

# REST-APIs

Grundlagen

Ralf Ueberfuhr, Trainer & Consultant

## Ralf Ueberfuhr

Trainer & Consultant

[ralf.ueberfuhr@ars.de](mailto:ralf.ueberfuhr@ars.de)

+49 89 32468-2050

ARS Computer und Consulting GmbH

[www.ars.de](http://www.ars.de)



## Was ist REST?

- Architekturstil:

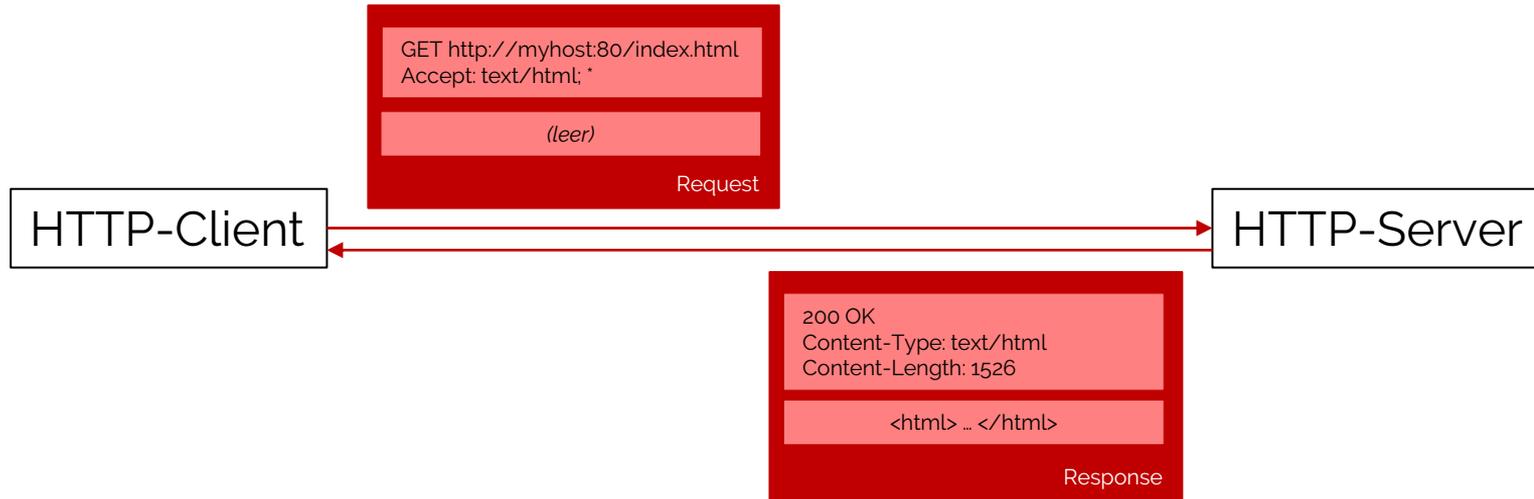
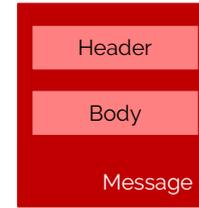
*„Nutze HTTP so, wie es beim Design gedacht war.“*



[https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

## Voraussetzung: HTTP

- Kommunikationsprotokoll im WWW
  - Text-basiert
  - Zustandslos
  - Request-Response-Modell → 2 Arten von Nachrichten



