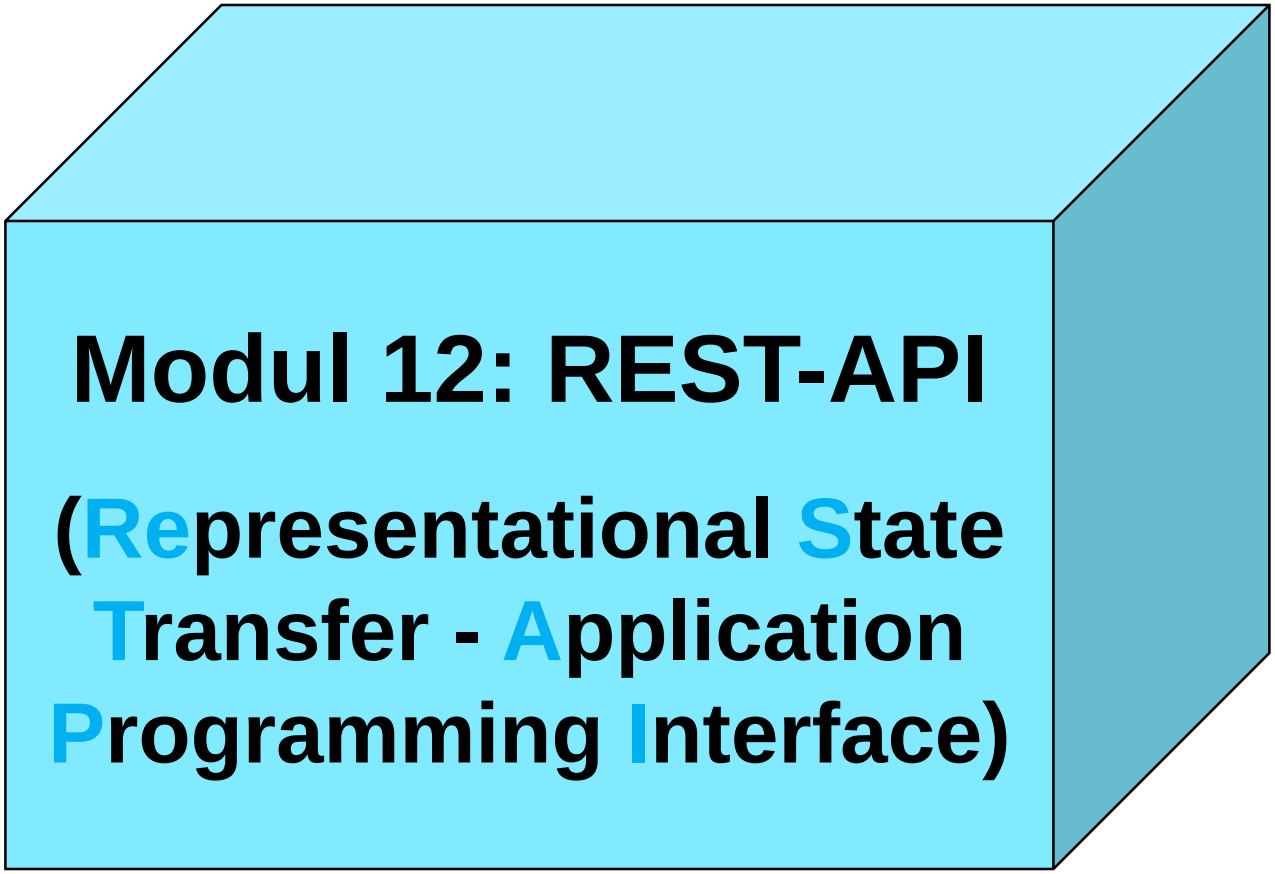
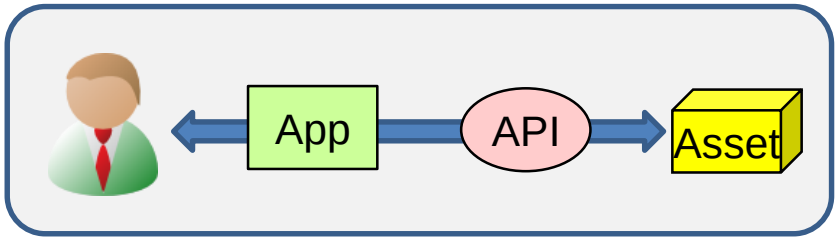

