2-alpha-1**

Diese Variante entspricht am stärksten der HTTP-Spezifikation, wird allerdings am seltensten genutzt.

API - Versionierungsstrategien - Per Subdomain

http://v0.1.api.example.com/invoice/1234

http://v0.2.alpha-1.api.example.com/invoice/1234

http://v1.0.0.api.example.com/customer/1234

Diese Variante wird oft als aufwändig angesehen, weil hier ein URL-Rewriting (per Rewrite Rules) oder DNS-Einstellungseingriff (per DNS-APIs) notwendig wird. Beides ist aber automatisierbar.

Representationen

Datenformate

Datenformate // Austauschformate

JSON und XML sind die häufigsten Austauschformate (z.B. UI Interaktion o. Datentransformation z.B. XML2Rechnung)

application/json

```json
{
  "firstName": "Rafael",
  "secondName": "Sobek",
  "address": {
    "street": "Kriegsstrasse 216",
    "city": "Karlsruhe"
  }
}
```

text/xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <address>
        <city>Karlsruhe</city>
        <street>Kriegsstrasse 216</street>
    </address>
    <firstName>Rafael</firstName>
    <secondName>Sobek</secondName>
</root>
```

Datenformate // Binäre Daten im Austauschformat

Mit Zuhilfenahme der Base64-Encodierung können binäre Daten in Austauschformaten genutzt werden bzw. übertragen werden.

application/json

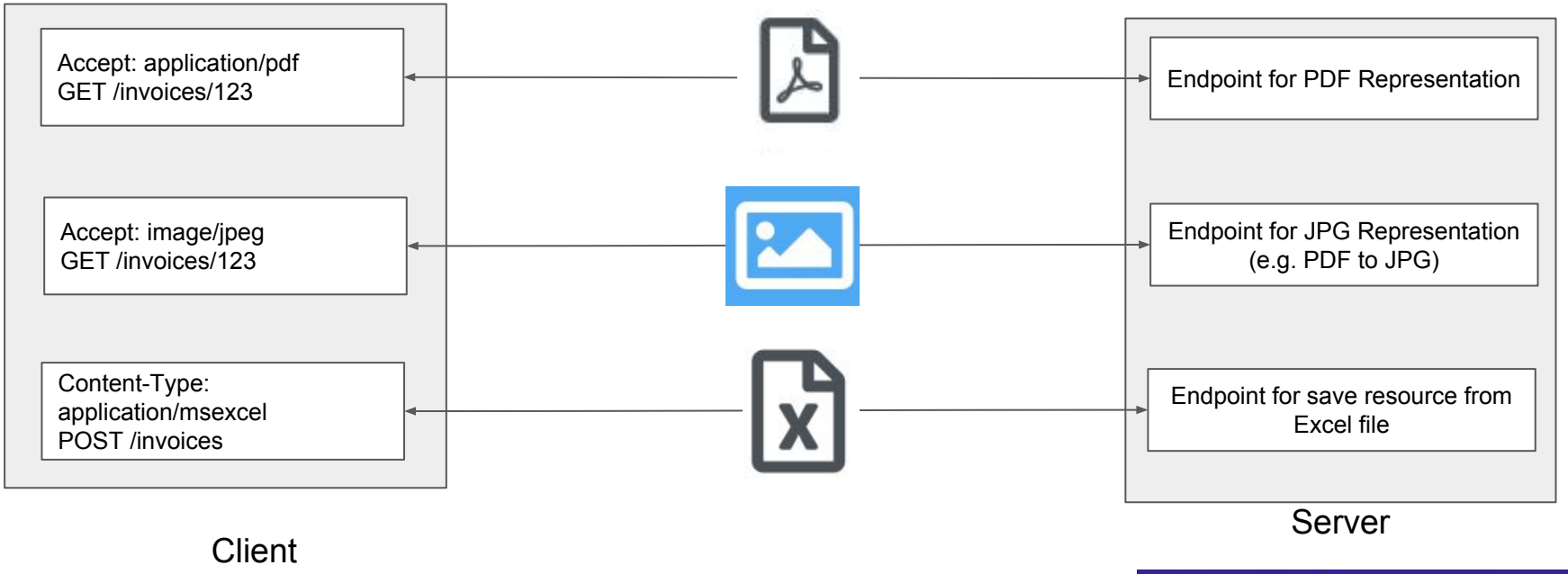
```
{
  "firstName": "Rafael",
  "secondName": "Sobek",
  "address": {
    "street": "Kriegsstrasse 216",
    "city": "Karlsruhe"
  },
  "picture": "4AAQSkZJRgABA…"
}
```

text/xml

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <address>
      <city>Karlsruhe</city>
      <street>Kriegsstrasse 216</street>
  </address>
  <firstName>Rafael</firstName>
  <secondName>Sobek</secondName>
  <picture base64data="4AAQSkZJRgABA…"/>
</root>
```

Datenformate // Binäre Daten als Representation

Binäre Daten (falls sie die Ressource repräsentieren) können per “Accept”- (lesend => GET) oder den “Content-Type”-Header gelesen oder gespeichert werden.



Webservice Security

REST & SOAP

Webservice Security - Überblick

Verschlüsselung
Übertragungsweg
(Asymmetisch vs. Symmetrisch)

Authentifizierung
(zentral vs. dezentral)

Authorisierung
(zentral vs. dezentral)

HTTP Restriktionen
HTTP-Methoden, - Header etc. begrenzen

Metering & Logging
Vorfeldüberprüfung der Anfragen
Filter (ContentTypeChecks, etc.)

Content Validation
Vorfeldüberprüfung der Anfragen
Filter (ContentTypeChecks, etc.)

Schematisierung
JSON Schema p. XSD Schema

Typisierung
Strenge vs. schwache Typisierung

Verschlüsselung Kommunikationsweg

Mit **SSL** in der **HTTP-Kommunikation** wird der **Schutz der übertragenen Daten** sichergestellt.
Dazu gehören auch die Authentifizierungs- sowie Authorisierungsdaten.

sobek innovations

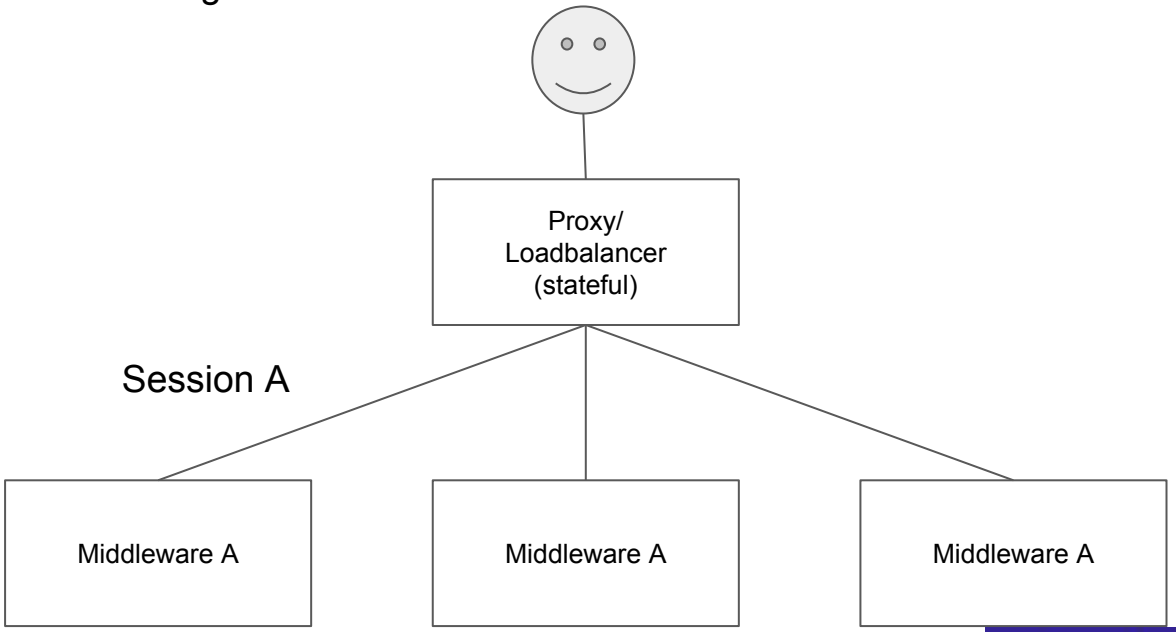
Identity & Accessmanagement

Das **Identitymanagement** beschäftigt
sich vordergründig mit der **Authentifizierung**.

Das **Accessmanagement** beschäftigt
sich vordergründig mit der **Autorisierung**.

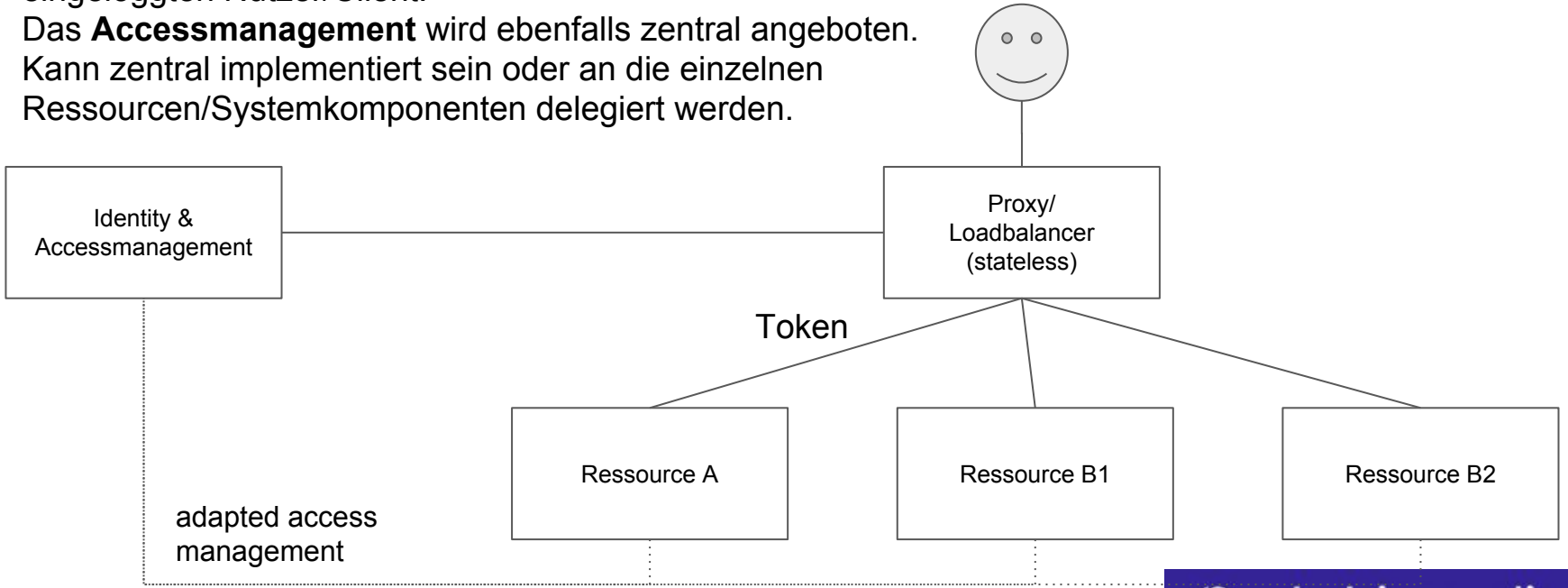
Identity & Accessmanagement // Monolithische Strategie

Bei monolithischen Systemen (Middlewares) erfolgt klassischerweise eine Sessionerzeugung. Benutzername und Password werden an ein Knoten übertragen. Dieser verifiziert und erzeugt bei erfolgreichen Authentifizierung eine Session.