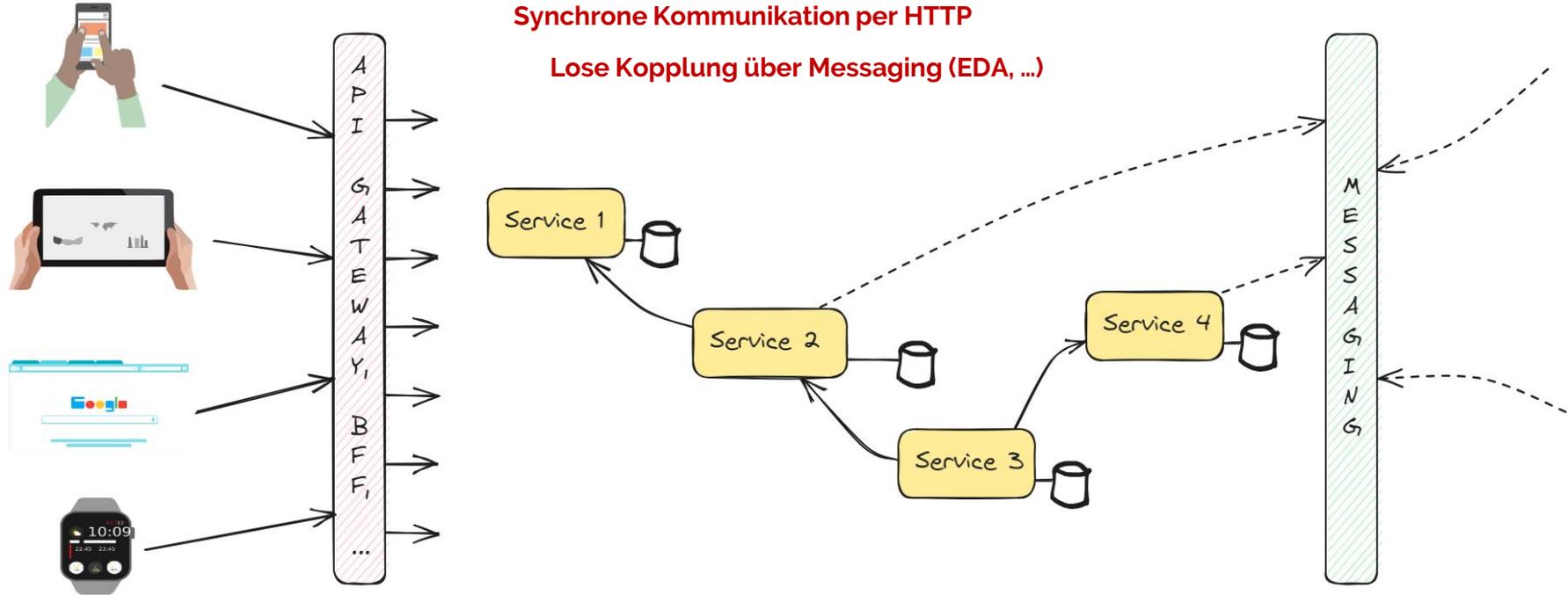
# Motivation: Verteilte Anwendungen (Microservices)

Schneiden nach Domäne (DDD)

Zustandslos

Synchrone Kommunikation per HTTP

Lose Kopplung über Messaging (EDA, ...)



Vorteile	Herausforderungen
Verteilte Entwicklung in kleineren Teams (Agilität)	Vervielfachung (ALM, ...) → Automatisierung!
Geringere Komplexität des einzelnen Service (Abhängigkeiten, Code, Fachlichkeit, CI/CD ...)	Größere Komplexität der Infrastruktur (Testen, Debugging, Überblick ...)
Abhängigkeiten werden in Infrastruktur „sichtbar“	Schnittstellen: Definition, Kompatibilität, ...
Technologieunabhängigkeit	Redundanzen von Daten (Hoheit?) und Code
Fehlertoleranz	Resilience Patterns
Skalierbarkeit	Verteilte Transaktionen, Security

*Herausforderungen verschieben sich aus der Anwendung in die Infrastruktur. (deklarativ)*

*Es gilt das CAP-Theorem. Es gibt auch Mischlösungen. Betrieb in der Cloud ist empfehlenswert.*

## Regel 1: Adressierung von Ressourcen

- Substantive
  - Mehrzahl (List Resources)
  - Einzahl (Single Resources)
- Verschachtelung möglich (Sub Resources)

```
# List Resource
```

```
http://api.foo.de/api/v1/customers
```

```
# Single Resource
```

```
http://api.foo.de/api/v1/customers/5
```

```
# Sub Resource
```

```
http://api.foo.de/api/v1/customers/5/address
```

## Regel 2: Verwendung der HTTP-Methoden

- Abbildung von CRUD-Operationen

Operation	HTTP-Methode
Create	POST vs. PUT
Read	GET
Update	PUT vs. PATCH
Delete	DELETE

### # Create

```
POST http://api.foo.de/api/v1/customers
```

### # Read

```
GET http://api.foo.de/api/v1/customers
```

```
GET http://api.foo.de/api/v1/customers/5
```

### # Update

```
PUT http://api.foo.de/api/v1/customers/5
```

### # Delete

```
DELETE http://api.foo.de/api/v1/customers/5
```

## Regel 2: Ändern mit PUT vs. PATCH

Kriterium	PUT	PATCH
Semantik	Überschreiben	Ändern
Übertragung	Kompletter Datensatz (Ausnahme: ID)	Zu ändernde Felder
Programmierung	Einfach (kompletter Ersatz)	Komplexer (Merging)