Modul 12: REST-API
(Representational State
Transfer - Application
Programming Interface)

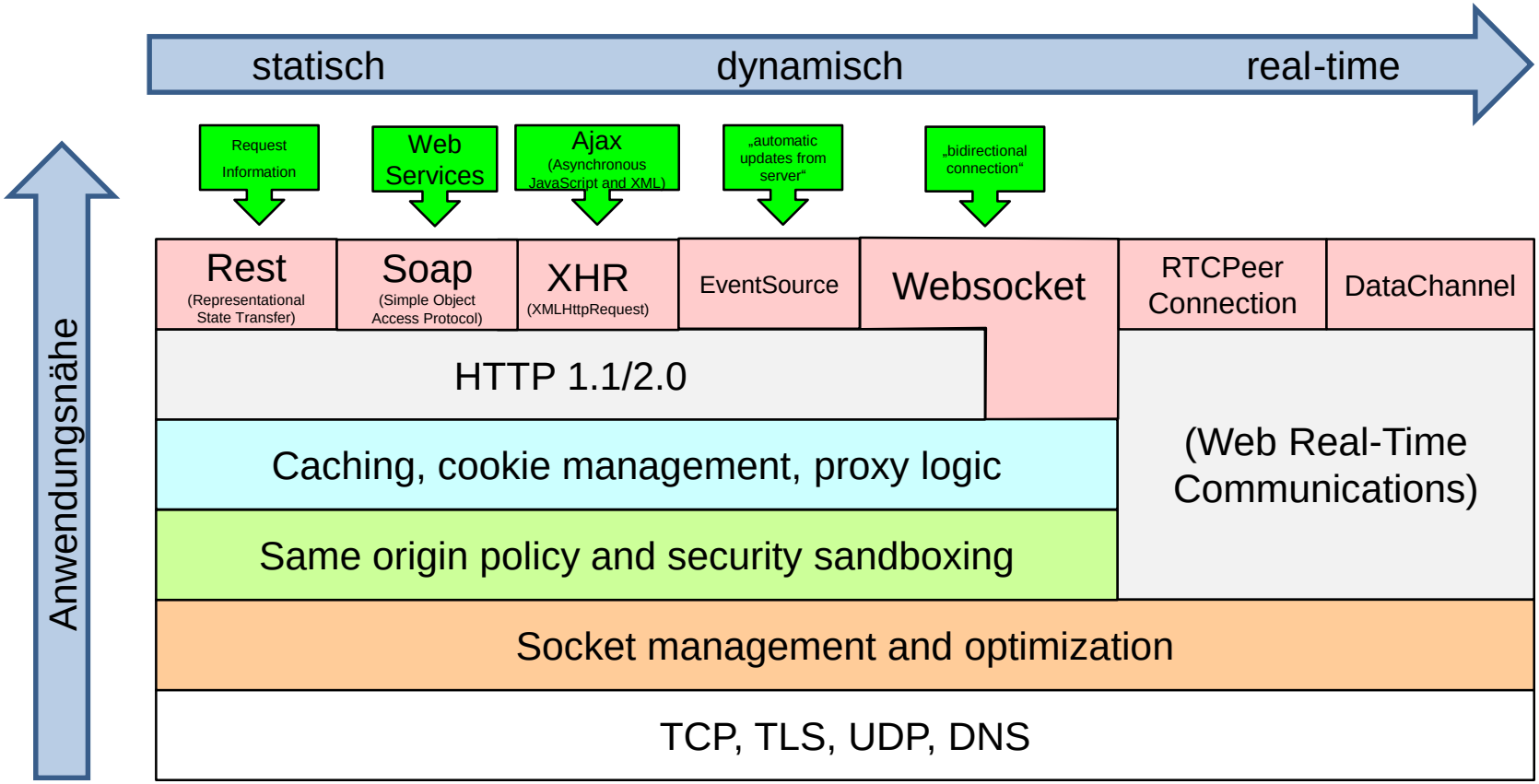
Was sind Web APIs?