Identity & Accessmanagement // Dezentrale Strategie I

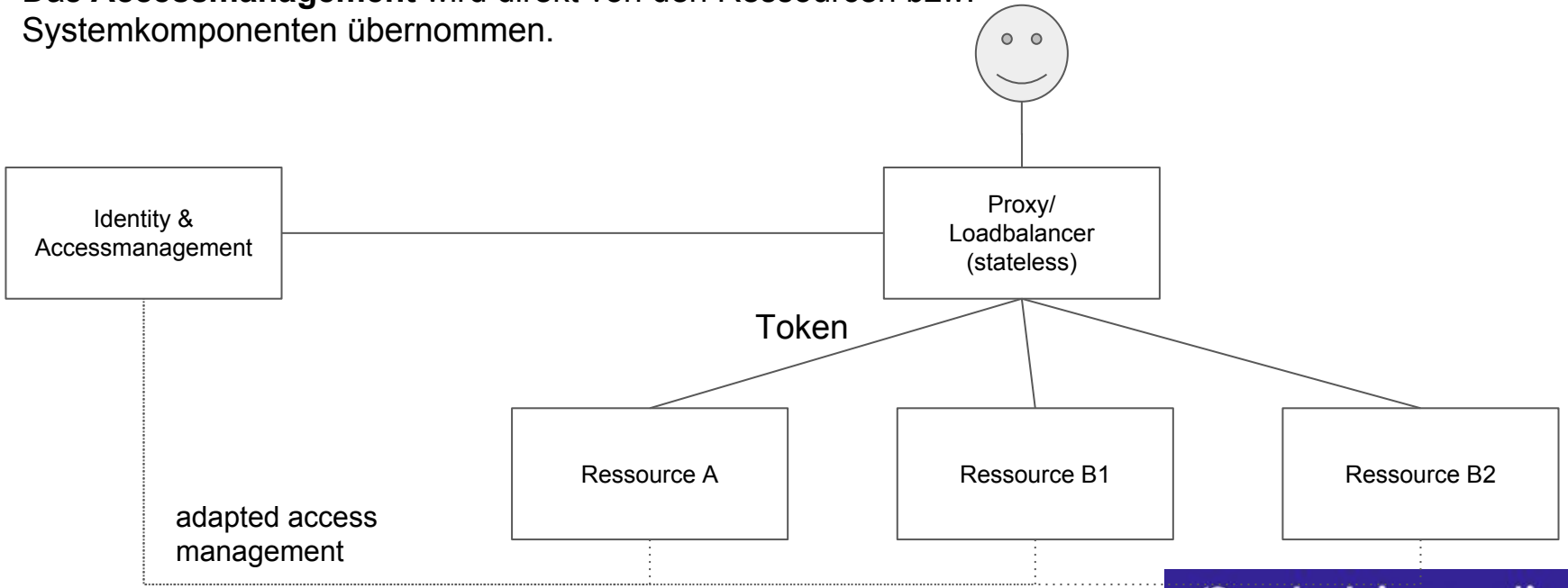
Das **Identitymanagement** übernimmt vollständig eine zentrales Identitymanagement-System. Dieses stellt sicher, dass die Authentifizierung korrekt funktioniert. Ein erzeugtes Token identifiziert den eingeloggten Nutzer/Client.
Das **Accessmanagement** wird ebenfalls zentral angeboten. Kann zentral implementiert sein oder an die einzelnen Ressourcen/Systemkomponenten delegiert werden.



Identity & Accessmanagement // Dezentrale Strategie II

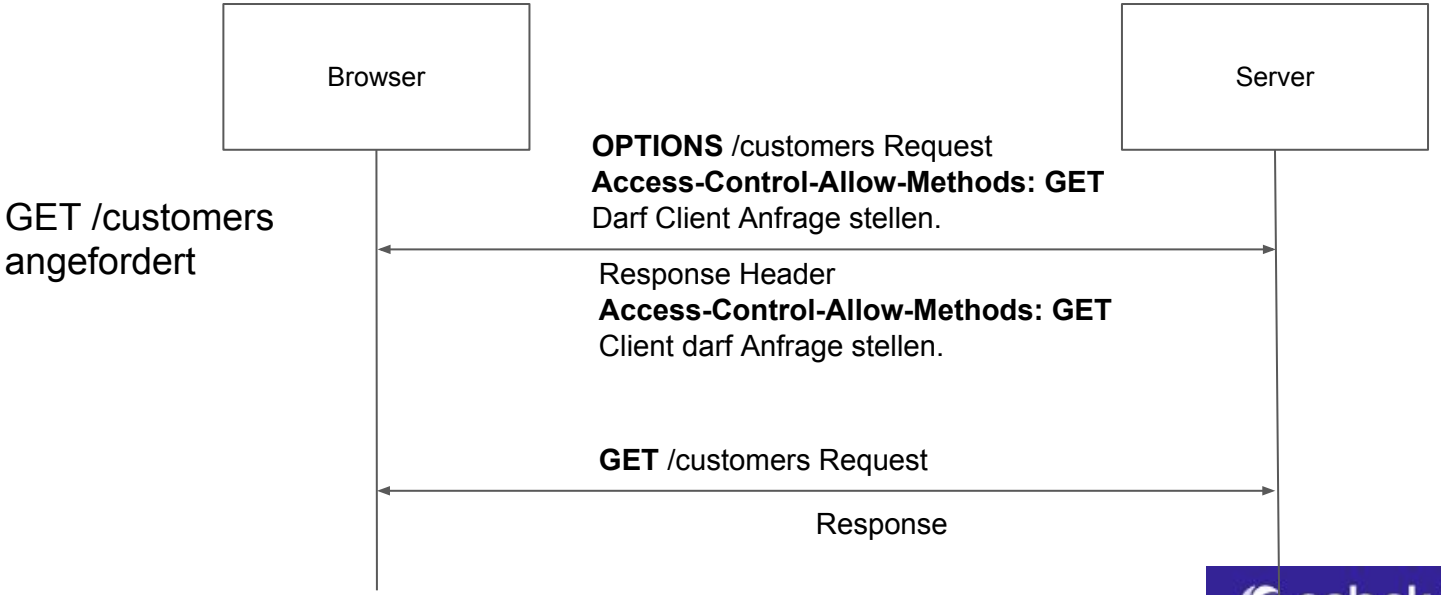
Das **Identitymanagement** ist hier ebenfalls zentral aufgestellt.

Das **Accessmanagement** wird direkt von den Ressourcen bzw. Systemkomponenten übernommen.



HTTP Restriktion // server- u. clientseitig

Serverseitig: Der Server bietet nur ausgewählte HTTP-Methoden, Header etc. an.
Client- u. Serverseitig: Die Cross-Origin-Policy verhindert die Anfragen von Browsern auf andere Domains.



Typisierung

Streng-typisierte Programmiersprachen sind bereits bzgl. Schematisierung stark aufgestellt.

```
public class Customer {
   private String name;
   private Date birthdate;
   public Date getBirthdate() {
       return birthdate;
   }
   public void setBirthdate(Date birthdate) {
       this.birthdate = birthdate;
   }
   public String getName() {
       return name;
   }
   public void setName(String name) {
       this.name = name;
   }
}
```

Schwach-typisierte Programmiersprachen benötigen immer noch eine zusätzliche Schemavalidierung.

```
Customer = function(name, birthdate) {
   this.name = name;
   this.birthdate = birthdate;
}
```

Zus. Businessregeln

```
public class Customer {
   private String name;
   private Date birthdate;
   public Date getBirthdate() {
       return birthdate;
   }
   public void setBirthdate(Date birthdate) {
       this.birthdate = birthdate;
   }
   public String getName() {
       return name;
   }
   public void setName(String name) {
       //Businessregeln
       Long birthYear = this.getBirthdate().getYear();
       if (birthYear != null &&
           this.getBirthdate().getYear() < 2010 && name == "Rafael") {
           throw new Exception();
       }
       this.name = name;
   }
}
```

Schematisierung

JSONSchema

```
{
   "type": "object",
   "properties": {
       "title": {
           "type": "string",
           "minLength": 2,
           "maxLength": 10

       },
       "phone": {
           "type": "string",
           "pattern": "^(\\([0-9]{3}\\))?[0-9]{3}-[0-9]{4}$"

       },
       "alignment": {
           "type": "string",
           "enum": [
               "left",
               "right"
           ],
           "default": "left"
       }
   }
}
```

XMLSchema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema>
   <sequence>
     <element minOccurs="0" name="title">
       <simpleType>
         <restriction base="string">
           <minLength value="2"/>
           <maxLength value="10"/>
         </restriction>
       </simpleType>
     </element>
     <element minOccurs="0" name="phone">
       <simpleType>
         <restriction base="string">
           <pattern value="^(\([0-9]{3}....$" />
         </restriction>
       </simpleType>
     </element>
     <element minOccurs="0" name="alignment">
       <simpleType>
         <restriction base="string">
           <enumeration value="left"/>
           <enumeration value="right"/>
         </restriction>
       </simpleType>
     </element>
   </sequence>
 </complexType>
</schema>
```

Typisierung

Streng-typisierte Programmiersprachen sind bereits bzgl. Schematisierung stark aufgestellt.

```
public class Customer {
   private String name;
   private Date birthdate;
   public Date getBirthdate() {
       return birthdate;
   }
   public void setBirthdate(Date birthdate) {
       this.birthdate = birthdate;
   }
   public String getName() {
       return name;
   }
   public void setName(String name) {
       this.name = name;
   }
}
```

Schwach-typisierte Programmiersprachen benötigen immer noch eine zusätzliche Schemavalidierung.

