RESTful Web Services

Eine sehr kurze Einführung.

Dozent: Prof. Dr. Michael Eichberg

Kontakt: michael.eichberg@dhbw-mannheim.de

Version: 1.0

Quelle: (teilweise) RESTful Web Services; Leonard Richardson & Sam Ruby; O'Reilly

Folien: https://delors.github.io/ds-restful/folien.de.rst.html

https://delors.github.io/ds-restful/folien.de.rst.html.pdf

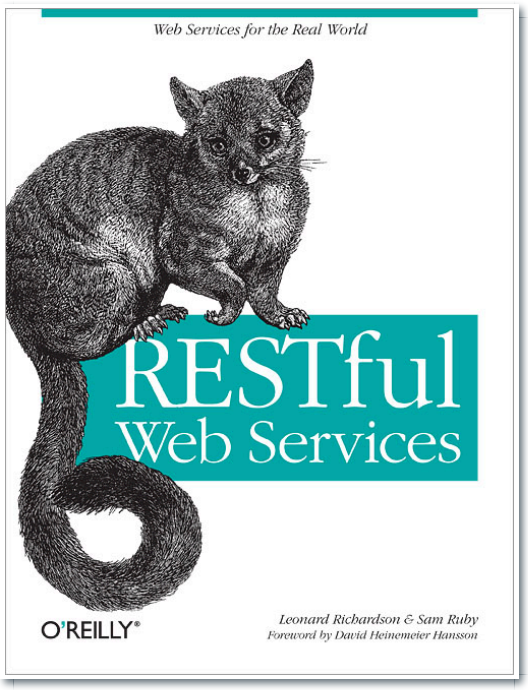
Fehler auf Folien melden:

https://github.com/Delors/delors.github.io/issues

Was ist ein *Web Service* im Kontext von RESTful Web Services

Ein *Web Service* ist lediglich eine Webseite, die von einem Computer angefordert und verarbeitet werden kann.

Ein *Web Service* ist eine „Webseite", die von einem *autonomen Programm* - im Gegensatz zu einem Webbrowser oder einem ähnlichen UI-Tool - konsumiert werden soll.



REST[1]

- REST = Representational State Transfer

  (im Wesentlichen eine Reihe von Entwurfsprinzipien zur Beurteilung von Architekturen; **ein Architekturstil**).
- Ressourcen werden durch einheitliche Ressourcenbezeichner (URIs) identifiziert
- Ressourcen werden durch ihre Repräsentationen manipuliert
- Nachrichten sind selbstbeschreibend und zustandslos

Von untergeordneter Bedeutung:

- Mehrere Repräsentationen werden akzeptiert oder gesendet
- „Hypertext" repräsentiert den Anwendungszustand

[1] REST wurde von Roy Fielding in seiner Dissertation beschrieben.

Eine mögliche Architektur für RESTful Web Services

Resource-oriented Architecture (ROA)

- Informationen zur Methode werden in die HTTP-Methode aufgenommen.
- Scoping-Informationen gehen in den URI ein. (d.h. welche Daten sind betroffen.)

REST-Stil

- Client-server
- Zustandslos 🇺🇸 *stateless*
- Cached
- Uniforme Schnittstelle (HTTP Methoden)
- Mehrschichtiges System

RESTful Web Services - Grundlagen

HTTP: das zugrunde liegende zustandslose Transportprotokoll:

Wesentliche Methoden:

GET:	seiteneffektfreie Abfragen von Informationen
POST:	Hinzufügen von neuen Informationen (ohne Angabe der Ziel URI)
PUT:	idempotente Aktualisierung oder Neuerzeugung von Informationen an der gegebenen URI
DELETE:	idempotentes Löschen von Informationen

URI: dient dem Auffinden von Ressourcen

„Repräsentation”: **JSON**, XML, SVG, WebP, XML, ...

Zwei Arten von Zustand: (1) Anwendungs-/Sitzungszustand

(🏴 *Application State / Session State*)

- „Zustand” bedeutet Anwendungs-/Sitzungsstatus

  > Der Anwendungsstatus ist die Information, die notwendig ist, um den Kontext einer Interaktion zu verstehen
  >
  > Autorisierungs- und Authentifizierungsinformationen sind Beispiele für den Anwendungsstatus.

