# RESTful Web Services

A brief introduction.


| | |
|---|---|
| **Dozent:** | Prof. Dr. Michael Eichberg |
| **Kontakt:** | michael.eichberg@dhbw.de |


| | |
|---|---|
| **Version:** | 1.0 |
| **Quelle:** | (teilweise) RESTful Web Services; Leonard Richardson & Sam Ruby; O'Reilly |

| | |
|---|---|
| **Slides/Script:** | https://delors.github.io/ds-introduction_to_rest/folien.en.rst.html<br>https://delors.github.io/ds-introduction_to_rest/folien.en.rst.html.pdf |
| **Reporting errors:** | https://github.com/Delors/delors.github.io/issues |

# What is a *Web Service* in the context of RESTful Web Services?

## Traditional view

A *Web Service* is simply a web page that can be requested and processed by a computer.

A *Web Service* is a "web page" that is to be consumed by an *autonomous programme* - as opposed to a web browser or similar UI tool.

# REST[1]

- REST = Representational State Transfer

  (Essentially a set of design principles for judging architecture; **an architectural style**).
- Resources are identified by uniform resource identifiers (URIs)
- Resources are manipulated by their representations
- Messages are self-describing and stateless

Of secondary importance:

- Multiple representations are accepted or sent
- "Hypertext" represents the application state

---

[1] REST was described by Roy Fielding in his dissertation.

# A possible architecture for RESTful web services

**Resource-oriented Architecture (ROA)**

- Information about the method is included in the HTTP method.
- Scoping information is included in the URI.

  (I. e. which data is affected.)

## REST-Style

- Client-server
- stateless
- Cached
- Uniform Interface (HTTP Methods)
- Multi-layered system

| | | |
|---|---|---|
| **HTTP:** | the underlying stateless transport protocol: | |
| | Essential methods: | |
| | **GET:** | sideeffect-free requests for information |
| | **POST:** | adding new information (without specifying the target URI) |
| | **PUT:** | idempotent update or creation of new information at the given URI |
| | **DELETE:** | idempotent deletion of information |
| **URI:** | used to find resources | |
| **Representation:** | **JSON**, XML, SVG, WebP, XML, … | |

# Two Types of State

1

## Application State / Session State

- "State" refers to Application-/Session State

  The application state is the information necessary to understand the context of an interaction

Authorization and authentication information are examples of application state.

- Maintained as part of the content transmitted from the client to the server and back to the client.
  I. e. the client manages the application state.
- Thus, any server can potentially resume the transaction at the point where it was interrupted.

2

## Resource State

- The resource state is the type of state that the *S* in *REST* refers to.
- The *stateless* restriction means that all messages must contain the entire application state (i. e. we effectively have no sessions).

# Multiple representations

- Most resources only have a single representation.
- REST can support any media type; JSON is the standard.

  (HTTP supports content negotiation.)
- Links can be embedded and reflect the structure with which a user can navigate through an application.

# Simple/first tests for RESTfulness

- Can I use a GET to retrieve the URLs I have POSTed to?
- Would the client notice if the server...
  - is restarted at any point between requests
  - is reinitialized when the client makes the next request.

# Resource modelling

- organize the application into URI-addressable resources (discrete resources should have their own stable URIs).
- use only the standard HTTP messages - GET, PUT, POST, DELETE and PATCH - to provide the full capabilities of the application

## HTTP methods

**GET** is used to query resources.

**PUT** is used to create a resource or update it if you know the URI.

**POST** is used to create a new resource. The response should then contain the URI of the created resource.

**DELETE** deletes the specified resource.

The difference between **PUT** and **POST** is that **PUT** is idempotent: a single or repeated calls have the same effect (i. e. a repeated call has no side effect), while successive identical **POST** calls can have additional effects, such as the repeated transfer of an order/the repeated creation of a message.

A **PATCH** request is regarded as a set of instructions for changing a resource. In contrast, a PUT request is a complete representation of a resource.

# Example Application del.icio.us

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

**del.icio.us enables us:**

- to get a list of all our bookmarks and filter this list by tags or date and to limit the number of retrieved bookmarks
- to retrieve the number of bookmarks created on different days
- to retrieve when we last updated our bookmarks
- to retrieve a list of all our markers
- to add a bookmark
- to edit a bookmark
- to delete a bookmark
- to rename a bookmark

# Example Application del.icio.us: Resources

**Bookmarks:** *http://del.icio.us/api/[username]/bookmarks*
**Tags:** *http://del.icio.us/api/[username]/tags*
**[username]:** is the username of the user whose bookmarks we are interested in

# Example Application del.icio.us: Repräsentation von Ressourcen

We define (in this example) some XML document formats and media types to identify them:

| Mediatype | Description |
|---|---|
| delicious/bookmarks+xml | list of bookmarks |
| delicious/bookmark+xml | one bookmark |
| delicious/bookmarkcount+xml | number of bookmarks per tag |
| delicious/update+xml | time at which the bookmarks were last updated |
| delicious/tags+xml | list of tags |
| delicious/tag+xml | a tag |

# Example Application del.icio.us: Query Bookmarks

| | |
|---|---|
| **URL:** | *http://del.icio.us/api/[username]/bookmarks/* |
| **Method:** | GET |
| **Querystring:** | tag = Filter by tag |
| | dt = Filter by date |
| | start = The number of the first returned bookmark |
| | end = The number of the last returned bookmark |
| **Return value:** | 200 OK & XML (delicious/bookmarks+xml) |
| | 401 Unauthorized |
| | 404 Not Found |

# Example application del.icio.us: Query bookmarks - example response

```
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>
```

## Example application del.icio.us: Information about a bookmark

| | |
|---|---|
| **URL:** | *http://del.icio.us/api/[username]/bookmarks/[hash]`* |
| **Method:** | *GET* |
| **Return value:** | 200 OK & XML (delicious/bookmark+xml) |
| | 401 Unauthorized |
| | 404 Not Found |

# Example application del.icio.us: Information about a bookmark - Example response

```
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>
```

## Example application del.icio.us: Query the number of bookmarks

| | |
|---|---|
| **URL:** | *http://del.icio.us/api/[username]/bookmarks/count* |
| **Method:** | GET |
| **Query parameter:** | tag = Filter by tag |
| **Return value:** | 200 OK & XML (delicious/bookmark+xml) |
| | 401 Unauthorized |
| | 404 Not Found |

## Example application del.icio.us: Query when the last change was made

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

# Example application del.icio.us: Adding a bookmark

| | |
|---|---|
| **URL:** | *http://del.icio.us/api/[username]/bookmarks/`* |
| **Method:** | POST |
| **Query document:** | XML (delicious/bookmark+xml) |
| **Return value:** | 201 Created & Location |
| | 401 Unauthorized |
| | 415 Unsupported Media Type(if the send document is not valid) |

# Example application del.icio.us: Adding a bookmark - example document

```
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>
```

# Example application del.icio.us: Update a bookmark

| | |
|---|---|
| **URL:** | *http://del.icio.us/api/[username]/bookmarks/[hash]`* |
| **Method:** | PUT |
| **Query document:** | XML (delicious/bookmark+xml) |
| **Return value:** | 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) |

## Example application del.icio.us: Delete a bookmark

| | |
|---|---|
| **URL:** | *http://del.icio.us/api/[username]/bookmarks/[hash]* |
| **Method:** | DELETE |
| **Return value:** | 204 No Content |
| | 401 Unauthorized |
| | 404 Not Found |