

A thick red vertical bar is positioned on the left side of the slide, extending from the top to the bottom.

Webbasierte Anwendungen

RESTful Web Services

Prof. Dr. Ludger Martin

Gliederung

- ◆ Einführung Web Services
- ◆ RESTful Web Services
 - ★ Methoden
 - ★ Ressource
 - ★ Server
 - ★ Client
 - ★ Fehlerbehandlung
 - ★ Ressource Design

Einführung Web Service

Definition (John Hagel III, John S. Brown):

At its core is the assumption that companies will in the future **buy** their information technologies as **services** provided over the **Internet** rather than owning and maintaining all their own hardware and software.

Einführung Web Service

- ◆ **Kommunikationspartner** sind nur **Maschinen**
(Anwendungen auf Web-Servern)
 - Bisher waren im Web nur menschliche Nutzer
- ◆ Anwendungen oft in **verschiedenen Sprachen** entwickelt
 - Daher muss die Kommunikation von allen verstanden werden
- ◆ Austausch von Daten durch **Übertragungsstandards** zwischen Kommunikationspartnern

Einführung Web Service

- ◆ W3C Definition von Web Services:
 - ★ Vollständig kompatibles System
 - ★ Selbstbeschreibende Schnittstelle
 - ★ Interaktion mit anderen Systemen über standardisierte Nachrichten

Einführung Web Service

♦ Vorteile:

- ★ **Einfacher** zu handhaben (im Gegensatz zu Corba/DCOM)
- ★ Vereinfachen die Kommunikation zwischen Firmen mit **verschiedenen Infrastrukturen**
- ★ Die Spanne an möglichen E-Business Lösungen wird erhöht (Erfragung der Lieferkette, Arbeitsgemeinschaften, Handelsgemeinschaften)
- ★ Ein Web Service kann im elektronischen Handel für viele Systeme bereitgestellt werden und fördert so den Wettbewerb

Einführung Web Service

- ◆ Vorteile: (Fortsetzung)
 - ★ Web Services können auch komplexe Workflow-Systeme abbilden
 - ★ **Kommunikation über Internet** (HTTP), Firewalls müssen nicht angepasst werden
 - ★ Basieren auf **anerkannten Standards** (Nachrichtenübertragung, Verzeichnisdienste)
 - ★ Web Services für fast alle Programmiersprachen vorhanden

Einführung Web Service

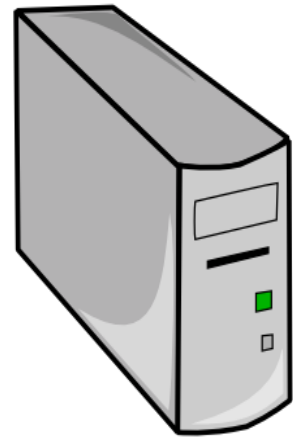
◆ Nachteile:

- ★ Web Service Anfragen **passieren Firewalls**, so können auch potentielle Angreifer passieren
→ Firewalls müssen Anfragen untersuchen
- ★ Oft **nicht** die **leistungsfähigste** Lösung

RESTful Web Services

- ♦ REST: **R**epresentational **S**tate **T**ransfer
- ♦ Dissertation von Roy Fielding.
University of California. 2000
- ♦ Ziel ist die Einfachheit
- ♦ Realisierung von Web Services in webgerechter Weise
- ♦ Ressourcenorientiert – auch als **ROA**
(**R**esource **O**riented **A**rchitecture) bezeichnet

RESTful Web Services



Web Service

Ein **Web Service**, der
Aktienkurswerte liefert.

Ein **Konsument** möchte z.B.
einen Web Service nutzen,
der Aktienkurswerte liefert.



Konsument

RESTful Web Services

Die **Selbstbeschreibung** informiert über die Kommunikationsmöglichkeiten mit dem Web Service, z.B. dass er den Aktiennamen erwartet und den Aktienpreis zurückliefert.