**PUT /customers/5**

```
{
  "name": "Schmidt"
}
```



**Customer ID = 5**

- Name = "Schmidt"
- Vorname = null

**Customer ID = 5**

- Name = "Maier"
- Vorname = "Tom"



**PATCH /customers/5**

```
{
  "name": "Schmidt"
}
```



**Customer ID = 5**

- Name = "Schmidt"
- Vorname = Tom

## Regel 2: Anlegen mit PUT vs. POST

- Idempotenz!
  - POST: Erzeugen der ID am Server
  - PUT: Erzeugen der ID am Client (Mitsenden)

HTTP-Methode	Idempotenz
GET	✓
PUT	✓
POST	?
PATCH	?
DELETE	✓

# Delete ✓

DELETE <http://api.foo.de/api/v1/customers/5>

# Delete ✗

DELETE <http://api.foo.de/api/v1/customers/oldest>

# Anlegen+Überschreiben mit PUT ✓

PUT <http://api.foo.de/api/v1/customers/john@acme.com>

PUT <http://api.foo.de/api/v1/customers/5/address>

# Anlegen mit POST ✓

POST <http://api.foo.de/api/v1/customers>

## Regel 2: Anlegen mit PUT vs. POST

- Semantik!
  - PUT: Ersetze die Ressource durch den mitgeschickten Datensatz. Erfolgt ein GET auf die Ressource, wird zukünftig dieser Datensatz geliefert.
  - POST: Die Ressource soll ihre eigene Logik auf den mitgeschickten Datensatz anwenden. (Listenressourcen: Hinzufügen eines neuen Elements inkl. Generierung der ID)

```
# Anlegen+Überschreiben mit PUT ✓
```

```
PUT http://api.foo.de/api/v1/customers/john@acme.com
```

```
# Anlegen mit POST ✓
```

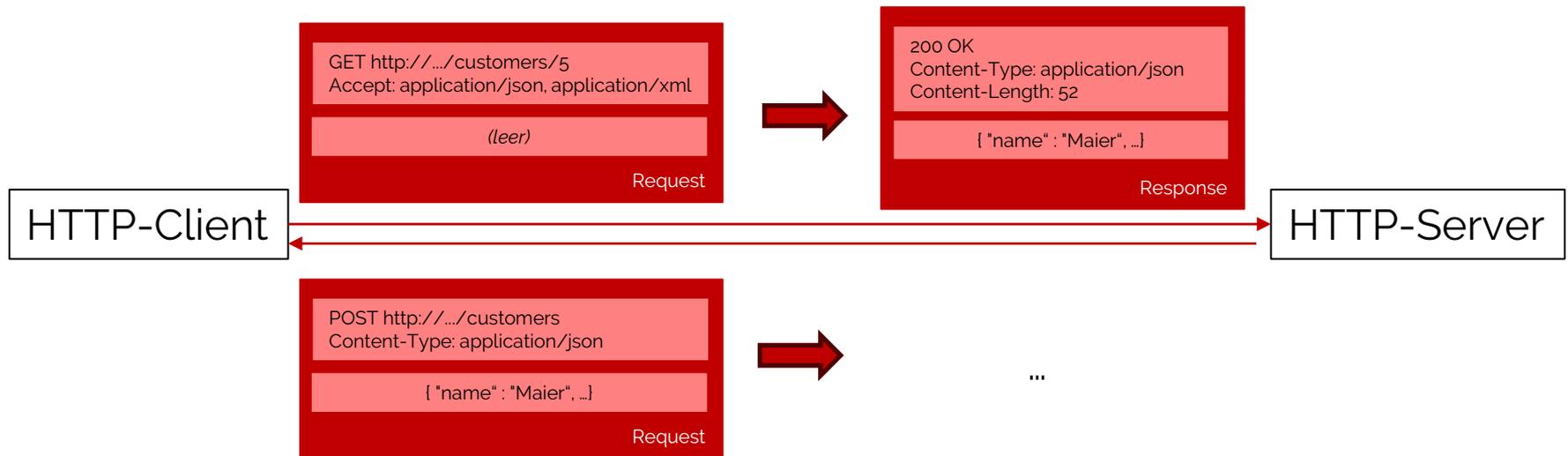
```
POST http://api.foo.de/api/v1/customers
```

```
# Aufruf einer Aktivitätsressource ✓
```

```
POST http://api.foo.de/api/v1/customers/5/notifications
```

