REST + HATEOAS

Prof. Dr.-Ing. Andreas Heil

Licensed under a Creative Commons Attribution 4.0 International license. Icons by The Noun Project.

v1.0.3

Was ist REST (nicht)?

- REST ist **kein** Protokoll
- REST ist **kein** Standard
- REST ist ein Architektur Stil für netzwerkbasierte Anwendungen
- REST definiert eine Menge von **grundlegenden Prinzipien**

Vorteile von REST

- Statuslos (engl. stateless)
- Skalierbar
- Fehlertolerant
- Lose gekoppelt
- Unterstützt von Natur aus Caching

Grundprinzipien

- Eine **URL**^2 identifiziert eine **Ressource**
- URLs haben eine Hierarchie
- Methoden führen Operationen auf Ressourcen aus
- Operationen sind implizit
- Hypermedia-Format für die Repräsentation von Daten
- Links zur Navigation

Die vier Grundprinzipien

Im Folgenden:

- Identifikation einer Nachricht
- Manipulationen von Ressourcen
- Selbstbeschreibende Nachrichten
- HATEOAS

Identifikation von Ressourcen

- /index.php?action=getarticle&id=376243
- /default/article/3/5/2/size
- ❌ Kann »gecached« werden
- ❌ Skalierbar
- ❌ Lesbar

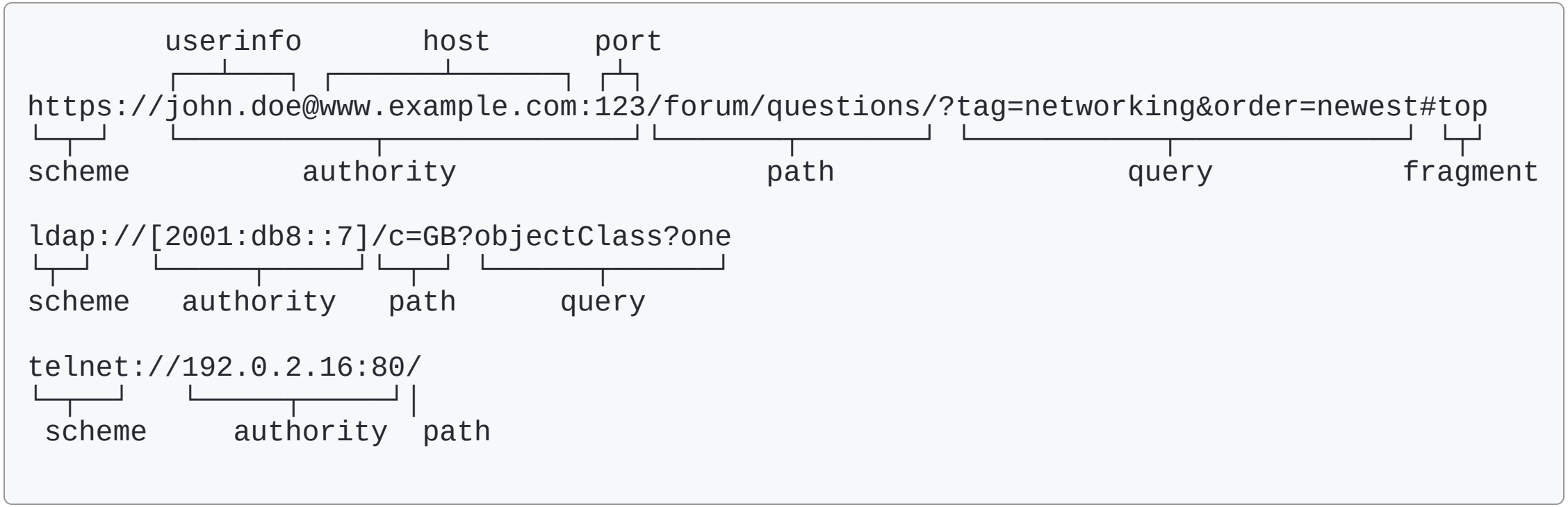
Lesbar und Wartbar

- /articles
  Wir wollen alle Artikel
- /articles/3/photos/5/comments/2
  Wir wollen den zweiten Kommentar am fünften Bild von Artikel 3
- /articles/3/photos/4/comments
  Wir wollen alle Kommentare am vierten Bild von Artikel 3
- ✅ Kann »gecached« werden
- ✅ Skalierbar
- ✅ Lesbar

Filtern mittels Query String, nicht mit der URI

- ❌ /photos/order/size/limit/5
- ❌ /photos/limit/5/order/size
- ✅ /photos?order=size&limit=5
- ✅ /photos?limit=5&order=size

Aufbau einer URL



Quelle: Wikipedia^2

Manipulation von Ressourcen

- **C**reate
- **R**ead
- **U**pdate
- **D**elete
- (**S**earch)