- Ziel der Web-API: Bereitstellen von Funktionalität im Internet.
- Eine Web-API wird von einem Programm und nicht von einem Menschen genutzt. Daher muss die Logik der API vorher im Detail spezifiziert sein.
- Informationen über eine Web-API:
  - Aufrufstelle
  - Funktionalität
  - Input/Output-Parameter
  - maschinenlesbare Dokumentation
  - SLAs, technische, kaufmännische, rechtliche Anforderungen



- Typen von APIs:
  - private API
  - Öffentliche API (z.B. Google APIs)

Einordnung der REST-API in den Protokollstack



compare https://hpbn.co/primer-on-browser-networking/ (Primer on Browser Networking)

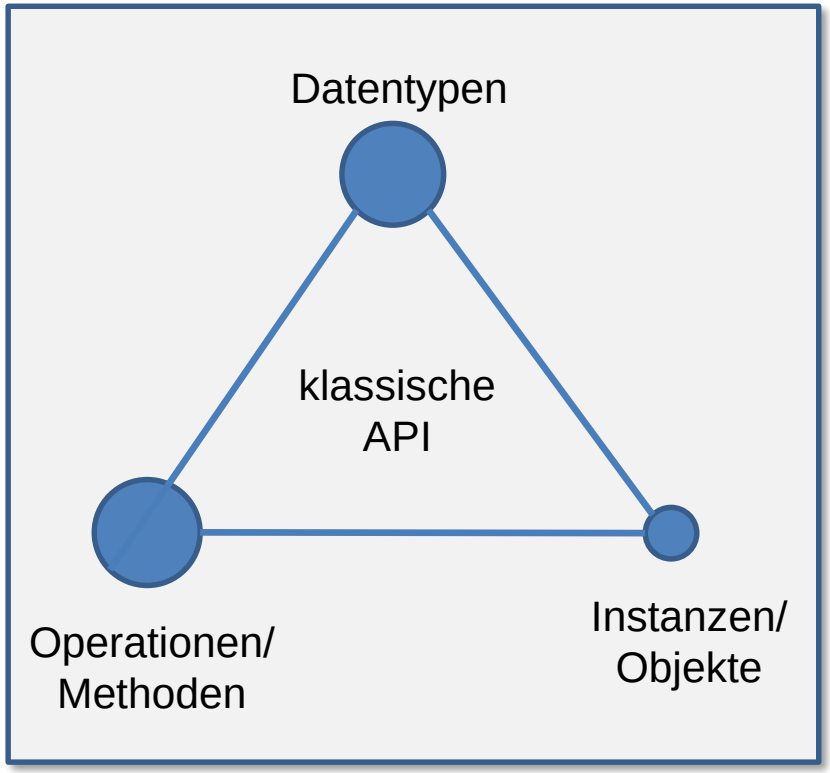
REST

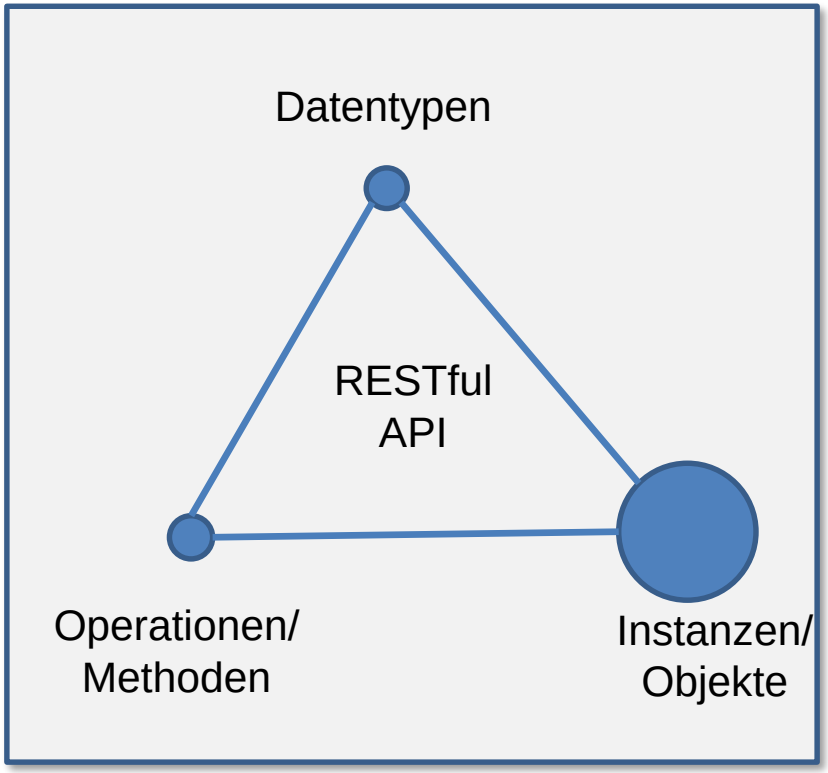
- **Ausgangspunkt:**
  Roy Thomas Fielding: Architectural Styles and the Design of Network-based Software Architectures, dissertation, University of California, Irvine, 2000.
  https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- **„Representational State Transfer“:**
  Die Ressourcen werden nicht direkt in ihrem Zustand manipuliert, sondern über ihr Repräsentationen.
  (In der Thesis von Fielding steht: "manipulation of resources through representations")
- **Grundidee: Architekturmodell für globales Informationssystem basierend auf Protokoll HTTP und auf Ressourcenidentifikation über URI.**