## Regel 3: Content Negotiation

- Repräsentation einer Ressource = Format im Body der Nachricht
  - Empfohlen bei REST: JSON
- Aushandeln zwischen Client und Server



## Regel 4: Status Codes

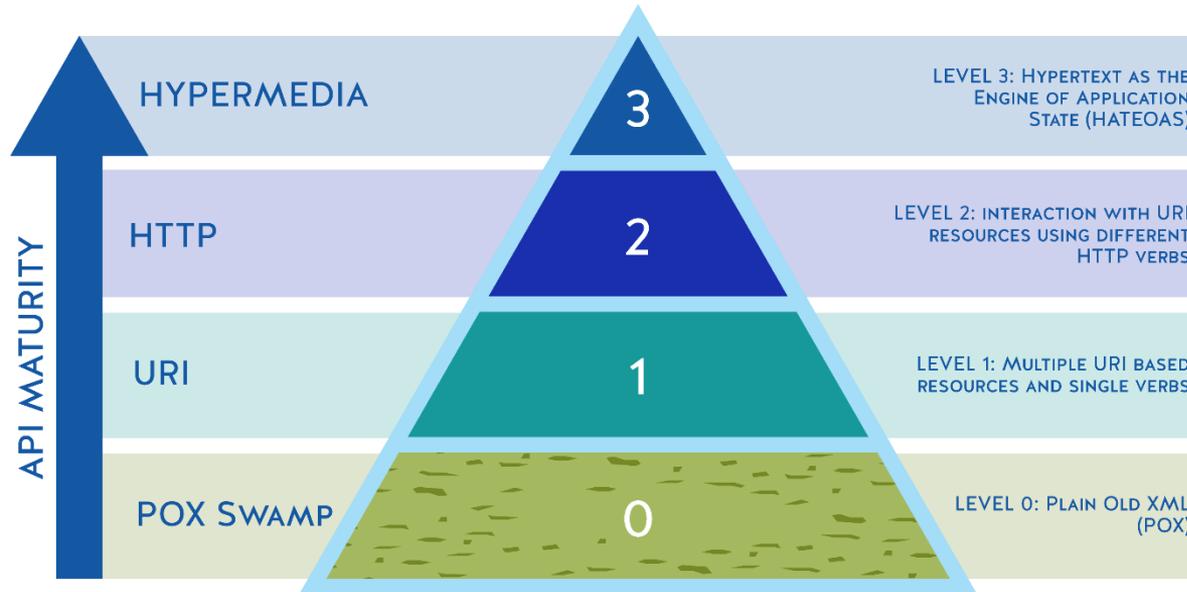
- HTTP definiert Statuscodes für unterschiedliche Fälle
  - Erfolgscodes
  - Daten sind fehlerhaft
  - Content Negotiation nicht erfolgreich (falsches Format)
  - Ressource nicht gefunden
  - Server-interner Fehler
  - ...
- Übersicht
  - <https://de.wikipedia.org/wiki/HTTP-Statuscode> (Überblick)
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status> (detaillierte Beschreibung)
  - <https://http.cat/>  bzw. <https://httpstatusdogs.com/> 

1. [Informational responses](#) ( 100 – 199 )
2. [Successful responses](#) ( 200 – 299 )
3. [Redirection messages](#) ( 300 – 399 )
4. [Client error responses](#) ( 400 – 499 )
5. [Server error responses](#) ( 500 – 599 )

## Regel 5: Zustandslosigkeit

- Microservices / REST APIs sind zustandslos
  - Keine HTTP Sessions
  - In-Memory-Daten lediglich als Redundanz (z.B. Cache)
- Grund: Skalierbarkeit / Ausfallsicherheit / Cloud-ServiceModel (Pets vs. Cattle)
- Informationen, die benötigt werden, um eine Anfrage zu bearbeiten, kommen
  - aus einer Datenbank,
  - von einem anderen REST-Service
  - von einer beliebigen anderen Datenquelle
  - mit der Anfrage: Request-Parameter, Header, Body (**API!**)
  - NICHT aus Daten einer vorherigen Anfrage

## THE RICHARDSON MATURITY MODEL



NORDICAPIS.COM