- Wird als Teil des vom Client zum Server und zurück zum Client übertragenen Inhalts beibehalten. d.h. der Client verwaltet den Anwendungszustand.
- Somit kann jeder Server die Transaktion potenziell an dem Punkt fortsetzen, an dem sie unterbrochen wurde.

Zwei Arten von Zustand: (2) Ressourcenzustand

(🇺🇸 *Resource State*)

- Der Ressourcenzustand ist die Art von Zustand, auf die sich das *S* in *REST* bezieht.
- Die Einschränkung „zustandslos” bedeutet, dass alle Nachrichten den gesamten Anwendungsstatus enthalten müssen (d.h., dass wir effektiv keine Sitzungen haben).

Mehrere Repräsentationen

- Die meisten Ressourcen haben nur eine einzige Darstellung.
- REST kann jeden Medientyp unterstützen; JSON ist der Standard.

  (HTTP unterstützt die Aushandlung von Inhalten.)
- Links können eingebettet werden und spiegeln die Struktur wieder, mit der sich ein Benutzer durch eine Anwendung bewegen kann.

Einfache/Erste Tests auf RESTfulness

- Kann ich die URLs, an die ich POSTe, mit einem GET abrufen?
- Würde der Client merken, wenn der Server...
  - an einem beliebigen Punkt zwischen den Anfragen neu gestartet wird
  - neu initialisiert wird, wenn der Client die nächste Anfrage stellt.

Ressourcenmodellierung

- Organisation der Anwendung in URI-adressierbare Ressourcen (diskrete Ressourcen sollten ihre eigenen stabilen URIs erhalten.)
- nur die Standard-HTTP-Nachrichten - GET, PUT, POST, DELETE und PATCH - verwenden, um die vollen Fähigkeiten der Anwendung bereitzustellen

HTTP Methoden

GET dient dem Abfragen von Ressourcen.

PUT dient dem Anlegen einer Ressource oder dem Aktualisieren, wenn man die URI kennt.

POST dient dem Erzeugen einer neuen Ressource. Die Antwort sollte dann die URI der angelegten Ressource enthalten.

DELETE löscht die angegebene Ressource.

Der Unterschied zwischen **PUT** und **POST** besteht darin, dass **PUT** idempotent ist: der einmalige oder mehrmalige Aufruf hat die gleiche Wirkung (d.h. keine Nebenwirkung), während aufeinanderfolgende identische **POST** Aufrufe zusätzliche Wirkungen haben können, wie z.B. die mehrmalige Übergabe eines Auftrags/das mehrmalige Anlegen einer Nachricht.

Eine **PATCH**-Anfrage wird als ein Satz von Anweisungen zur Änderung einer Ressource betrachtet. Im Gegensatz dazu ist eine PUT-Anfrage eine vollständige Darstellung einer Ressource.

Beispielanwendung del.icio.us

Quelle: https://www.peej.co.uk/articles/restfully-delicious.html

del.icio.us ermöglicht es:

- eine Liste aller unserer Lesezeichen zu erhalten und diese Liste nach Marker oder Datum zu filtern bzw. die Anzahl zu begrenzen
- Die Anzahl der Lesezeichen, die an verschiedenen Tagen erstellt wurden, abzurufen
- abzufragen wann wir das letzte Mal unsere Lesezeichen aktualisiert haben
- eine Liste all unserer Marker abzurufen
- hinzufügen eines Lesezeichens
- bearbeiten eines Lesezeichens
- löschen eines Lesezeichens
- umbenennen eines Markers

Beispielanwendung del.icio.us: Ressourcen

Lesezeichen: *http://del.icio.us/api/[username]/bookmarks*
Marker: *http://del.icio.us/api/[username]/tags*
[username]: ist der Benutzername des Nutzers, an dessen Lesezeichen wir interessiert sind

Beispielanwendung del.icio.us: Repräsentation von Ressourcen

Wir definieren (in diesem Beispiel) einige XML-Dokumentformate und Medientypen, um sie zu identifizieren:

Mediatype	**Description**
delicious/bookmarks+xml	Liste von Lesezeichen
delicious/bookmark+xml	ein Lesezeichen
delicious/bookmarkcount+xml	Anzahl der Lesezeichen eines Tage
delicious/update+xml	Zeitpunkt wann die Lesezeichen zuletzt aktualisiert wurden
delicious/tags+xml	eine Liste von Markern
delicious/tag+xml	ein Marker