Grundprinzipien von REST

- Schwerpunkt von REST liegt auf der Konzeptbildung.
- Ergebnis ist ein Architekturstil REST, der beschreibt, wie man HTTP + URI für die Realisierung von Web_APIs einsetzt.
- Ergebnis ist eine REST-konformen Implementierung einer HTTP-API.
  → RESTful HTTP.
- REST Grundprinzipien:
  - HTTP Standardmethoden für den Zugriff
  - Statuslose Kommunikation
  - Ressourcen mit eindeutiger Identifikation
  - Unterschiedliche Repräsentationen der Ressourcen
  - Verknüpfungen/Hypermedia
- Bei REST stehen die Daten und nicht die Funktionen im Mittelpunkt. Die Daten sind unstrukturiert und können "Alles" sein.

Optionsdreieck – Operationen, Datentypen, Objekte

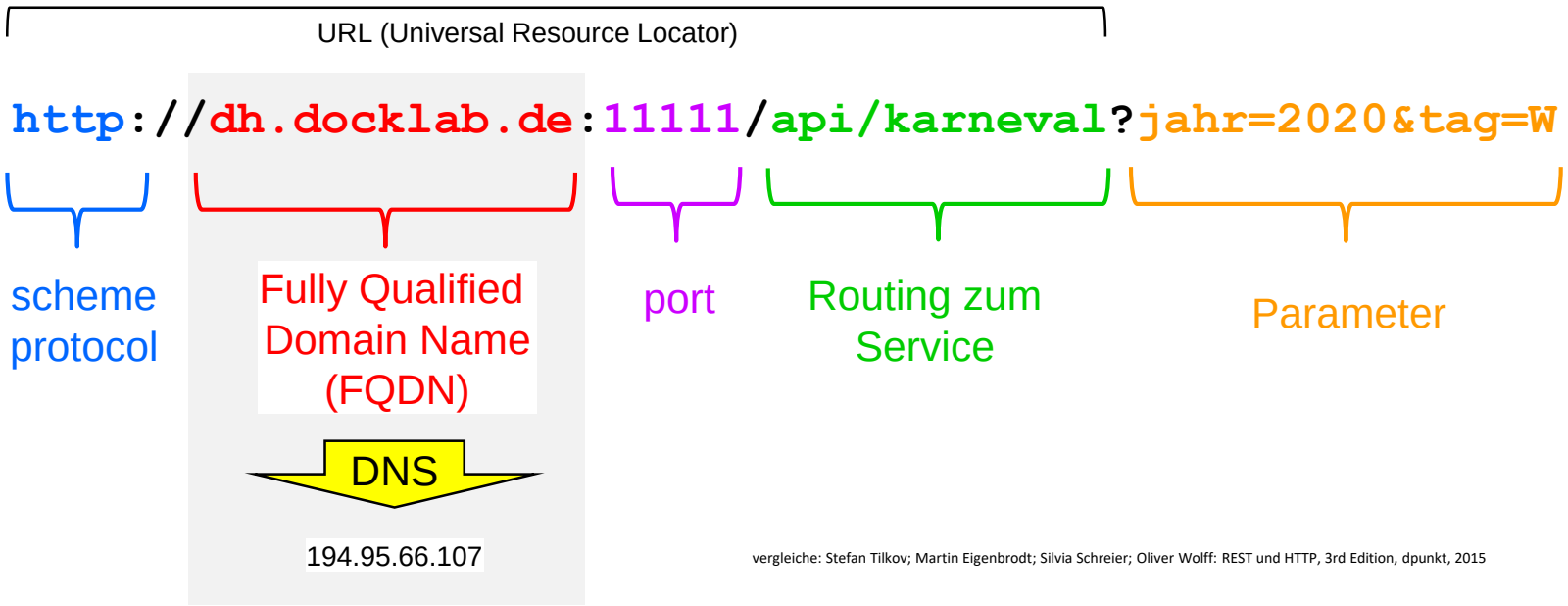




vergleiche: Stefan Tilkov; Martin Eigenbrodt; Silvia Schreier; Oliver Wolff: REST und HTTP, 3rd Edition, dpunkt, 2015