```
Customer = function(name, birthdate) {
   this.name = name;
   this.birthdate = birthdate;
}
```

Message Exchange Patterns

Welche gibt es?

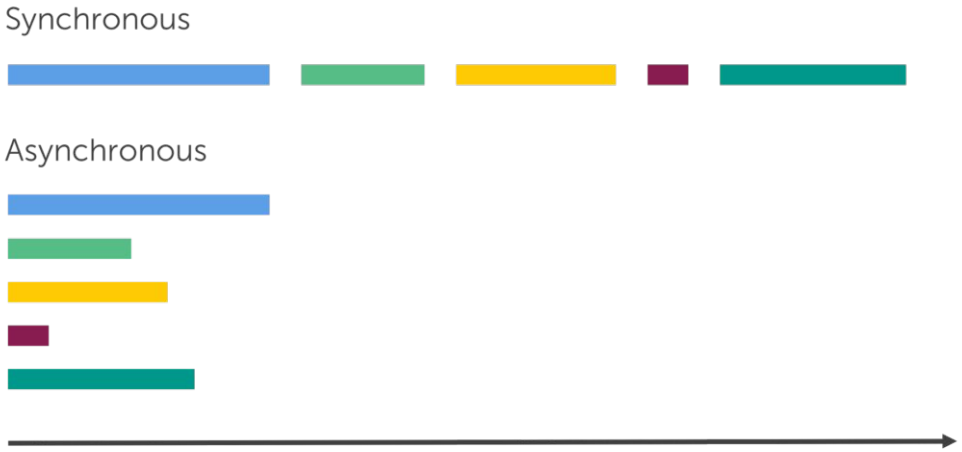
Message Exchange Patterns // Synchrone vs. Asynchrone Kommunikation

Synchrone Kommunikation

Die **Aufrufe erfolgen** client- und serverseitig **aufeinanderfolgend**. Der **Client wartet** mit den nächsten Befehl **bis** der **Server geantwortet** hat.

Asynchrone Kommunikation

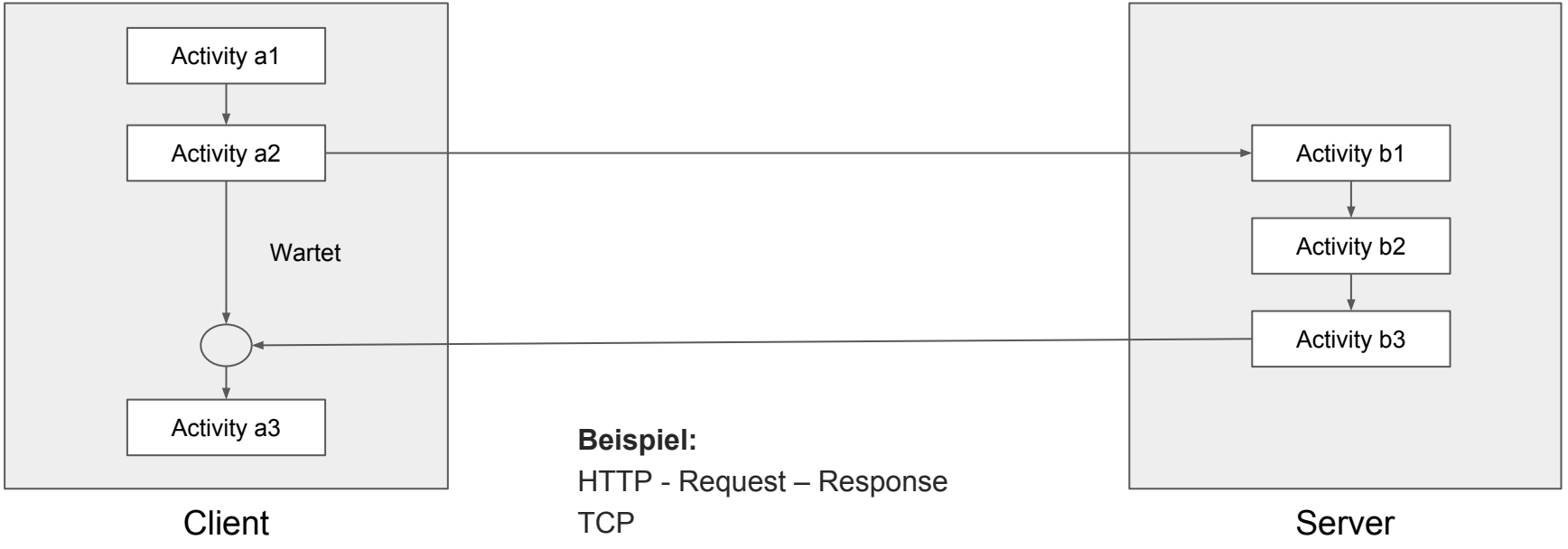
Die **Aufrufe** (Befehle) **auf dem Client** laufen **unabhängig von getätigten Anfragen an den Server** weiter**.**



Aus: https://www.sovrn.com

Message Exchange Patterns // Synchrone Kommunikation // Synchrone Anfrage

Wird auch als **blockierender Nachrichtenaustausch** bezeichnet. Der **Client wartet** solange bis die Antwort des Servers empfangen worden ist.

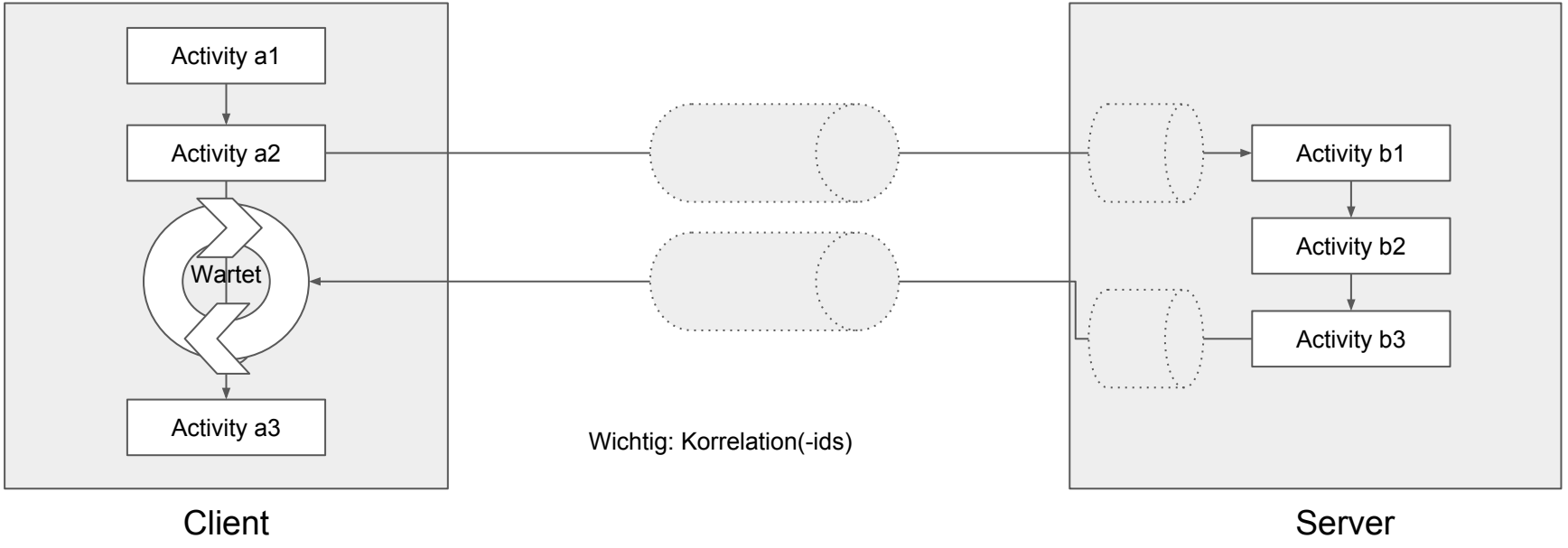


Beispiel:
HTTP - Request – Response
TCP

Ganz Wichtig: Timeouts

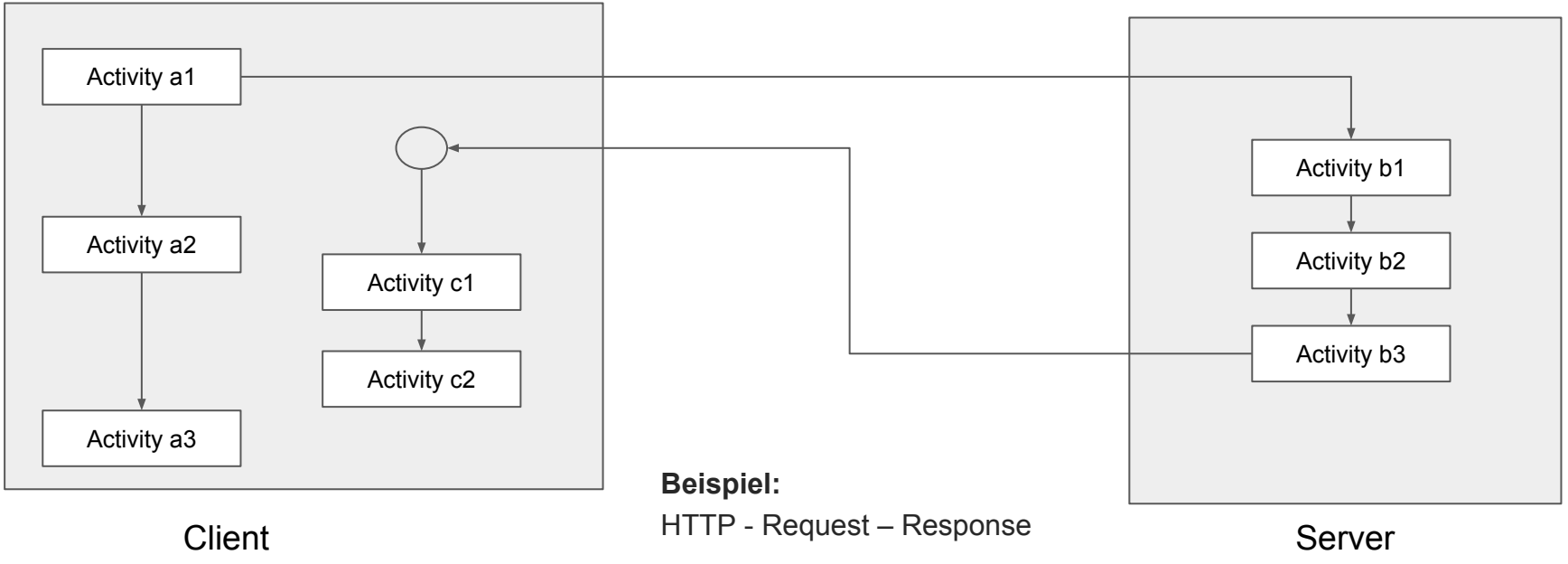
Message Exchange Patterns // Synchrone Kommunikation // Synchroner Empfangsmodus

Wird auch als **blockierender Nachrichtenaustausch** bezeichnet. Der **Client wartet in Form einer Schleife** solange bis die Antwort des Servers empfangen worden ist.



Message Exchange Patterns // Asynchrone Kommunikation // Request-Callback

Wird auch als **nicht-blockierender Nachrichtenaustausch** bezeichnet. Der **Client triggert eine Serveranfrage an und hinterlegt eine Callbackfunktion,** welche **bei Eintreffen** der Nachricht **ausgeführt** wird.

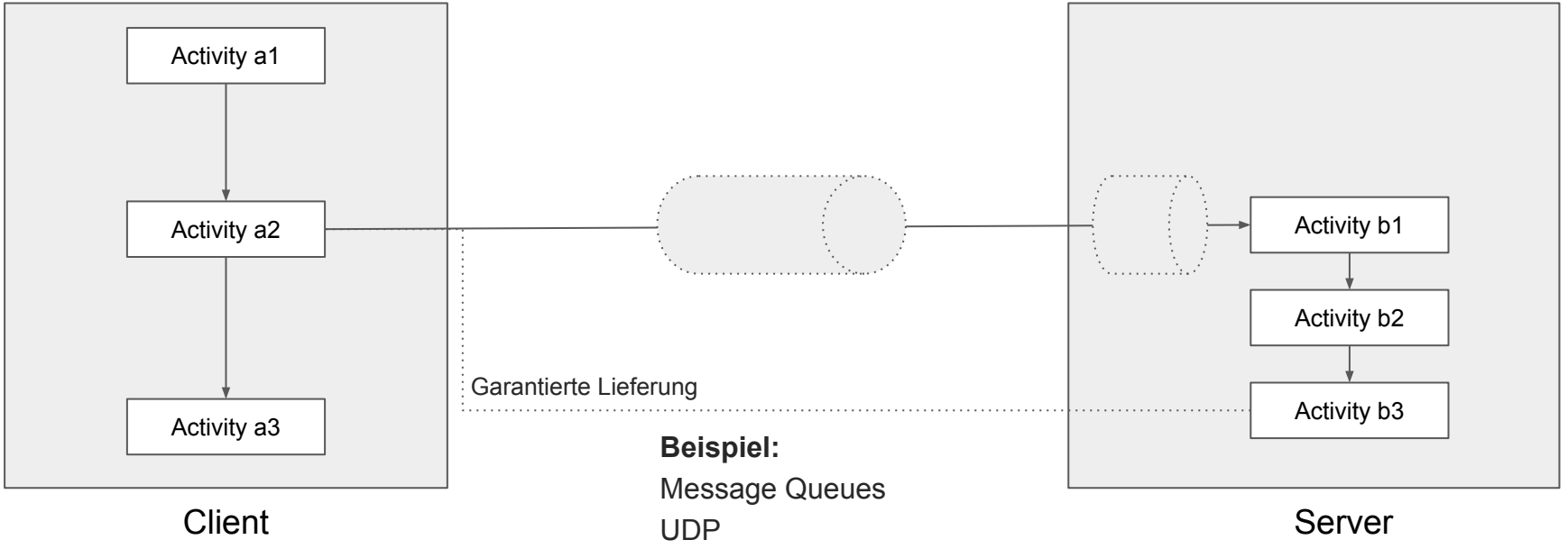


Beispiel:
HTTP - Request – Response

Wichtig: Timeouts

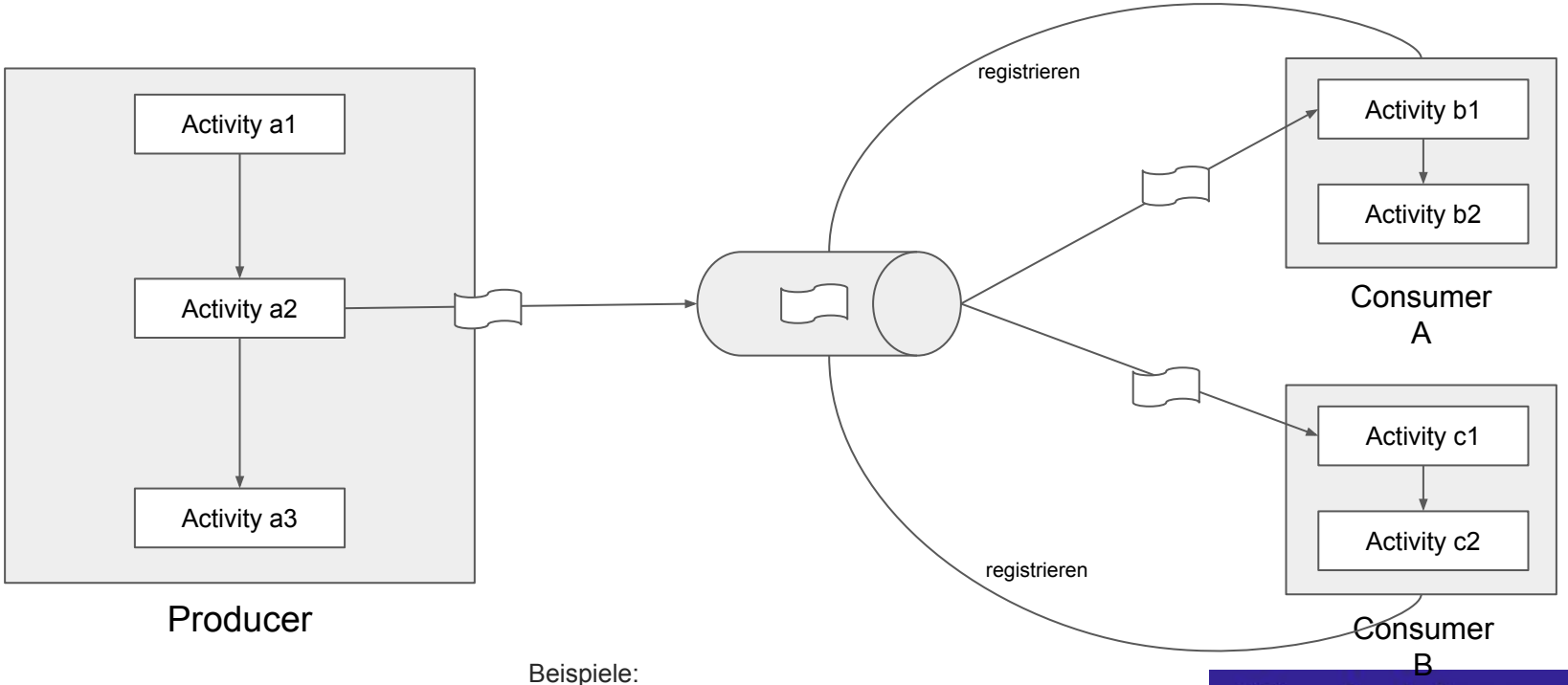
Message Exchange Patterns // Asynchrone Kommunikation // Fire & Forget

Der Client-Aufruf wird versendet **ohne dass eine Antwort erwartet** wird (bspw. Briefeinwurf). Der Client führt seine eigene Logik unabhängig von der Verarbeitungslogik mgl. Empfänger weiter aus. Dabei wird unterschieden zwischen **garantierter und nicht garantierter Lieferung**.



Message Exchange Patterns // Asynchrone Kommunikation // Publish & Subscriber

Zwei Interessenten (**Consumer**) **registrieren** sich für **bestimmte** Nachrichten (**Messages**) im Nachrichteneingang (**Queue**).