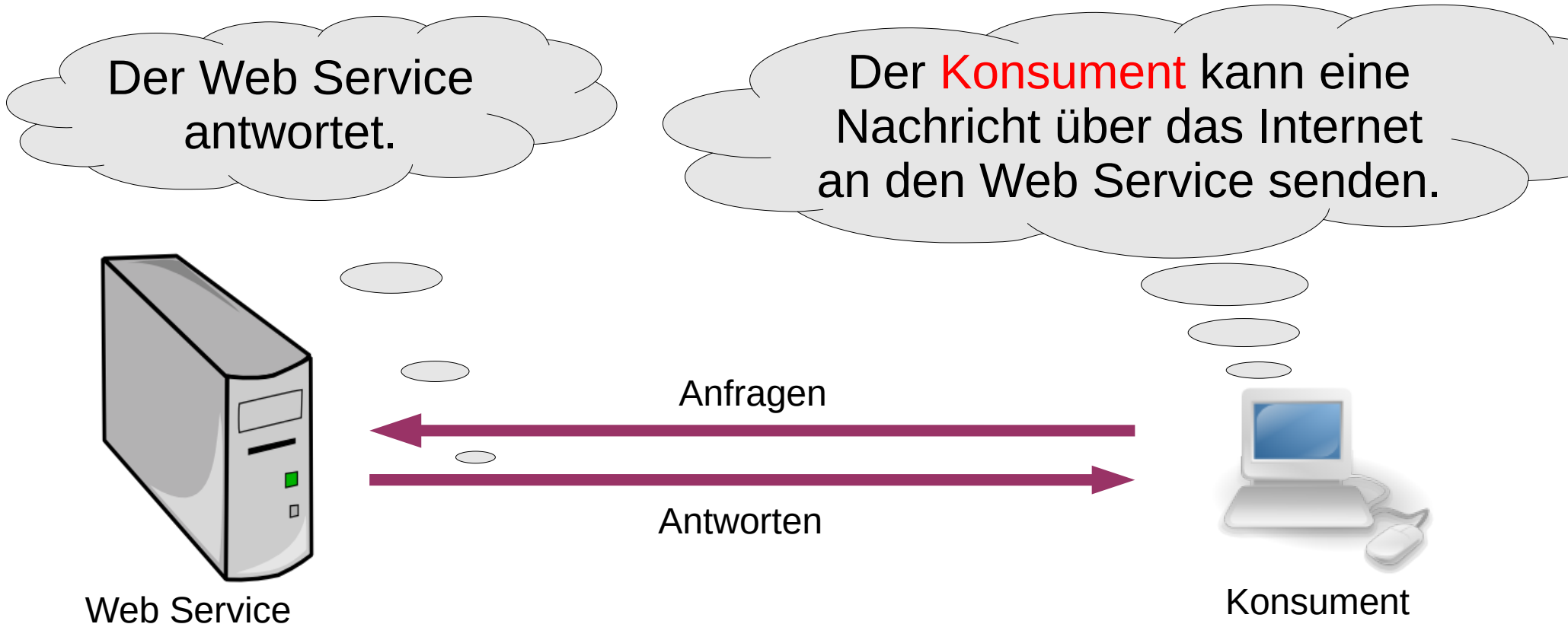


Web Service

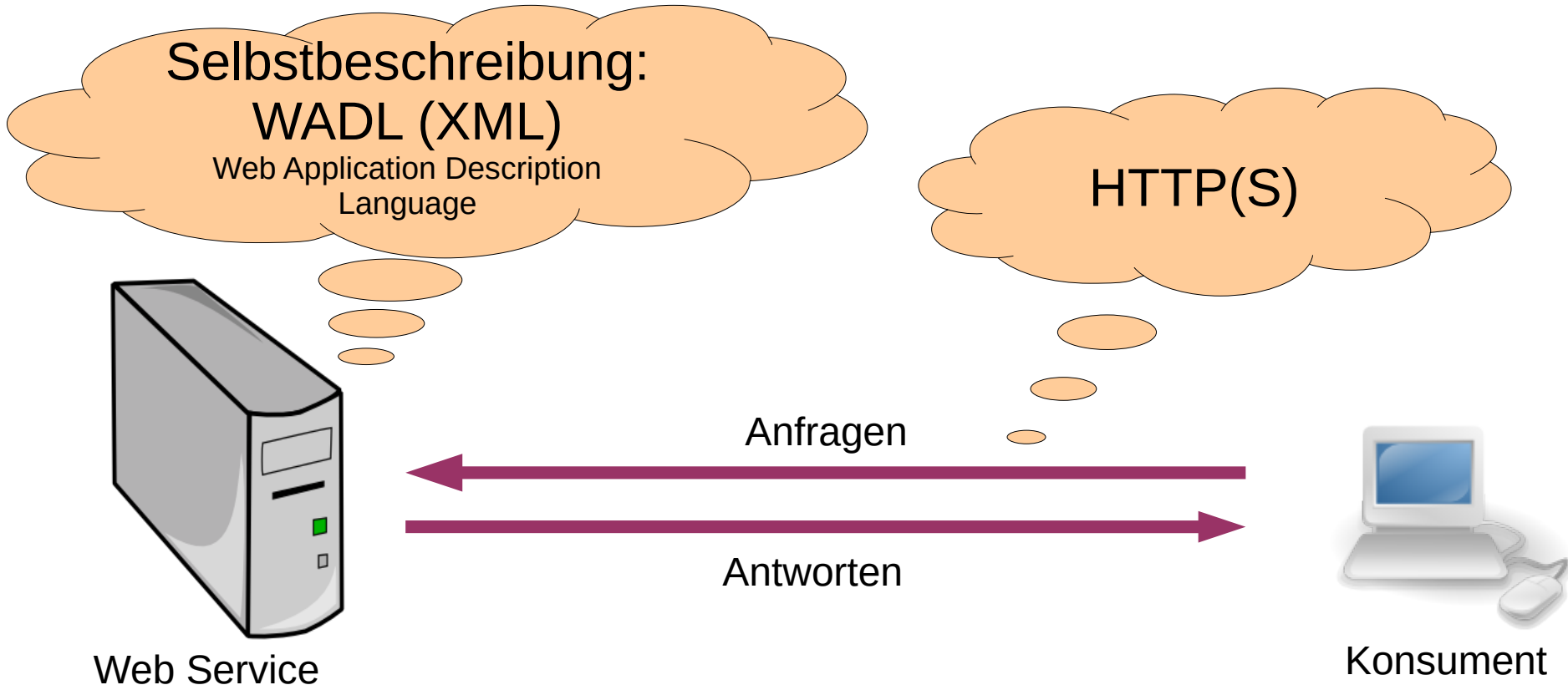


Konsument

RESTful Web Services



RESTful Web Services



RESTful Web Services

♦ Ressourcen

- ★ ein Web Service kapselt Ressourcen
- ★ Beispiele für Ressourcen: Bücher, News, Bilder, ...
- ★ Ressourcen werden über einen standardisierten und einheitlichen Mechanismus angesprochen, die URI
- ★ eine Ressource kann ihren Zustand ändern

♦ Repräsentationen


- ★ sind die Darstellungen einer Ressource

RESTful Web Services

Methoden

◆ GET

- ★ fragt die Repräsentation einer Ressource ab
- ★ lesende Operation
- ★ idempotent



Ähnlich zu Level 2
Richardson Maturity Model
aber mit weiteren HTTP Methoden

◆ PUT

- ★ erzeugt eine neue Ressource oder aktualisiert sie
- ★ idempotent

◆ POST

- ★ erzeugt eine neue Ressource oder kann einer bestehenden etwas hinzufügen
- ★ nicht idempotent

→ problematisch Netzwerkausfall