Hinweis: REST ≠ CRUD(S)!!!einself

CRUD HTTP Verb Mapping

- **C**reate ▶ **POST**
- **R**ead ▶ **GET**
- **U**pdate ▶ **PUT** (manchmal auch **PATCH**)
- **D**elete ▶ **DELETE**
- (**S**earch) ▶ **GET**

Resourcen erzeugen

- *POST* erzeugt eine neue Ressource
- Der Server entscheider auf Basis der URI der Ressource
- Beispiele
  - Web: Anlegen eines neuen Posts in einem Blog
    - Server entscheidet über die URI des Blog-Eintrags und z.B. der Kommentare an dem Post
  - Service
    - Anlage eines neuen Datensatzes (z.B. neuer Student*in)

Sichere Operationen

- Jeder Client sollt Requests so oft wiederholen wie nötig
- Grundeigenschaft der Zustandslosigkeit

Idempotenz

- Requests müssen idempotent sein

Beispiel

```
static int a = 0;
int add(int b) {
    a = a+b;
    return a;
}
```

```
int add(int a, int b) {
    return a + b;
}
```

Alle Aufrufe außer **POST**

Selbstbeschreibende Nachrichten

- Weitere Bedingung der Zustandslosigkeit!
- Es müssen alle Informationen für die Bearbeitung der Anfrage vorhanden sein
  - Wie (Methode + Content-Type)
  - Was (URI)
  - Wann (Vorbedingungen)
  - Wer (Authentifikation)

Beispiel

```
GET /person/heil HTTP/1.1
Host: www.hs-heilbronn.de
Accept: application/json,application/xml;q=0.9,*/*;q=0.8
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT
```

Request

- Methode

```
GET /person/heil HTTP/1.1
```

- Content Type^3

```
Accept: application/json,application/xml;q=0.9,*/*;q=0.8
```

- Wer

```
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

- Wann

```
If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT
```

Frage

Woher wissen wir nun, welche URLs wir aufrufen können, wo sich die Ressourcen befinden, was wir damit machen können?

WSDL? WADL? RTFM?

Lösung: HATEOAS

Hypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate

Die vermutlich...

... am meisten missachtete Bedingung bei der Anwendung von REST:

> REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; selfdescriptive messages; and, hypermedia as the engine of application state.

Meint Roy T. Fielding^4... schon mal gehört, ne? Ist der Typ, der REST quasi erfunden hat...

These

Viele Programmierer, die eine »RESTful API« oder einen »REST-Web Service« entwickeln, haben sich nie wirklich mit dem Thema auseinandergesetzte... sonst gäbe es nicht das hier:

- https://api.example.org/v1/booking/
- https://api.example.org/v1.1/booking/

oder noch schlimmer

- https://free.example.org/boking/
- https://pro.example.org/boking/

Aber...

... was war da falsch?

Links (1)

- Links nutzen um dem Client das »Entdecken« von Operationen und Ressourcen zu ermöglichen
- Links beschreiben Möglichkeiten mit der Ressource zu interagieren
- Die URLs müssen den Clients nicht bekannt sein
- Hierdurch lässt sich der Status der Ressource kontrollieren

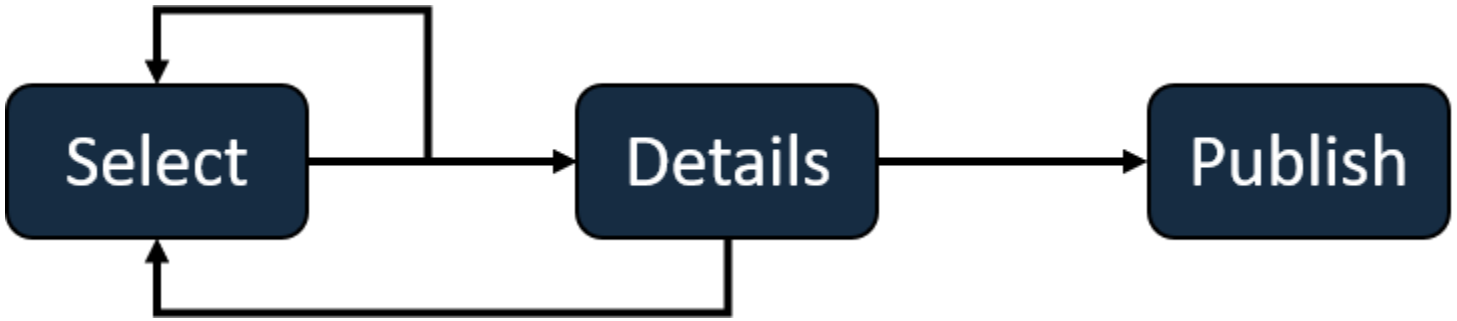
Links (2)

Aufbau von Links

- Target (**href**, erforderlich)
- Beschreibung (kurz!) (**rel**, erforderlich)
  - Z.B.: "details", "cancel", "publish")
- Content Type (**type**, optional)
- HTTP-Methode (**method**, optional)

Noten API

- Status innerhalb der REST API
- Links in HATEOAS beschreiben die Statusübergänge



Suche nach einem/r Student*in