Beispielanwendung del.icio.us: Lesezeichen abfragen

URL:	*http://del.icio.us/api/[username]/bookmarks/*
Methode:	GET
Querystring:	tag= Filtern nach Marker
	dt= Filtern nach Datum
	start= Die Nummer des ersten zurückzugebenden Lesezeichen
	end= Die Nummer des letzten zurückzugebenden Lesezeichen
Rückgabewert:	200 OK & XML (delicious/bookmarks+xml)
	401 Unauthorized
	404 Not Found

Beispielanwendung del.icio.us: Lesezeichen abfragen - Beispielantwort

`GET http://del.icio.us/api/peej/bookmarks/?start=1&end=2`

```
<?xml version="1.0"?>
<bookmarks start="1" end="2"
    next="http://del.icio.us/api/peej/bookmarks?start=3&amp;end=4">
    <bookmark url="http://www.example.org/one" tags="example,test"
        href="http://del.icio.us/api/peej/bookmarks/a211528fb5108cddaa4b0d3aeccdbdcf"
        time="2005-10-21T19:07:30Z">
        Example of a Delicious bookmark
    </bookmark>
    <bookmark url="http://www.example.org/two" tags="example,test"
        href="http://del.icio.us/api/peej/bookmarks/e47d06a59309774edab56813438bd3ce"
        time="2005-10-21T19:34:16Z">
        Another example of a Delicious bookmark
    </bookmark>
</bookmarks>
```

Beispielanwendung del.icio.us: Informationen bzgl. eines Lesezeichens

URL: *http://del.icio.us/api/[username]/bookmarks/[hash]`*

Methode: *GET*

Rückgabewert: 200 OK & XML (delicious/bookmark+xml)

401 Unauthorized

404 Not Found

Beispielanwendung del.icio.us: Informationen bzgl. eines Lesezeichens - Beispielantwort

`GET http://del.icio.us/api/peej/bookmarks/a211528fb5108cdd`

```
<?xml version="1.0"?>
<bookmark url="http://www.example.org/one" time="2005-10-21T19:07:30Z">
    <description>
        Example of a Delicious bookmark
    </description>
    <tags count="2">
        <tag name="example" href="http://del.icio.us/api/peej/tags/example"/>
        <tag name="test" href="http://del.icio.us/api/peej/tags/test"/>
    </tags>
</bookmark>
```

Beispielanwendung del.icio.us: Abfrage der Anzahl der Lesezeichen

URL: *http://del.icio.us/api/[username]/bookmarks/count*

Methode: GET

Abfrageparameter:

tag= filter by tag

Rückgabewert: 200 OK & XML (delicious/bookmark+xml)

401 Unauthorized

404 Not Found

Beispielanwendung del.icio.us: Abfrage wann die letzte Änderung vorgenommen wurde

URL:	*http://del.icio.us/api/[username]/bookmarks/update*
Methode:	GET
Rückgabewert:	200 OK & XML (delicious/bookmark+xml) 401 Unauthorized 404 Not Found

Beispielanwendung del.icio.us: Hinzufügen eines Lesezeichens

URL: *http://del.icio.us/api/[username]/bookmarks/`*

Methode: POST

Anfragedokument:

XML (delicious/bookmark+xml)

Rückgabe: 201 Created & Location

401 Unauthorized

415 Unsupported Media Type(if the send document is not valid)

Beispielanwendung del.icio.us: Hinzufügen eines Lesezeichens - Beispielübermittlung

POST http://del.icio.us/api/peej/bookmarks/

```
<?xml version="1.0"?>
<bookmark url="http://www.example.org/one"
    time="2005-10-21T19:07:30Z">
    <description>Example of a Delicious bookmark</description>
    <tags>
        <tag name="example" />
        <tag name="test" />
    </tags>
</bookmark>
```

Beispielanwendung del.icio.us: Aktualisierung eines Lesezeichens

URL: *http://del.icio.us/api/[username]/bookmarks/[hash]`*

Methode: PUT

Anfragedokument:

XML (delicious/bookmark+xml)

Rückgabewert: 201 Created & Location

401 Unauthorized

404 Not Found (new bookmarks cannot be created using put!)

415 Unsupported Media Type (if the send document is not valid)

Beispielanwendung del.icio.us: Löschen eines Lesezeichens

URL: *http://del.icio.us/api/[username]/bookmarks/[hash]*

Methode: DELETE

Rückgabewert: 204 No Content

401 Unauthorized

404 Not Found