RESTful Web Services

Methoden

◆ DELETE


- ★ löscht eine Ressource
- ★ idempotent

◆ HEAD

- ★ Metadaten über eine Ressource holen
- ★ selten genutzt
- ★ idempotent

◆ OPTIONS

- ★ Rückgabe unterstützter Methoden
- ★ selten genutzt
- ★ idempotent



CONNECT, TRACE und PATH
in der Regel nicht genutzt

RESTful Web Services

Ressource

- ◆ Beispiel Bibliothek:
 - ★ besteht aus einer Reihe von Büchern
 - ★ neue Bücher können hinzugefügt werden
 - ★ es gibt Bibliotheksnutzer
 - ★ Nutzer darf sich 2 Bücher ausleihen
 - ★ Nutzer kann Bücher zurückbringen
 - ★ Auflisten von Büchern und Nutzern

RESTful Web Services

Ressource

- ◆ Um Objekte zu finden, wird nach Nomen geschaut
 - ★ Buch
 - ★ Nutzer
- ◆ alle Geschäftsabläufe werden zu Methoden
 - ★ neues Buch/Nutzer aufnehmen
 - ★ Bücher/Buchdaten und Nutzer/Nutzerdaten auflisten
 - ★ Buchstatus erfassen (Ausleihe/Rückgabe)
 - ★ Buchdaten ändern
 - ★ Buch löschen