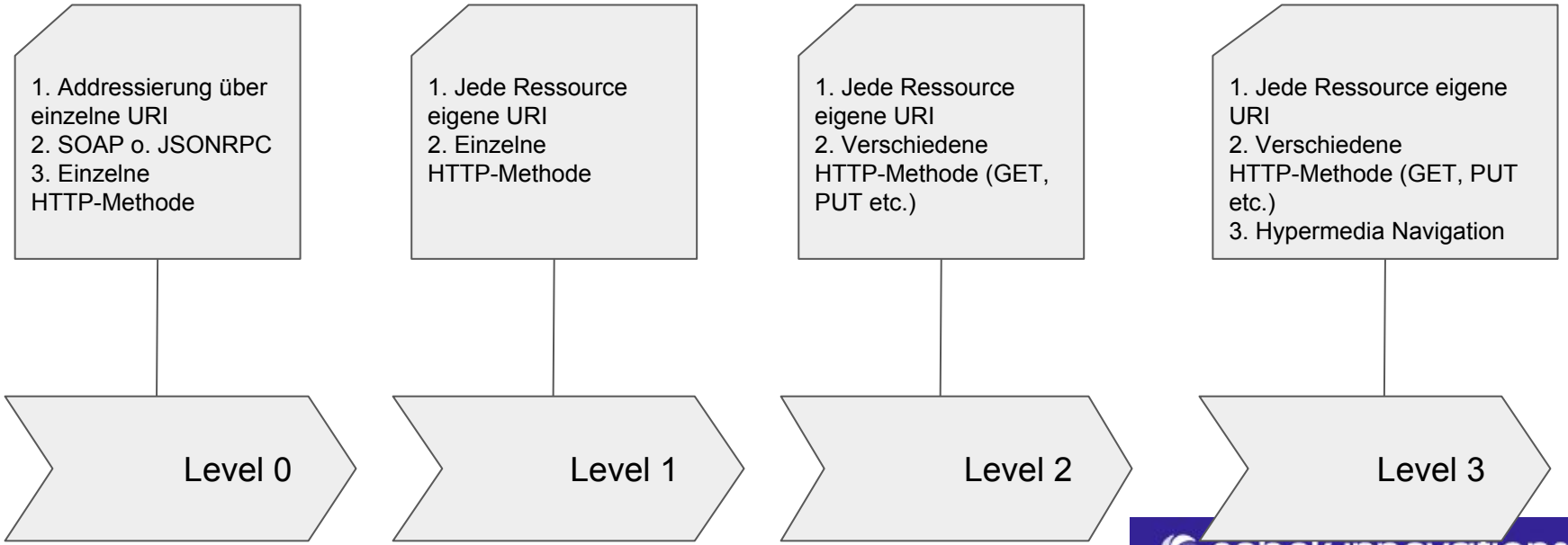
Beispiele:
JMS, Java Swing, Javascript, etc.

Richardson Maturity Model

Reifegradmodell f. REST-Services

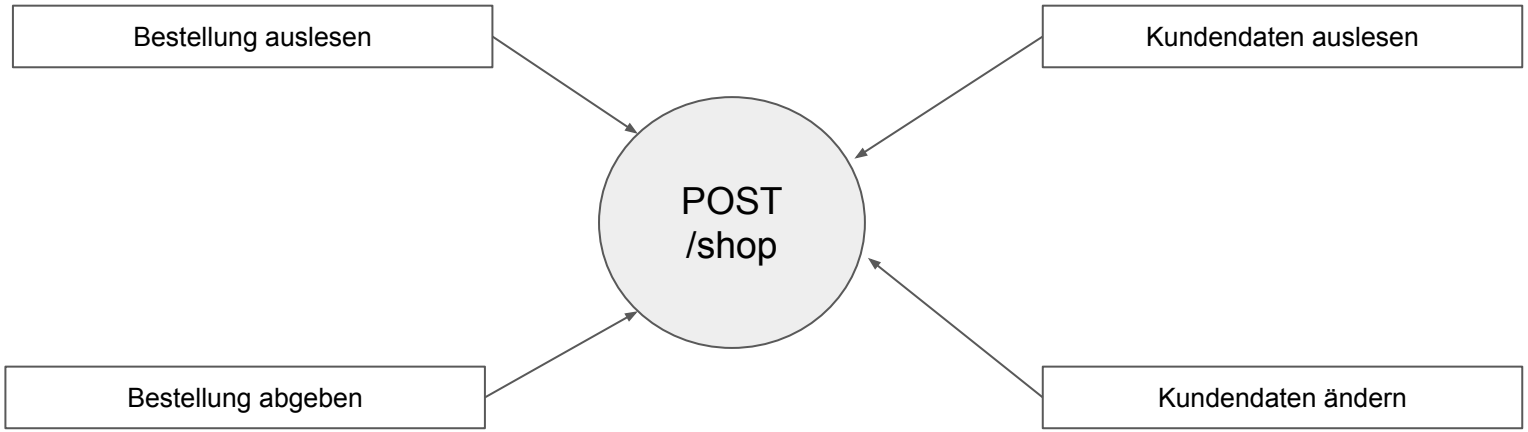
Datenformate // Binäre Daten als Representation

Beschreibt den Reifegrad eines RESTful-Services



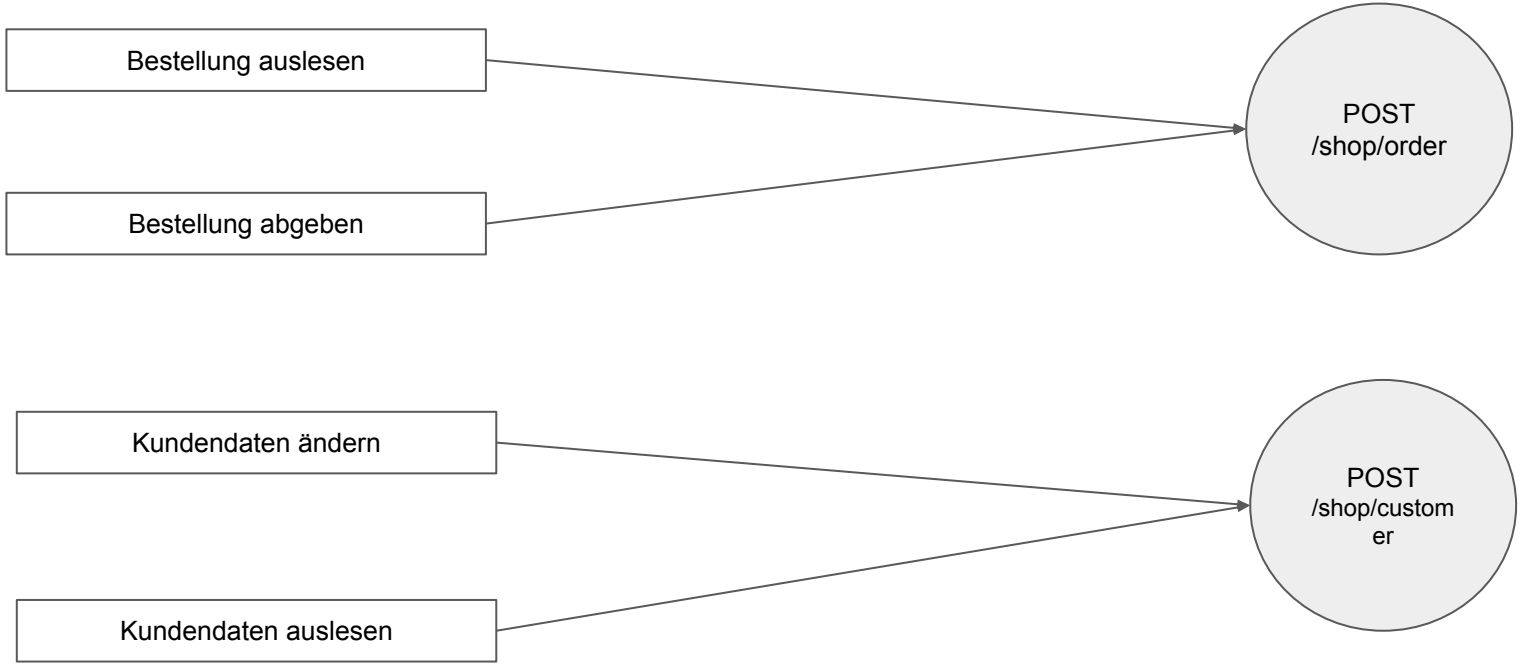
Level 0

Addressierung über einzelne URI. Einzelne HTTP-Methode.



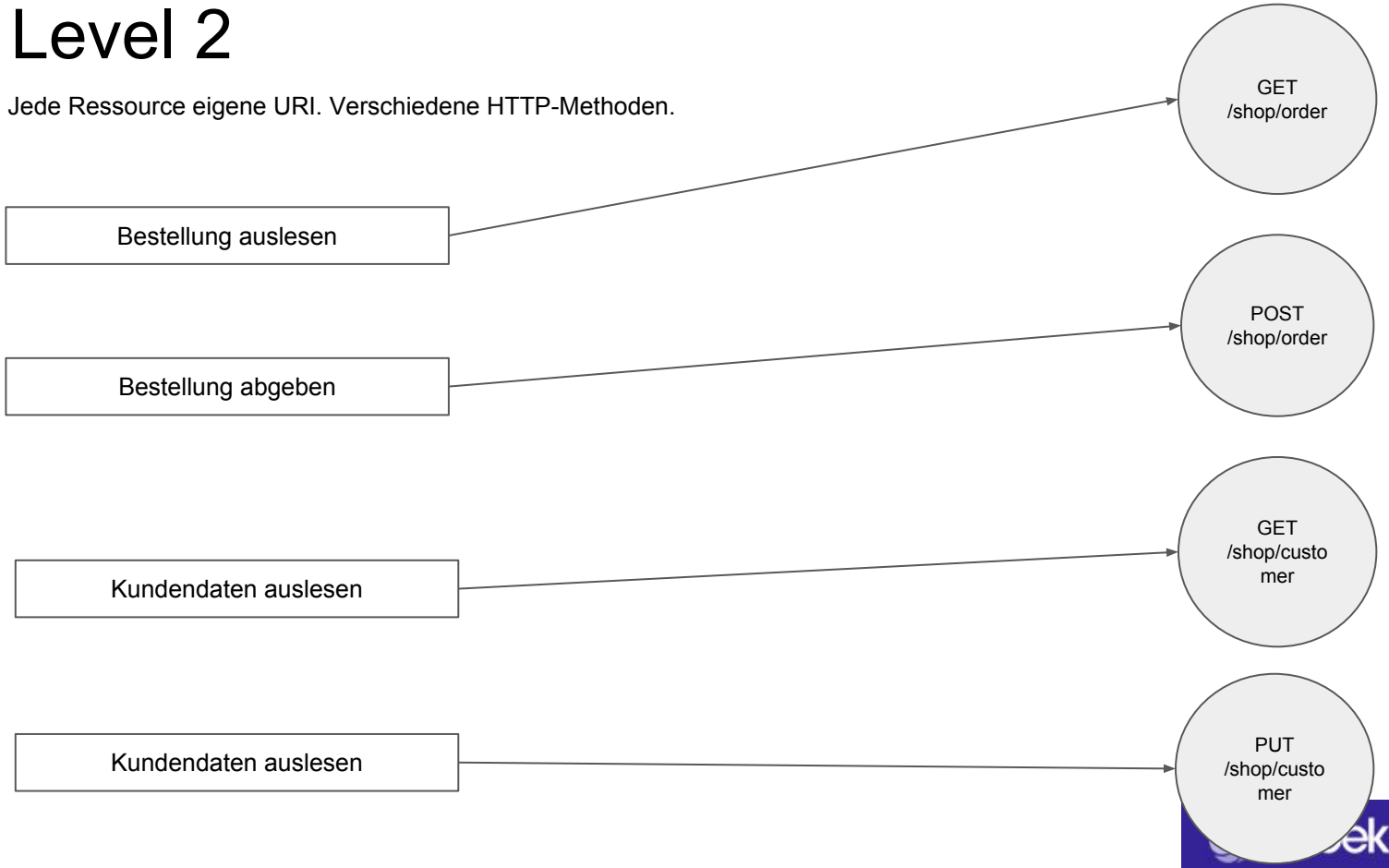
Level 1

Jede Ressource eigene URI. Einzelne HTTP-Methode.



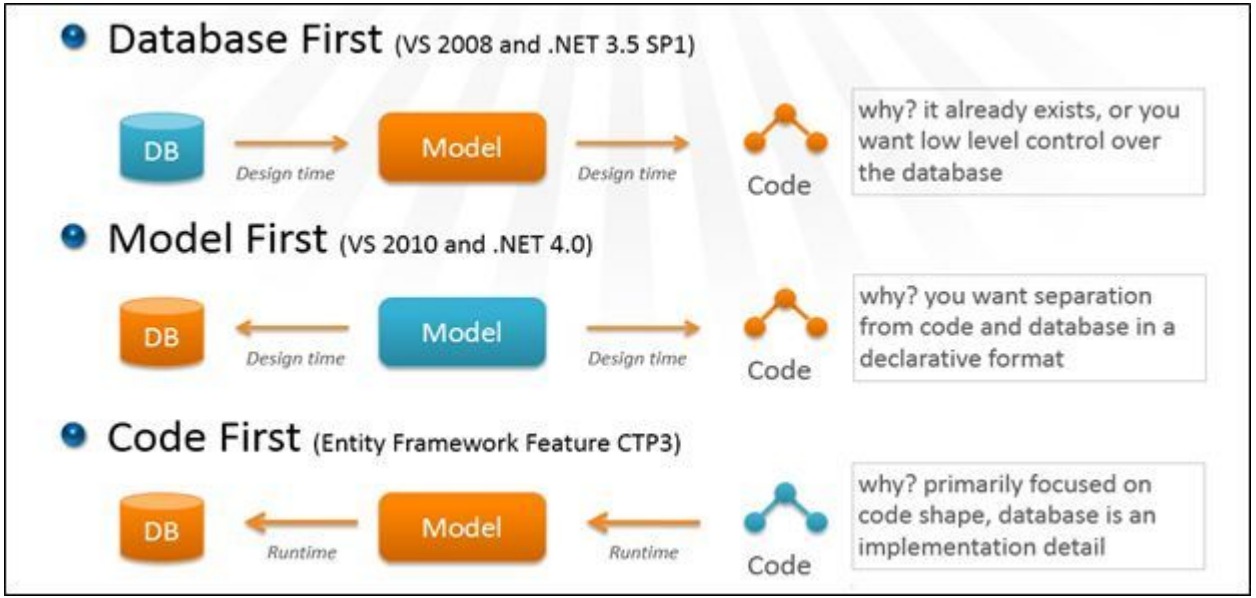
Level 2

Jede Ressource eigene URI. Verschiedene HTTP-Methoden.



Contract First vs. Code First

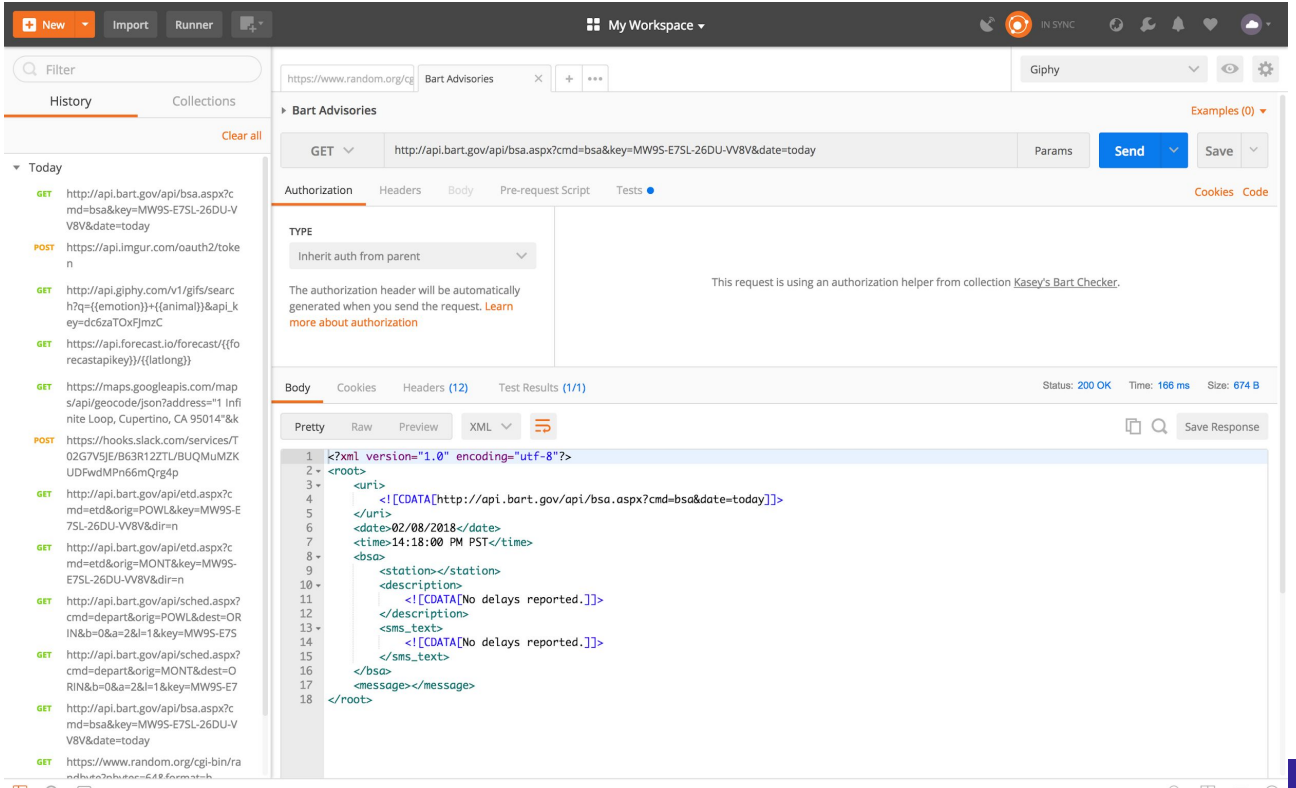
SOAP - Contract First vs. Code First vs. Database First



aus c-sharpcorner.com

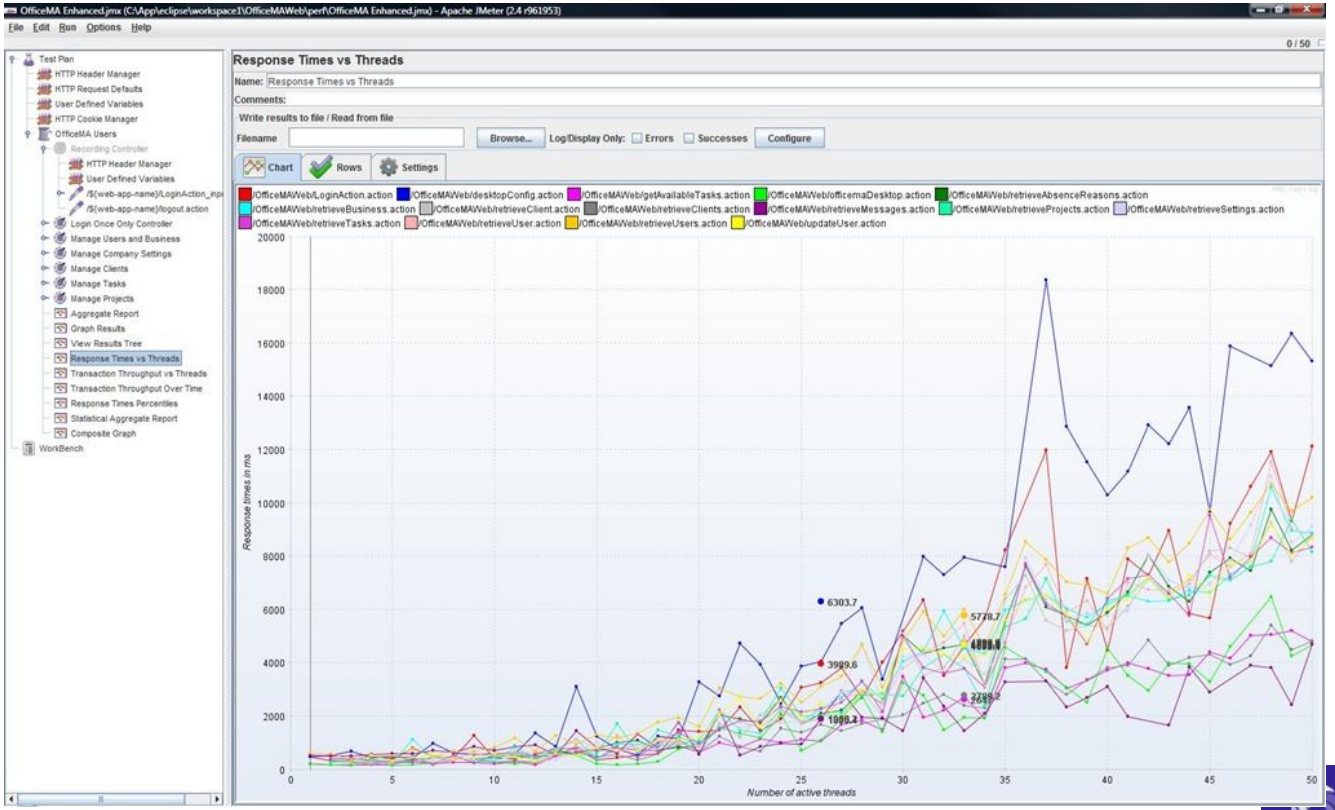
Entwicklungstools

Postman // REST-Testing



sobek innovations

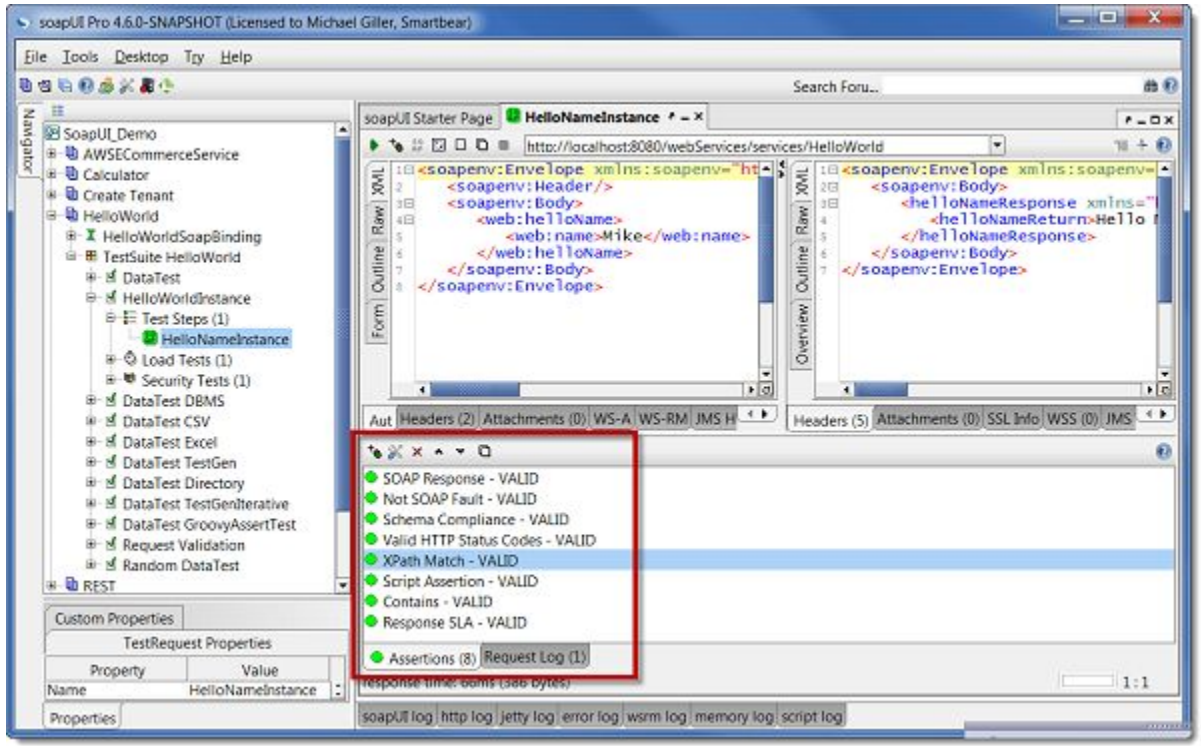
JMeter // REST u. SOAP // Tests Massenanfragen



aus omerio.com

sobek innovations

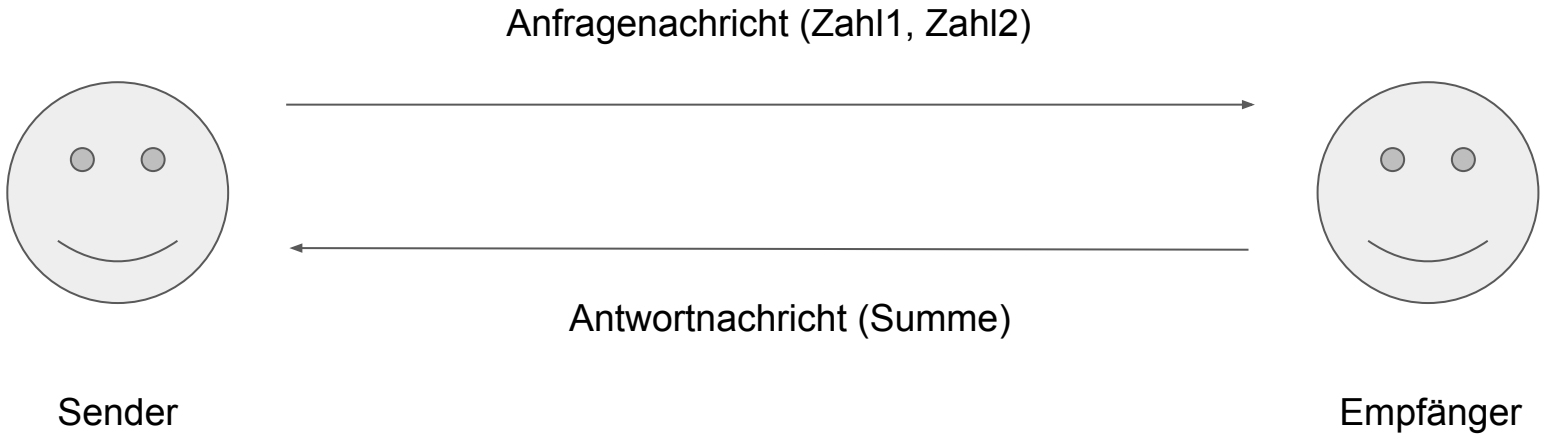
SoapUI // REST u. SOAP // Tests Massenanfragen



aus smartbear.de

sobek innovations

SOAP - WSDL - Aufbau



APIs - SOAP (Simple Object Access Protocol)

Abbildung von Objektorientierten APIs (Daten und Funktionen)

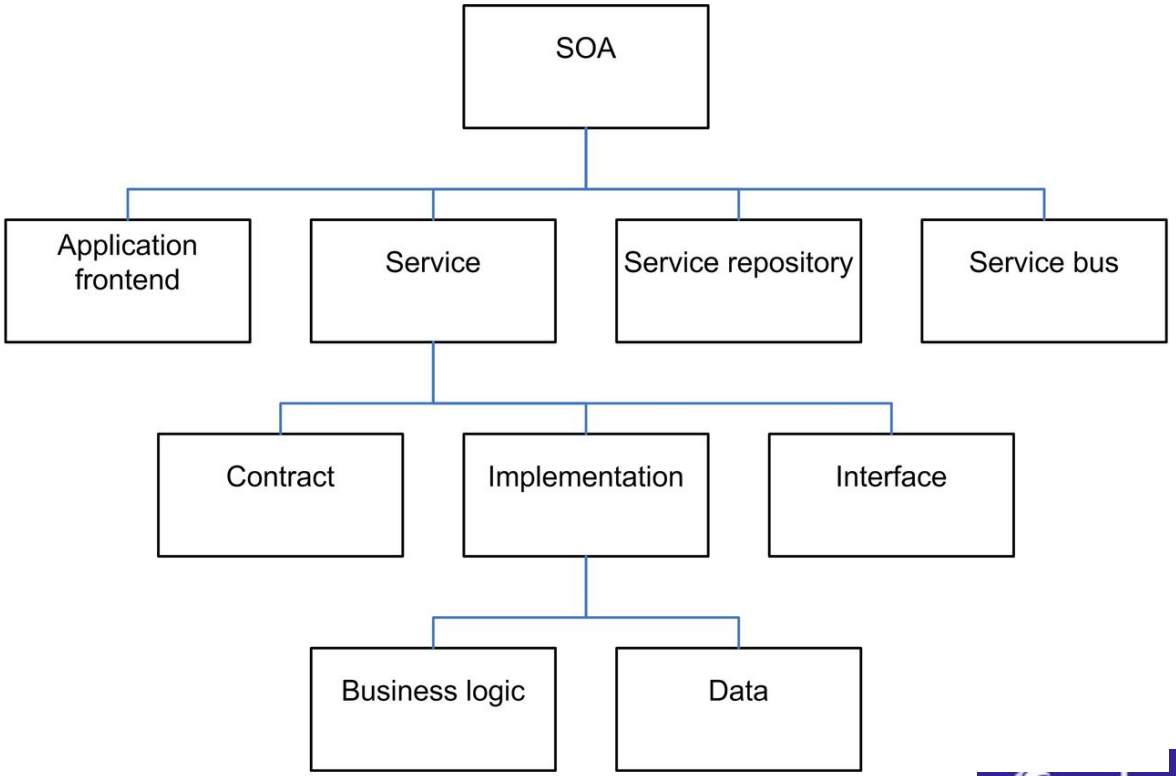
“SOAP (ursprünglich für Simple Object Access Protocol) ist ein Netzwerkprotokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und **Remote Procedure Calls** durchgeführt werden können. SOAP ist ein industrieller Standard des World Wide Web Consortiums (W3C).” aus Wikipedia

APIs - SOAP (Simple Object Access Protocol)

Abbildung von Objektorientierten APIs (Daten und Funktionen)

“SOAP stützt sich auf **XML** zur Repräsentation der Daten und auf Internet-Protokolle der Transport- und Anwendungsschicht (vgl. TCP/IP-Referenzmodell) zur Übertragung der Nachrichten. **Die gängigste Kombination ist SOAP über HTTP und TCP**. SOAP kann beispielsweise auch über SMTP[1] oder JMS[2] verwendet werden. Die mit der Nachricht übermittelten Nutzdaten müssen nicht zwingend in XML gesendet werden, andere Formate wie Base64 oder CSV sind auch möglich” aus Wikipedia

APIs - Service Oriented Architectures



SOAP - WSDL - Aufbau