```
POST /search?spo=3 HTTP/1.1
Host: grades.hs-heilbronn.de
Accept: application/vnd.hs-heilbronn.stud-v1+json

{ "studentid": "123456",
  "course": "seks"
}
```

Response

```
HTTP/1.1 200 OK
Content-type: application/vnd.hs-heilbronn.stud-v1.0+json

{ "grades": [
  {"course": "seks", "id": "261761", "date": "2019-09-16", "grade": "5.0",
    "links": [
        { "href": "/student/123456/", "methode": "GET",
          "rel": "details", "type": "application/vnd.hs-heilbronn.stud+json" },
        {
          "href": "/cancel", "methode": "POST",
          "rel": "cancel", "type": "application/vnd.hs-heilbronn.stud+json" }
    ]
  },
   {"course": "seks", "id": "261761", "date": "2020-01-23", "grade": "2.3",
    "links": [
        { "href": "/student/123456/", "methode": "GET",
          "rel": "details", "type": "application/vnd.hs-heilbronn.stud+json" },
        {
          "href": "/cancel", "methode": "POST",
          "rel": "cancel", "type": "application/vnd.hs-heilbronn.stud+json" },
        {
          "href": "/publish", "methode": "POST",
          "rel": "publish", "type": "application/vnd.hs-heilbronn.stud+json" }
    ]
  }
  ]
}
```

REST Klassiker

Versionierung (1)

- ❌ /api/v1.0/student/123456/grades
- ❌ /api/v1.1/student/123456/grades

💥 Nach REST zwei unterschiedliche Ressourcen!

Versionierung (2)

- 🟩 OK

```
GET /api/student/123456/grades HTTPS/1.1
Host hs-heilbronn.de
Accept: application/vnd.hs-heilbronn.de.stud+json;version=1.0
```

- ✅ Besser

```
GET /api/student/123456/grades HTTPS/1.1
Host hs-heilbronn.de
Accept: application/vnd.hs-heilbronn.de.stud+json-v1.1
```

Versionierung (3)

- URLs bleiben bestehen
- keine »Breaking Changes« bei neuen Versionen für alte Clients, keine Anpassung bei alten Clients, kein Aufwand! W00t!!! 😲
- Einfache Wartung und Weiterentwicklung (aka »Evolvierbarkeit«)

URLS und Verben

- ❌ /api/v1.1/student/123456/search/grades
- ❌ /api/v1.1/student/123456/grades/new
- ❌ /api/v1.1/student/123456/grades/list

No comment on this... 🤢

Ein Klassiker

- ❌ /api/student/123456
- ❌ /api/student/aheil

💥 Nach REST zwei unterschiedliche Ressourcen!

Wenn es gar nicht anders geht

```
GET /api/student/aheil/ HTTP/1.1
Host hs-heilbronn.de
Accept: application/vnd.hs-heilbronn.stud-v1+json
```

```
HTTP/1.1 302 Found
Location /api/student/123456
```

HTTP Status Codes (1)

- Status Code sind wichtig
- Repräsentieren das Ergebnis der Aktion
- https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP Status Codes (2)

- 1xx ▶ Informativ
- 2xx ▶ Erfolg
- 3xx ▶ Weiterleitung (engl. redirect)
- 4xx ▶ Client Error
- 5xx ▶ Server Error

Wichtige 200er Codes

- 200 OK ▶ Ressource gefunden
- 201 Created ▶ Ressource angelegt
- 204 No Content ▶ Ressource gelöscht

Wichtige 300er Codes

- 301 Moved Permanently ▶ Ressourcen neu organisiert/verschoben
- 302 Found ▶ Redirect für ein spezielles Objekt (z.B. Suche)
- 303 Other ▶ Redirect aufgrund einer Operation
- 304 Not modified ▶ Ressource wurde nicht geändert

Wichtige 400er Codes

- 400 Bad Request ▶ Fehlerhafter Payload
- 401 Unauthorized ▶ Keine Berechtigung für die Operation
- 403 Forbidden ▶ Keine Berechtigung für die Operation, obwohl angemeldet
- 404 Not found ▶Ressource wurde nicht gefunden
- 405 Method not allowed ▶ Falsche Methode aufgerufen
- 406 Not acceptable ▶ Korrektes Foramt kann nicht geliefert werden
- 412 Precondition failed ▶ ETag stimmt nicht überein
- 418 I'm a teapot ▶ https://www.google.com/teapot^5

Zusammenfassung

- Web-basierte Anwendungen basieren auf Zustandsautomaten und standardisierten Protokollen
- Es gibt viele *Meinungen* was RESTful ist, jenachdem mit wem man spricht
- Oftmals ein verlorener Kampf...
- Es gibt gute und schlechte REST-APIs, mit und ohne Hypermedia

Acknowledgement

Diese Vorlesung basiert teilweise auf einer Vorlesung von
Guy K. Kloss, Auckland University of Technology^1

Referenzen