RESTful Web Services

Ressource

- ◆ Operationen, HTTP Methoden URIs

HTTP Methoden	URI	Beschreibung
GET	/book	Bücher auflisten
POST*	/book	Buch aufnehmen
GET	/book/42	Buch holen
PUT	/book/42	Buchdaten ändern
DELETE	/book/42	Buch löschen

- ◆ URI: /book oder /book/<b_id>

- ◆ einfache Operationen sind nun möglich

*mit anderer URI könnte ggf. auch PUT

genutzt werden

RESTful Web Services

Ressource

- ♦ für komplexere Operationen wie Verleih und Rückgabe muss die URI erweitert werden

- ★ Verleih

PUT /user/<u_id>/book/<b_id>

- ★ Rückgabe

DELETE /user/<u_id>/book/<b_id>

RESTful Web Services

Ressource

URI	HTTP	Erfasst	Operation
/book	GET	book	alle Bücher auflisten
/book	POST*	book	Buch neu aufnehmen
/book/<b_id>	GET	book	Buchdaten auflisten
/user	GET	user	alle Nutzer auflisten
/user	POST*	user	neuen Nutzer aufn.
/user/<u_id>	GET	user	Daten eines Nutzers
/user/<u_id>/book	GET	user	zeige Ausleihstatus von Büchern
/user/<u_id>/book/<b_id>	PUT	user	Leihe Buch aus
/user/<u_id>/book/<b_id>	DELETE	user	Buch Rückgabe

*mit anderer URI könnte ggf. auch PUT genutzt werden

RESTful Web Services

Ressource

- ◆ Ressourcen können unterschiedlichen Typ haben, sowohl in der Anfrage als auch in der Antwort.
- ◆ Im HTTP-Protokoll als `Content-Type` angegeben.
- ◆ Ein Typ kann als MIME spezifiziert werden
 - ★ `text/plain`
 - ★ `text/xml`
 - ★ `application/json`
 - ★ `image/png`
 - ★ ...

RESTful Web Services

Server

◆ Beispiel: Buchdaten auflisten

```
var app = express();

app.get('/book/:b_id', function (req, res) {
  res.set('Content-Type',
    'application/json');
  let book = { "b_id": req.params.b_id,
    "title": "Pippi Langstrumpf"
  };
  res.send(JSON.stringify(book));
});
```

RESTful Web Services

Client

◆ Daten von einem Buch auflisten

```
const axios = require('axios');
```

```
axios.get('http://localhost:3000/book/42')  
  .then(function (response) {  
    // handle success  
    console.log(response);  
  })  
  .catch(function (error) {  
    // handle error  
    ...  
  })  
  .finally(function () {  
    // always executed  
  });
```

axios Bibliothek,
Modul `axios` muss
installiert werden

```
axios.get(url[, config])
```

Wir werden noch eine
andere Bibliothek für den
Browser kennenlernen

RESTful Web Services

Fehlerbehandlung

♦ Im Service

- ★ HTTP-Statuscode wird genutzt
- ★ (<http://de.wikipedia.org/wiki/HTTP-Statuscode>)
- ★ Wenn Status 200 OK ist, ist keine Aktion notwendig
- ★ Ansonsten Status mit `res.status(code)` setzen
`res.status(400).send('Bad Request');`

♦ Im Client

- ★ `status`-Attribut von `response-Object` (axios)

RESTful Web Services

Fehlerbehandlung

- ◆ Die wichtigsten (Fehler-)Meldungen
 - ★ erfolgreich: 200 OK
 - ★ erfolgreich Ressource angelegt: 201 Created
 - ★ falsches Format: 415 Unsupported Media Type
 - ★ sinnlose Daten: 400 Bad Request
 - ★ inkonsistent/unmöglich: 400 Bad Request
409 Conflict
 - ★ Anmeldung erforderlich: 401 Unauthorized
 - ★ existiert bereits: 409 Conflict
 - ★ unspezifiziert: 500 Internal Server Error
503 Service Unavailable