Ressourcen in REST

- **Jede Ressource wird durch eine URI identifiziert.**
- **Für komplexe Ressourcen, etwa DB-Einträge, wird oft recht künstlich eine URI konstruiert wird.**



vergleiche: Stefan Tilkov; Martin Eigenbrodt; Silvia Schreier; Oliver Wolff: REST und HTTP, 3rd Edition, dpunkt, 2015

Ressourcen in REST

- Ressourcen haben bei REST keinen bestimmten Typ.
- Eine Ressource kann mehrere Repräsentationen haben.
- Damit Client und Server sich verstehen, müssen sie sich auf eine Präsentationen der Ressourcen verständigen.
- Hierzu kann HTTP content negotiation verwendet werden.
  - Client im Header:
    ```
    Accept: text/html, application/xml, application/json
    ```
  - Server im Header:
    ```
    Content-Type: text/html; charset=utf-8
    Content-Type: application/json; charset=utf-8
    ```
- Mehr als eine Repräsentation einer Ressource sind möglich und sinnvoll.

Ressourcen in REST

- Die REST-API ist eine abstrahierte Schnittstelle zu den "wahren" Ressourcen. Eine REST-API ist nicht eindeutig, sondern muss immer designed werden.
- Die Konstruktion sinnvoller URIs ist eine wichtige Aufgabe beim Design einer REST-API
- Die URIs einer API sollen möglichst stabil bleiben. Änderung in der URI-Struktur sind in API-Versionen zusammenzufassen.

Statuslose Kommunikation von REST

- Die Kommunikation soll möglichst statuslos erfolgen. Die einzelnen Request-Response-Paare sollen möglichst weitgehend voneinander entkoppelt sein.
- Bei REST darf der Server keinen Sitzungszustand mitführen.
  Großer Vorteil: Der Server ist jederzeit austauschbar.
  → Docker, Kubernetes, Skalierbarkeit

Bei REST wird der Sitzungszustand komplett vom Client gehalten (Fall 1). Der Server wandelt Sitzungsinformation bei Bedarf in einen Ressourcenstatus um (Fall 2). Ein Ressourcenstatus wird persistent in einer Datenbank gespeichert (anders als ein Sitzungsstatus)

Fall 1: Zustand vollständig auf dem Client.

- Kann z.B. über Cookie realisiert werden. Cookie repräsentiert den gesamten Sitzungszustand.

Fall 2: Server wandelt in Ressourcenstatus um.

- Beispiel: Server speichert Warenkorb in Kundendatenbank.

Insgesamt lockere Kopplung zwischen Client und Server.

HTTP-Methoden + REST

Methode	Beschreibung
GET	Anforderung einer Ressource. Die Ressource wird in der URL spezifiziert. Die URL kann komplex sein (→ REST). GET wird von allen Browsern unterstützt. Mit GET können über die URL auch Daten zum Server übertragen werden. GET ist sicher, idempotent und cacheable. Ohne Body. Beispiel: GET /search?platform=Linux&category=media
RESTful GET	• Versehentliches Löschen von Daten. Beispiel: Wird in der URI im Query-Teil ein delete eingebaut, so können Daten gelöscht werden. http://example.com/videos/operation=delete& key=85434 • Mit Conditional GET kann dem Server mitgeteilt werden, unter welchen Umständen er die Daten liefern soll. Mögliche Header *If-Match*, *If-Modified-Since*

HTTP-Methoden + REST

Methode	Beschreibung

POST	Wird verwendet, um Daten an den Server zu senden. Hauptanwendung Onlineformulare. Format der Daten wird im Attribut Content-Type: beschrieben. Die Daten werden im Body übergeben. Beispiel:
	`Content-Type: application/x-www-form-urlencoded`
	
	`vorname=martin&nachname=leischner`
	
	POST ist nicht sicher, nicht idempotent, bedingt cacheable (d.h. nur mit freshness Info). Mit Body.