```
<definitions>
      <!-- Aufzählung verwendete Datentypen -->
      <types>
      </types>
      <!-- Beschreibung der Nachricht bzgl. Anfrage -->
      <message name="NachrichtAnfrage">
            <!-- Parameterbeschreibung der Nachricht -->
      </message>
      <!-- Beschreibung der Nachricht bzgl. Antwort -->
      <message name="NachrichtAntwort">
            <!-- Parameterbeschreibung der Nachricht -->
      </message>
      <portType name="NachrichtAustauschPort">
            <operation name="NachrichtAustausch">
                  <input message="NachrichtAnfrage" />
                <output message="Antwort" />
            </operation>
      </portType>
      <binding name="NachrichtAustauschSoap"> ... </binding>
      <service name="NachrichtService"> ... </service>
</definitions>
```

SOAP - WSDL - Types

<types>-Element

Enthält definierte Datentypen, die im <message>-Element verwendet werden.

Bsp:

```
<types>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.com/buch/suche.xsd"
xmlns="http://www.example.com/buch/suche.wsdl">
            <xs:complexType name="NachrichtAnfrageTyp">
                  <xs:sequence>
                        <xs:element name="Zahl1" type="xs:number"/>
                        <xs:element name="Zahl2" type="xs:number"/>
                  </xs:sequence>
            </xs:complexType>
            <xs:complexType name="NachrichtAntwortTyp">
                  <xs:sequence>
                        <xs:element name="Summe" type="xs:number"/>
                  </xs:sequence>
            </xs:complexType>
...
```

SOAP - WSDL - Types