RESTful Web Services

Ressource Design

- ◆ Exkurs: objektorientierter Entwurf
 - ★ System in bewegliche Teile zerlegen → Nomen
 - ★ Jedes Nomen bekommt eigene Klasse
 - ★ Das Verhalten mit anderen Nomen wird mit Methoden der Klasse abgebildet
- ◆ Exkurs: RPC
 - ★ Verben (Motions) werden Prozeduren

RESTful Web Services

Ressource Design

◆ Abbildung auf Ressource

- ★ Problem: in HTTP ist keine unendliche Menge an Methoden vorhanden
 - Create (PUT, POST)
 - Modify (PUT)
 - Read (GET)
 - Delete (DELETE)
- ★ Zur Abhilfe muss ein Verb/Methode (aus OO-Design) in ein Objekt/Ressource umgewandelt werden
 - subscribe to → subscription

RESTful Web Services

Ressource Design - nur lesend

- ◆ Design von „read-only resources“
- ◆ Ausschließlich GET
- ◆ 1. Schritt: „data set“ – Beispiel: „map service“
 - ★ Karten mit Längen- und Breitengraden
 - ★ Punkte – Orte, Länder, Flüsse
 - ★ Orte mit Namen – evtl. nicht eindeutig
 - ★ Orte in der Nähe von anderen Orten

RESTful Web Services

Ressource Design - nur lesend

◆ 2. Schritt: Ressourcen bilden

- ★ Ressource: „anything interesting enough to be the target of a hyper link“
- ★ 3 Arten
 - „**predefined one-off**“: Beschreibung von anderen Ressourcen (Selbstbeschreibung)
 - **Objekte**: alle Objekte, die von dem Service bereitgestellt werden sollen – evtl. große Anzahl oder unendlich viel
 - **Ergebnisse eines Algorithmus**: ausgeführt auf das „data set“ – z.B. Anfrage an eine Suchmaschine
- ★ map service: overview page, point resource, search results

RESTful Web Services

Ressource Design - nur lesend

◆ 3. Schritt: Name vergeben

- ★ Name ist die URI
- ★ URI soll alle Abgrenzungen enthalten
- ★ Pfadvariablen bilden Hierarchie ab:
`/parent/child`
- ★ Interpunktion symbolisieren gleiche Ebene
`/parent/child1;child2`
- ★ (';' Reihenfolge wichtig - ',' Reihenfolge uninteressant)
- ★ Query-Variablen als Eingabe von Algorithmen
`/parent/child?q=value&...`

Domain jeweils weggelassen

RESTful Web Services

Ressource Design - nur lesend

◆ map service:

- ★ `/America`
- ★ `/Europe`
- ★ `/Europe/Berlin`
- ★ `/America/Mount%20Rushmore`
- ★ `/43.9,-103.46` (Mount Rushmore)
- ★ `/?show=Springfield`
- ★ `/America/Mount%20Rushmore?show=dinners`

RESTful Web Services

Ressource Design - nur lesend

- ◆ 4. Schritt: Typ der Ressource (nur Antwort)
 - ★ HTML
 - ★ JSON
 - ★ XML
 - ★ Image
 - ★ ...

RESTful Web Services

Ressource Design - nur lesend

- ◆ Bei nur lesenden Operationen bietet es sich an, Zeit und Bandbreite zu sparen

- ★ Last-Modified

- Frühere Antwort: `Last-Modified: Tue, 08 May 2012 20:00:51 GMT`
- Neue Anfrage: `If-Modified-Since: Tue, 08 May 2012 20:00:51 GMT`
- Neue Antwort: `HTTP/1.0 304 Not Modified`

- ★ Nutzung von ETag

- Frühere Antwort: `ETag: 346fb23d0f23`
- Neue Anfrage: `If-None-Match: 346fb23d0f23`
- Neue Antwort: `HTTP/1.0 304 Not Modified`

RESTful Web Services

Ressource Design – lesend und schreibend

- ♦ Soll lesend und schreibend zugegriffen werden, so sind oft Accounts notwendig
- ♦ Oft mit API-Keys auf Nutzer- oder Anwendungsebene
- ♦ Auch mit HTTP Authorization möglich
 - ★ Anmeldeinformationen im HTTP-Header enthalten
 - ★ Wenn nicht autorisiert: 401 Unauthorized
WWW-Authenticate