RESTful POST	• Kann beispielsweise verwendet werden, um an eine vorhandene Liste eine weiteres Element hinzuzufügen. Das hinzuzufügende Element findet sich im Body des POST-Requests. Bei Erfolg HTTP-Statuscode: Created
	• Serverseitig wird eine neue Ressource erzeugt und eine für diese eine neue Id vergeben.
	• Kann benutzt werden, um Funktionen anzustoßen. In den Body wird einfach das auszuführende Kommando geschrieben. Beispiel:
	
	`POST docklab.de:9998/api/ubuntu`
	
	`apt update & apt upgrade`

HTTP-Methoden + REST

Methode	Beschreibung
HEAD	Gibt nur die Header zurück, die im Fall eines GET-Requests verwendet worden wären. Client kann dann entscheiden, wie er weiter vorgehen will. HEAD ist sicher, idempotent und cacheable. Ohne Body. Beispiel: HEAD /downloads/video.mpeg HTTP/1.1
RESTful HEAD	HEAD-Requests an der API effizient ausführen. Also kein internes GET ausführen und dann alles bis auf die Header wegschmeißen, sondern besser vorher HEAD-Request abfangen und dann nur die Header liefern.

HTTP-Methoden

Methode	Beschreibung
PUT	Kreiert eine neue Ressource oder ersetzt eine vorhandene. Die Daten werden im Body übergeben. Erfolgreiche Antwort (Code 200 oder 204) enthält keinen Body. Das neue Kreieren einer Ressource wird durch den Code 201 (Created) angezeigt. PUT ist nicht sicher, idempotent, nicht cacheable. Mit Body. Beispiel: PUT /api/data/id-1265 HTTP/1.1
RESTful PUT	• Die Daten im Body werden benutzt, um die Ressource zu kreieren, sind aber nicht die Ressource. Sie "repräsentieren" die Ressource. • Ein PUT-Request ist auf die ganze Ressource gerichtet, z.B. Erzeugen einer neuen Ressource mit einer neuen - vom Client vorgegebenen - URI (bzw. Id), vollständiger Update einer Ressource. • Die neue URI bzw. Id wird (im Gegensatz zu POST) bei PUT vom Client vorgegeben.

HTTP-Methoden

Methode	Beschreibung
OPTIONS	Fragt den Server nach den zugelassenen HTTP-Methoden. Hierbei kann eine spezielle URL oder für den ganzen Server das Sternsymbol * angegeben werden. Die Optionen finden sich im Response Header *Allow*. Die Methode OPTIONS wird oft deaktiviert. OPTIONS ist sicher, idempotent, aber nicht cacheable. Ohne Body. Beispiel: OPTIONS http://dh.docklab.de:60000 HTTP/1.1
RESTful OPTIONS	Dem Client wird mitgeteilt, welche Methode von der angefragten Ressource unterstützt wird. Der Api-Entwickler muss sich überlegen, ob er diese Methode unterstützen möchte.

HTTP-Methoden

Methode	Beschreibung
DELETE	Löscht die angegebene Ressource. Wird für API-Server verwendet. Ist bei contentorientierten Servern abgeschaltet. DELETE ist nicht sicher, idempotent, aber nicht cacheable. Body möglich. Beispiel:

RESTful DELETE	• Löschen einer einzigen Ressource über URI problemlos möglich.
	• Löschen mehrerer Ressourcen über einen Query-String mit Ids möglich.
	Vorteil: Schneller als wiederholte Einzellöschung.
	Problem: Viele Ids → lange URI.
	• Übergabe der zu löschenden Ids im Body des DELETE-Requests.
	Problem: Syntax+Semantik des Bodys ist nicht eindeutig definiert.
	• DELETE bedeutet nicht unbedingt physisches/reales Löschen der Ressource, sondern nur gelöscht an der API-Präsentation (konkret: es reicht markiert als gelöscht.)
	• DELETE Status Codes (RFC7231):
	  • 200 (erfolgreich + Statusmeldung im Body)
	  • 202 (Bearbeitung noch nicht beendet)
	  • 204 (erfolgreich + kein Body)

HTTP-Methoden

Methode	Beschreibung
TRACE	Loop-Back-Test. Request wird so zurückgeschickt, wie er empfangen wurde. TRACE ist sicher, idempotent, aber nicht cacheable. Ohne Body. Beispiel:
RESTful TRACE	Keine spezielle RESTful Anwendung.