```
...
            <xs:complexType name="AnfrageTyp">
                  <xs:sequence>
                        <xs:element name="Zahlen" type="NachrichtAnfrageTyp"/>
                  </xs:sequence>
            </xs:complexType>
            <xs:complexType name="AntwortTyp">
                  <xs:sequence>
                        <xs:element name="Summe" type="NachrichtAntwortTyp"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:schema>
</types>
```

REST Beispiel mit JAXRS

Restimplemetierende Klasse - Mapping Annotationen

@Path(/invoices) => Klassenannotation

@GET, @POST, @PUT, @DELETE => Operationsannotations

@Produces(MediaType.APPLICATION_JSON) => Erzeugter Content-Type

@Consumes(MediaType.APPLICATION_JSON) => Akzeptiertes Body-Format

@Path("/{id}") => Pfadabbildung an Methode /invoices/123

Klassen // Model

```
public class Invoice {
    private String product;
    private Double price;

    public Invoice(String product, Double price) {
        this.product = product;
        this.price = price;
    }
    // TODO setter getter
}
```

Klassen // REST-Serviceklasse

```
@Path("/invoices")
public class InvoiceService {

   Map<Long, Invoice> invoices = new HashMap<Long, Invoice>();

   public InvoiceService() {
       Invoice testInvoice =
          new Invoice("Viessmann Heizkessel", 4000.0);
       invoices.put(1l, testInvoice);
   }
   //TODO HTTP Operation Method
}
```

Klassen // REST-Serviceklasse // GET-Methoden

```
@GET
    @Path("/") // /invoices
    @Produces(MediaType.APPLICATION_JSON)
    public Response getInvoices() {
        return Response.status(200).entity(this.invoices.values()).build();
    }