RESTful Web Services

Ressource Design – lesend und schreibend

- ◆ Arten der HTTP Authorization (Auswahl)
 - ★ Es können auch mehrere gewählt werden
 - ★ Basic
 - Server:
`WWW-Authenticate: Basic realm="User Visible Realm"`
 - Client:
In Form `Benutzername:Passwort` **als Base64-codiert übertragen**
`Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

RESTful Web Services

Ressource Design – lesend und schreibend

♦ Arten der HTTP Authorization (Auswahl)

★ Digest Access

- Server sendet im WWW-Authenticate-Header zufällig erzeugte Zeichenfolge (nonce)

- Browser berechnet

$$HA1 = MD5(A1) = MD5(\text{username} : \text{realm} : \text{password})$$

$$HA2 = MD5(A2) = MD5(\text{method} : \text{digestURI})$$

$$\text{response} = MD5(HA1 : \text{nonce} : HA2)$$

- Sicherer als Basic

RESTful Web Services

Ressource Design – lesend und schreibend

- ◆ Accounts können auch über Ressourcen verwaltet werden
- ◆ 1. Schritt: „data set“
`username` und `password`
- ◆ 2. Schritt: Ressourcen bilden
`user account`
- ◆ 3. Schritt: Name vergeben
Benutzername ist die Abgrenzung → URI
`.../user/{user-name}`

RESTful Web Services

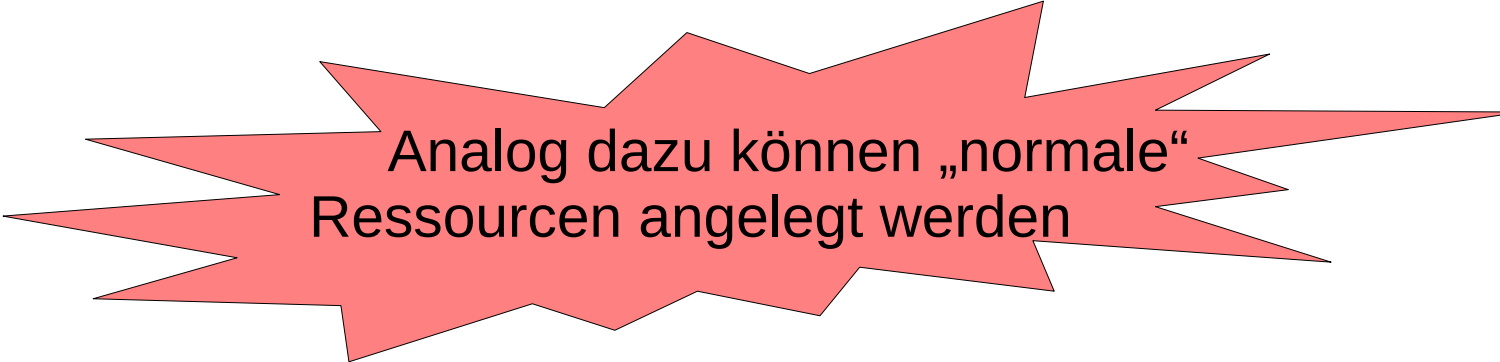
Ressource Design – lesend und schreibend

- ◆ 4. Schritt: Repräsentationsdesign
 - ★ Anlegen: PUT ohne Authentifizierung, Passwort im body
 - ★ Ändern: PUT mit Authentifizierung, neues Passwort im body
 - ★ Löschen: DELETE mit Authentifizierung
 - ★ Abfragen: GET gibt kein Passwort zurück, ggf. Infos wie ein Account angelegt werden kann
→ Selbstbeschreibung

RESTful Web Services

Ressource Design – lesend und schreibend

- ◆ 5. Schritt: Typ der Ressource (Anfrage und Antwort)
 - ★ HTML
 - ★ JSON
 - ★ XML
 - ★ Image
 - ★ ...



Analog dazu können „normale“
Ressourcen angelegt werden

RESTful Web Services

Ressource Design – lesend und schreibend



Vertraulichkeit durch HTTPS!

Literatur

- ◆ Leonard Richardson und Sam Ruby: RESTful Web Services, O'Reilly, 2007
- ◆ Samisa Abeysinghe: RESTful PHP, Packt Publishing, 2008
- ◆ npm Inc: axios, <https://www.npmjs.com/package/axios>,
zugegriffen 15.05.2023