HTTP-Spezialmethoden

Methode	Beschreibung
PATCH	Während die Methode PUT eine Ressource komplett ersetzt, kann mit PATCH eine Ressource gezielt modifiziert werden. Die im Body spezifizierten Modifikationen müssen auf die Ressource ganz oder gar nicht angewendet werden ("atomar"). Die Modifikationen können z.B. über einen JSON PATCH (RFC6902) syntaktisch korrekt im Body angegeben werden. PATCH ist nicht sicher, nicht idempotent, nicht cacheable. Mit Body. Der Body der Response enthält eine Repräsentation der geänderten Ressource.

RESTful PATCH	Gezieltes Verändern einer Eigenschaft einer Ressource, z.B. Ändern des Preises eines Produkts. Beispiel:
	`PATCH /shop/products/p34 HTTP/1.1`
	`Host: api.myshop.de Content-Type: application/json`
	`{ "price": 5.34 }`
	Der API-Programmierer muss festlegen, was der Body der Response enthält, zum Beispiel die komplette geänderte Ressource /shop/products/p34 mit allen ihren Eigenschaften (nicht nur den Preis).

HTTP-Spezialmethoden

Methode	Beschreibung
CONNECT	Durch CONNECT wird ein Tunnel zur angefragten Ressource aufgebaut. Danach können über den Tunnel "blind" Daten ausgetauscht werden, bis der Tunnel geschlossen wird. CONNECT soll nur für Anfragen an einen Proxy verwendet werden. CONNECT ist nicht sicher, nicht idempotent, nicht cacheable. Ohne Body. Beispiel:
RESTful CONNECT	Keine spezielle RESTful Anwendung.

Beispiel einer einfachen REST-API: Karnevalservice

Der Karneval Service liefert für ein gegebenes Jahr das Datum der Karnevalsfesttage:

- **Weiberfastnacht, Rosenmontag, Aschermittwoch sowie das Datum von Ostersonntag.**

Die Aufrufparameter sind `jahr` und `tag`. Der Parameter `jahr` ist eine natürliche Zahl zwischen 1900 und 2100. Mit dem Parameter `tag` wird der gewünschte Karnevalsfesttag gesteuert. Es gilt:

- **`W` = Weiberfastnacht,**
- **`R` = Rosenmontag,**
- **`A` = Aschermittwoch sowie**
- **`O` = Ostern.**

Der Rückgabewert ist ein JSON String, der die Bezeichnung des Karnevalsfesttags sowie das zugehörige Datum enthält.

REST-Prinzip: Funktionsaufruf und alle Parameter werden für ein GET in die URL kodiert. Ergebnis wird im JSON-Format geliefert.

Beispiel einer REST-API: Karnevalservice

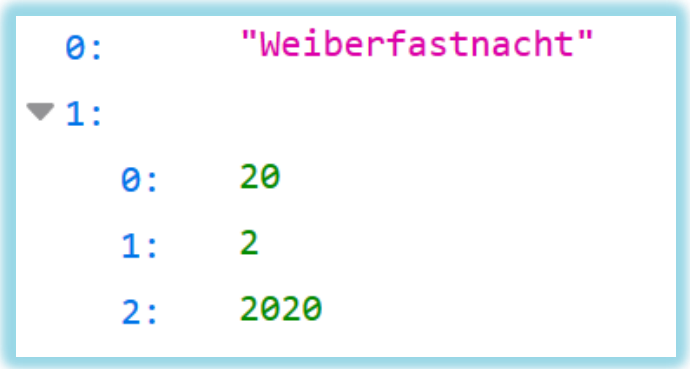
Das Eingabeformat wird sofort aus nachfolgendem Beispiel klar.

Gesucht ist Weiberfastnacht im Jahr 2020. Eingabe-URL:

`https://rest.docklab.de/api/karneval?jahr=2020&tag=W`

Ausgabe:

- **Chrome:** `["Weiberfastnacht",[20,2,2020]]`
- **Firefox:**



Literaturhinweise

- Roy Thomas Fielding: *Architectural Styles and the Design of Network-based Software Architectures*, dissertation, University of California, Irvine, https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf, 2000. (letzter Zugriff 13.05.21)
- Stefan Tilkov, Martin Eigenbrodt, Silvia Schreier, Oliver Wolf: *REST und HTTP, 3rd Edition*, dpunkt, 2015.
- restfulapi.net: *What is REST*, https://restfulapi.net/. (letzter Zugriff 14.05.21)