    @GET
    @Path("/{id}") // /invoices/123
    @Produces(MediaType.APPLICATION_JSON)
    public Response getInvoice(@PathParam("id") Long id) {
        return Response.status(200).entity(this.invoices.get(id)).build();
    }
```

Klassen // REST-Serviceklasse // GET-Methoden

```
@GET
    @Path("/") // /invoices
    @Produces(MediaType.APPLICATION_JSON)
    public Response getInvoices() {
        return Response.status(200).entity(this.invoices.values()).build();
    }

    @GET
    @Path("/{id}") // /invoices/123
    @Produces(MediaType.APPLICATION_JSON)
    public Response getInvoice(@PathParam("id") Long id) {
        return Response.status(200).entity(this.invoices.get(id)).build();
    }
```

Klassen // Serviceklasse // Create/Update-Methoden

```
@POST
    @Path("/") // /invoices
    @Produces(MediaType.APPLICATION_JSON)
    public Response createInvoice(Invoice invoice) {
        this.invoices.put((long) (Math.random() * 10000), invoice);
        return Response.status(201).build();
    }

    @PUT
    @Path("/{id}") // /invoices/123
    @Produces(MediaType.APPLICATION_JSON)
    public Response updateInvoice(@PathParam("id") Long id, Invoice invoice) {
        this.invoices.put(id, invoice);
        return Response.status(201).build();
    }
```

Servlet Dependency

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
```

Servlet Beispiel

```
public class ServletExample extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse
response)

      throws ServletException, IOException {

      message = "Hallo Consumer!!!";

      response.setContentType("text/html");

      PrintWriter out = response.getWriter();

      out.println("<h3>" + message + "</h3>");

   }

}
```

Entwicklungdeployment

```
mvn clean install jetty:run
```

Transaktionen mit REST

Transaktion

Als **Transaktion** bezeichnet man in der Informatik **eine Folge von Programmschritten**, die als eine **logische Einheit** betrachtet werden, weil sie den Datenbestand nach fehlerfreier und vollständiger Ausführung in einem konsistenten Zustand hinterlassen.

aus Wikipedia

REST Transaktionwraper I

1. **Neue Transaktion anfordern (Warenkorb)**

   POST /api/transactions

   RequestBody: {}

   ReponseBody: {txn: 321, state: ‘pending’, steps: []}

2. **Änderung anfordern (Warenkorb hinzufügen)**

   POST /api/transactions/321

   RequestBody: {articleId: 123, amount: 2, type: ‘add’} =>

   ResponseBody: {txn: 321, state: ‘pending’, [{txnp: 1, articleId: 123, amount: 2, type: ‘add’}]

REST Transaktionwrapper II

3. **Änderung anfordern (Warenkorb hinzufügen)**

POST /api/transactions/321

RequestBody: {articleId: 456, amount: 1, type: ‘add’} =>

ResponseBody: {txn: 321, state: ‘pending’, [
{txnp: 1, articleId: 123, amount: 2, type: ‘add’},
{txnp: 2, articleId: 456, amount: 1, type: ‘add’}
]}

]}

REST Transaktionwrapper III

4. **Transaktion abschließen (Bestellung absenden)**

PUT /api/transactions/321

RequestBody: {state: ‘commit’’}
ResponseBody: {txn: 321, state: commited’, [
{txnp: 1, articleId: 123, amount: 2, type: ‘add’},
{txnp: 2, articleId: 456, amount: 1, type: ‘add’}
]}

REST Beispiel mit Spring REST

RestController - Mapping Annotationen bzgl. HTTP-Methode

@PostMapping => führt ein CREATE durch (hier Body)

@GetMapping => führt ein GET durch

@PutMapping => führt ein UPDATE durch (hier Body)

@DeleteMapping => führt ein DELETE durch

@RequestBody => Deklariert bei Parameterübergabe das HTTP-Body-Objekt

Maven Projekt erstellen

```
mvn archetype:generate -DgroupId=de.sobek.innovations
-DartifactId=training.app.rest.springws
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

Danach in der IDE öffnen.

Dependencies konfigurieren // Springboot Parent

```
<parent>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-parent</artifactId>
	<version>2.0.5.RELEASE</version>
</parent>
```

Dependencies konfigurieren // Repositories

```
<repositories>
  <repository>
    <id>spring-releases</id>
    <url>https://repo.spring.io/libs-release</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>spring-releases</id>
    <url>https://repo.spring.io/libs-release</url>
  </pluginRepository>
</pluginRepositories>
```

Dependencies konfigurieren // Build Task

```
<repositories>
  <repository>
    <id>spring-releases</id>
    <url>https://repo.spring.io/libs-release</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>spring-releases</id>
    <url>https://repo.spring.io/libs-release</url>
  </pluginRepository>
</pluginRepositories>
```

Dependencies konfigurieren // Repositories

```
<properties>
  <java.version>1.8</java.version>
</properties>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Dependencies konfigurieren // 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>
<dependency>
     <groupId>com.jayway.jsonpath</groupId>
     <artifactId>json-path</artifactId>
     <scope>test</scope>
</dependency>
```

Klassen // Model // Resourcenklasse

```
public class Customer {

    private long id;
    private String firstName;
    private String secondName;

    public Customer(long id, String firstName, String secondName) {
        this.id = id;
        this.firstName = firstName;
        this.secondName = secondName;
    }

    //TODO setter and getter
}
```

Dependencies konfigurieren // RestController

```
@RestController
public class CustomerController {

   private Map<Long, Customer> customers = new HashMap<Long,Customer>();

   //NOT IMPLEMENTED NOW
}
```

RestController - HTTP-Status Mapping über eigene Exception

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason="customer not found")
   public class NotFoundException extends RuntimeException {}
```

CustomerKlasse // Kunden auslesen // Get

```
@GetMapping("/customers")
    public Collection<Customer> getAllCustomers() {
        return customers.values();
    }

@GetMapping("/customers/{customerId}")
    public Customer get(@PathVariable Long customerId) {
        if (!customers.containsKey(customerId)) {
            throw new NotFoundException();
        }
        return customers.get(customerId);
    }
```

CustomerKlasse // Kunden anlegen

```
@PostMapping("/customers")
    public void createCustomer(@RequestBody Customer customer) {
        long createdId = Math.round(Math.random() * 100000000);
        customer.setId(createdId);
        this.customers.put(createdId, customer);
    }
```

CustomerKlasse // Daten des Kunden aktualisieren

```
@PutMapping("/customers/{customerId}")
   public void updateCustomer(Long customerId, @RequestBody Customer customer) {
       if (!customers.containsKey(customerId)) {
           throw new NotFoundException();
       }
       this.customers.put(customerId, customer);
   }
```

CustomerKlasse // Kunden löschen

```
@DeleteMapping("/customers/{customerId}")
    public void deleteCustomer(Long customerId) {
        if (!customers.containsKey(customerId)) {
            throw new NotFoundException();
        }
        this.customers.remove(customerId);
    }
```

Klassen // Main-Klasse // Springboot

```
@SpringBootApplication
public class App {

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

}
```

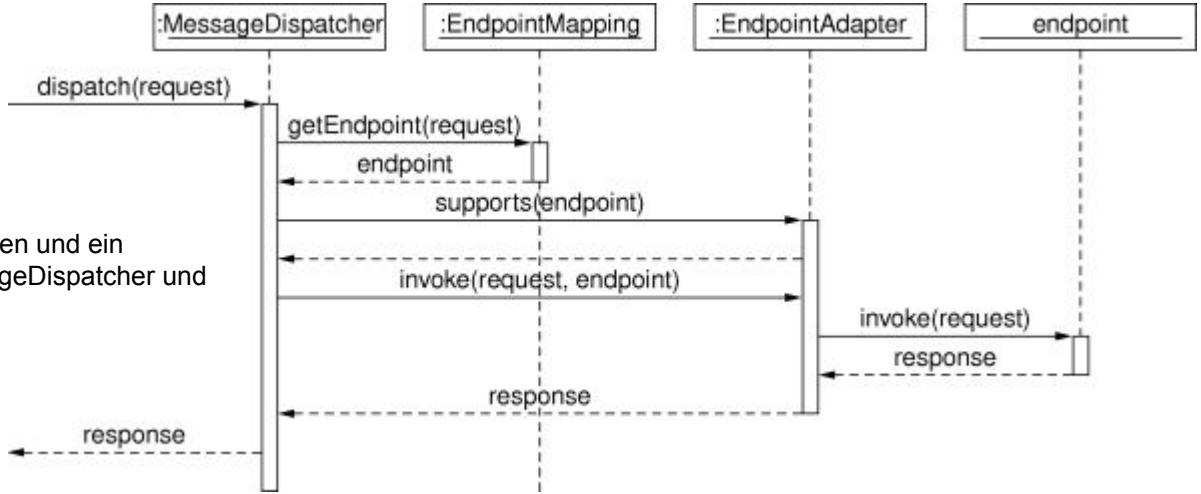
Klassen // Starten // Springboot

```
mvn clean spring-boot:run
```

SOAP Beispiel mit Spring Webservices

MessageDispatcher

1. Die MessageDispatcher-Klasse initiiert die Request-Bearbeitung.
2. Per getEndpointMapping wird in einer intern aufgebauten Registry geschaut, ob für den eingegangen Request ein Endpoint registriert wurde.
3. Falls ein Endpoint Adapter wird dieser aufgerufen und ein Response-Objekt erzeugt sowie an den MessageDispatcher und dieser an den Aufrufer zurückgegeben.



Maven Projekt erstellen

```
mvn archetype:generate -DgroupId=de.sobek.innovations
-DartifactId=training.app.rest.springws
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false

Download: ...
```

Danach in der IDE öffnen.

XJC - Plugin // Wandelt XML Schema in JAXBObjekte

```
<plugin>
   <groupId>org.codehaus.mojo </groupId>
   <artifactId>jaxb2-maven-plugin </artifactId>
   <version>1.6</version>
   <executions>
       <execution>
           <id>xjc</id>
           <goals>
               <goal>xjc</goal>
           </goals>
       </execution>
   </executions>
   <configuration>
       <schemaDirectory>${project.basedir}/src/main/resources </schemaDirectory>
       <outputDirectory>${project.basedir}/src/main/java </outputDirectory>
       <clearOutputDir>false</clearOutputDir>
       <packageName>de.sobekinnovations.invoices.contract </packageName>
   </configuration>
</plugin>
```

XML Schema erstellen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://sobekinnovations.de/invoices"
   xmlns:tns="http://sobekinnovations.de/invoices" elementFormDefault="qualified">
   <xs:element name="GetInvoiceRequest">
   </xs:element>
   <xs:element name="GetInvoiceResponse">
       <xs:complexType>
           <xs:sequence>
               <xs:element name= "Invoice" type="tns:Invoice" />
           </xs:sequence>
       </xs:complexType>
   </xs:element>
   <xs:complexType name="Invoice">
       <xs:sequence>
           <xs:element name="id" type="xs:long" />
           <xs:element name="description" type="xs:string" />
           <xs:element name="price" type="xs:float" />
       </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Endpoint erstellen

```
@Endpoint
public class InvoicesEndpoint {
   @PayloadRoot(namespace = "http://sobekinnovations/invoices", localPart = "GetInvoiceRequest")
   @ResponsePayload
   public GetInvoiceResponse processCourseDetailsRequest(@RequestPayload GetInvoiceRequest request) {
       GetInvoiceResponse response = new GetInvoiceResponse();
       Invoice invoice = new Invoice();
       invoice.setId(Math.round(Math.random() * 100000));
       invoice.setDescription("Viessmann Heizkessel");
       invoice.setPrice(5000);

       response.setInvoice(invoice);

       return response;
   }
}
```

sobek innovations

Message Dispatcher // Konfiguration

```
@EnableWs
@Configuration
public class WebServiceConfig {

   @Bean
   public ServletRegistrationBean messageDispatcherServlet(ApplicationContext context) {
       MessageDispatcherServlet messageDispatcherServlet = new MessageDispatcherServlet();
       messageDispatcherServlet.setApplicationContext(context);
       messageDispatcherServlet.setTransformWsdlLocations(true);
       return new ServletRegistrationBean(messageDispatcherServlet, "/ws/*");
   }

}
```

Message Dispatcher // WSDL

```
@EnableWs
@Configuration
public class WebServiceConfig {

   ...

   @Bean(name = "invoices")
   public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema invoicesSchema) {
       DefaultWsdl11Definition definition = new DefaultWsdl11Definition();
       definition.setPortTypeName("InvoicePort");
       definition.setTargetNamespace("http://sobekinnovations.de/invoices");
       definition.setLocationUri("/ws");
       definition.setSchema(invoicesSchema);
       return definition;
   }

   @Bean
   public XsdSchema invoicesSchema() {
       return new SimpleXsdSchema(new ClassPathResource("invoices.xsd"));
   }
}
```

Spring Boot Main Class

```
@SpringBootApplication
public class App {

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

SSL

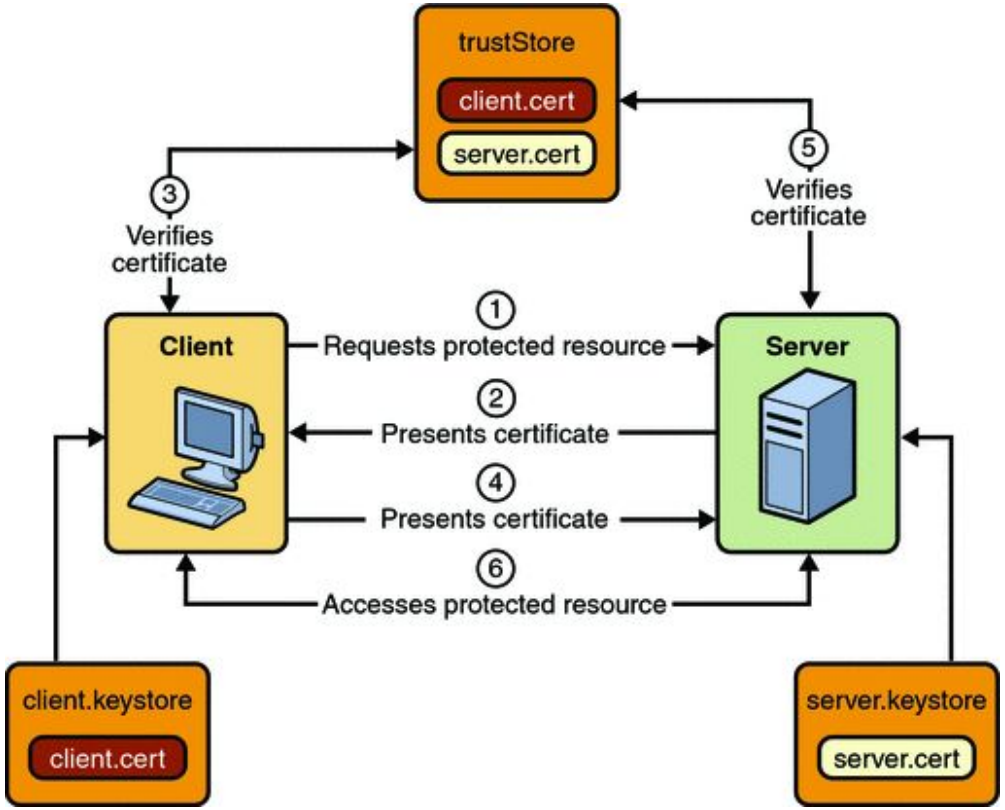
Client & Server-Zertifikate

SSL - Zertifikate

Ein SSL-Zertifikat ist ein Public-Key-Zertifikat. Im asymmetrischen Verschlüsselungssystemen wird mit einem Public-Key-Zertifikat die **Eigentümerschaft des öffentlichen Schlüssels bestätigt**. Desweiteren enthalten Zertikate auch Authorisierungsbereiche.

Vertraulichkeit	Authentizität	Integrität

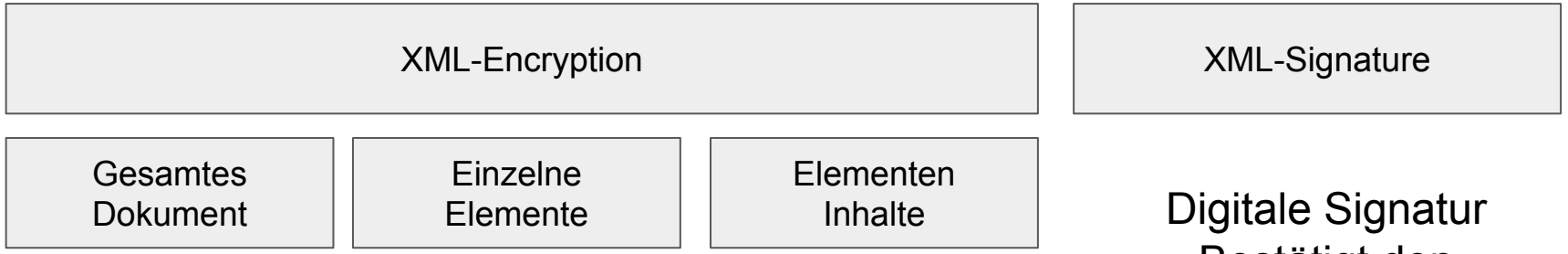
SSL - Client und serverseitige Zertifizierung



WS-Security

WS-Security - Überblick

WS Security setzt auf zwei bestehende Standards. Die XML-Encryption und -SIgnature.



Welche XML-Teile bei Webservices SOAP-Elemente möchte ich verschlüsseln.

SAML “2”

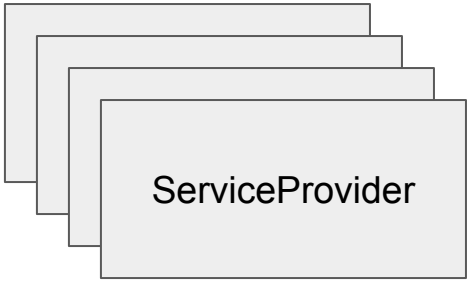
SAML - Überblick

Security Framework (XML) zur Sicherstellung folgender Aufgaben:

Single Sign On

Authorisierungsdienste

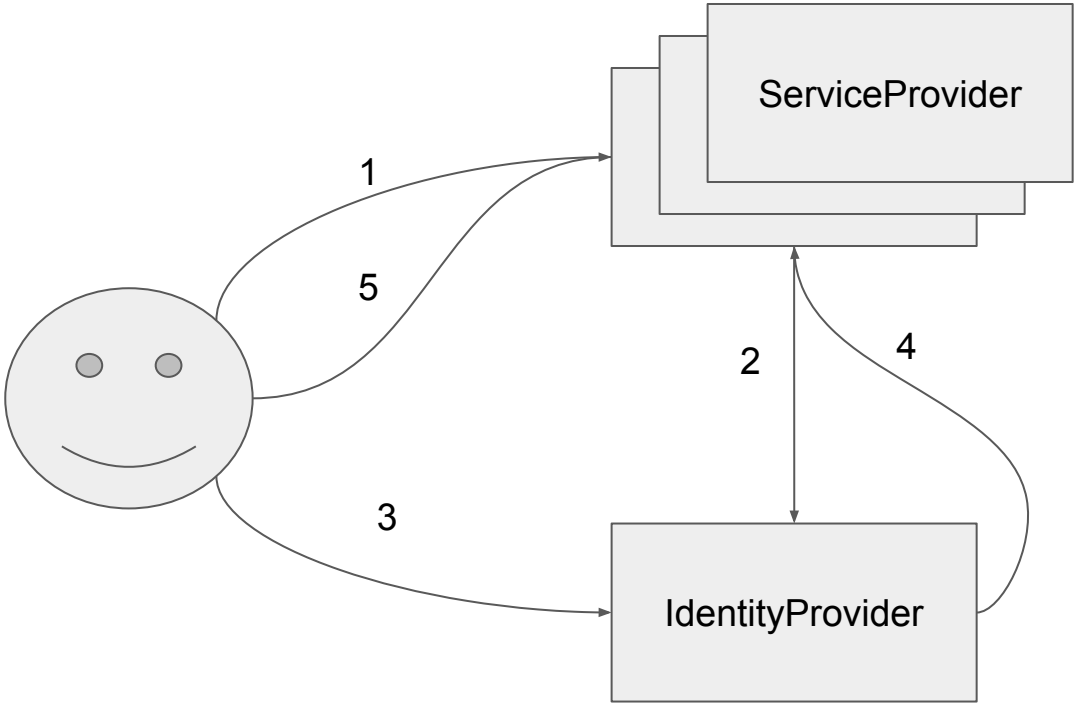
SAML unterscheidet zwischen dem Service Provider (z.B. Webapplikation) und Identity Provicer.:



IdentityProvider

Single Sign On

Ein Anwender ist nach der Anmeldung in einer Webanwendung automatisch auch in anderen Webanwendungen authentifiziert.



1. Anwender greift auf den Service Provider zu
2. ServiceProvider initiert eine Authentifizierunganfrage an den IdentityProvider. Es erfolgt eine Weiterleitung auf die Webseite des IdentityProviders.
3. Der Anwender befindet sich in einer Loginmaske des IdentityProviders und meldet sich mit seinen Credentials hier an.
4. Im Hintergrund wird eine Antwort mit einer sg. Assertion an den ServiceProvider vom IdentityProvider versendet.
5. Nun landet eingeloggt der Nutzer auf der authentifizierten Seite des ServiceProviders
6. Alternativ kann sich direkt beim IdentityProvider einloggen und ist auch automatische in anderen SPs eingeloggt.

Single-Point of Failure“

sobek innovations

Identity Provider - Anbieter

JAXWS

APIs - SOAP Beispiel

Einfacher Taschenrechner // Zuerst kommt das Interface

```java
package de.soap;

import javax.jws.*;

/** Dienst-Interface */
@WebService
public interface ICalculator {
   Integer add(    @WebParam( name = "zahl1" ) Integer zahl1, @WebParam( name = "zahl2" ) Integer zahl2) throws Exception;
}
```

APIs - SOAP Beispiel

Einfacher Taschenrechner // Dann die Implementierung des Interfaces

```java
package de.soap;

import javax.jws.WebService;

/** Dienstimplementierung */
@WebService( endpointInterface="de.soap.ICalculator" )
public class CalculatorImpl implements ICalculator
{
   @Override public Integer add( Integer zahl1, Integer zahl2 ) {
      return zahl1 + zahl2;
   }
}
```

APIs - SOAP Beispiel

Einfacher Taschenrechner // Server hochfahren

```java
/** Testserver fr den Webservice */
public class TestWsServer
{
   public static void main( final String[] args )
   {
      String   url = ( args.length > 0 ) ? args[0] : "http://localhost:4434/calculator";
      Endpoint ep = Endpoint.publish( url, new CalculatorImpl() );
   }
}
```

APIs - SOAP Beispiel

Einfacher Taschenrechner // Client mit Objektorientierten Interface

```java
public class TestWsClient {
      public static void main(final String[] args) throws Exception {
            String url = (args.length > 0) ? args[0] : "http://localhost:4434/calculator";
            // test( "TestWsClient", url, 1, 2, false );
            Service service = null;
         int timeoutSekunden = 20;
         while( service == null ) {
            try {
               service = Service.create(
                     new URL( url + "?wsdl" ),
                     new QName( "http://soap.de/", "CalculatorImplService" ) );
            } catch( WebServiceException ex ) {
               if( timeoutSekunden-- <= 0 ) throw ex;
               try { Thread.sleep( 1000 ); } catch( InterruptedException e ) {}
            }
         }
         ICalculator calculator = service.getPort( ICalculator.class );
         System.out.println(calculator.add(1, 2));
      }

}
```

APIs - SOAP Beispiel

Einfacher Taschenrechner // Server hochfahren

```java
/** Testserver fr den Webservice */
public class TestWsServer
{
   public static void main( final String[] args )
   {
      String   url = ( args.length > 0 ) ? args[0] : "http://localhost:4434/calculator";
      Endpoint ep = Endpoint.publish( url, new CalculatorImpl() );
   }
}
```

Beispiele

APIs - Beispiel

Abbildung von Rechnungen (Pluralität)

Alle Rechnungen auslesen	GET /invoices 200
Alle Rechnungen eines Kunden auslesen	GET /invoices?customerId=1
Eine Rechnungen auslesen	GET /invoices/2

APIs - Beispiel

Abbildung von Rechnungen (Pluralität)

Rechnung erstellen	POST /invoices 200 Anmerkung: Rückgabe erstellte Ressource oder Id der erstellten Ressource
Rechnung updaten	PUT /invoices/2 200 Anmerkung: Rückgabe aktualisierte Ressource oder nichts
“Rechnung löschen”	DELETE /invoices/2 200