

AMSTERDAM

INTERNATIONAL
SOFTWARE DEVELOPMENT

CONFERENCE 2012



goto;

conference

Conference: May 24-25 / Training: May 26



REST: From GET to HATEOAS

... or how to create RESTful APIs



Who am I?

~ just some java guy ~



Jos Dirksen

Architect @ JPoint

- Live in Waalwijk
- Married
- Daughter (2.5 y/o)
- Blog at:
www.smartjava.org



Interests

- Java & Scala
- REST, WS-*
- HTML5
- Snowboarding
- Reading
- Cooking



Books

Shameless self promotion:

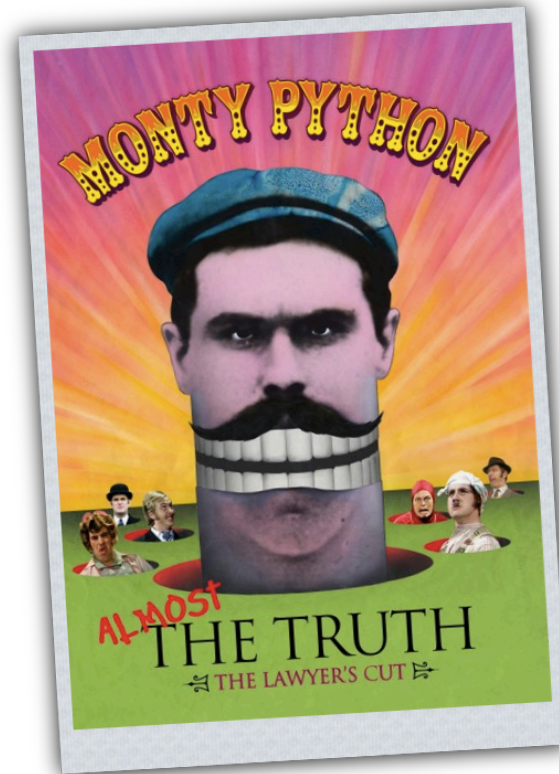
- SOA Governance in Action, Manning, 2012
- Open Source ESBs in Action, Manning, 2008

Disclaimer

~ you will encounter opinionated content ~

Heavily opinionated

- There are many truths
- This is mine
- I'm not a Restafarian



In the beginning...

~ It was a dark place ~



The world before **REST!**



Many different 'standards':

RMI, **SOAP**, Corba, DCE, DCOM

From many different parties:

Sun, Microsoft, IBM, OASIS, OMG

Caused many problems:

- Bad interoperability.
- Reinvent the wheel.
- Vendor 'lock-in'.

And then came REST!

“**R**epresentational **S**tate **T**ransfer (**REST**) is a style of software architecture for distributed hypermedia systems such as the World Wide Web”

REST is based on a set of constraints

~ Rest 101 ~

1. Client-server

Separate clients and servers.

2. Stateless server

Each request from a client contains all the information necessary to service the request.

3. Cacheable

Clients can cache responses, responses must indicate if this is allowed.

4. Uniform interface

There is a uniform interface between clients and servers.

5. Layered System

Must allow concepts such as load balancers, proxies and firewalls.

6. Code-On-Demand (optional)

Client can request code from server and execute it.

Constraint 4: **Uniform** interface

~ Rest 101 ~

- A. **Identification of resources:**
E.g. by using an URI.
- B. **Manipulation of resources through representations:**
A representations allows user to modify/delete resource .
- C. **Self-descriptive messages:**
Process message based on message and meta-data.
- D. **Hypermedia as the engine of application state:**
State transitions are defined in representations.



And all was good!



Why do this? Why be RESTful?

- Scalable
- Fault-tolerant
- Recoverable
- Secure
- Loosely coupled

“Exactly what we want in the applications we are developing!”

But not everybody understood...

- GET: /getAllDogs
- GET: /saveDog?name=brian&age=7
- GET: /feedDog?food=123&dog=brian

instead of:

- GET: /dogs
- PUT: /dog/brian
- POST: /dog/brian/food/123



“In your URLs – nouns are good; verbs are (usually) bad”

Twitter API

~ just saying your RESTful doesn't make it so ~

Bad URLs:

- `POST statuses/destroy/:id`
- `GET statuses/show/:id`
- `POST direct_messages/new`

Instead of:

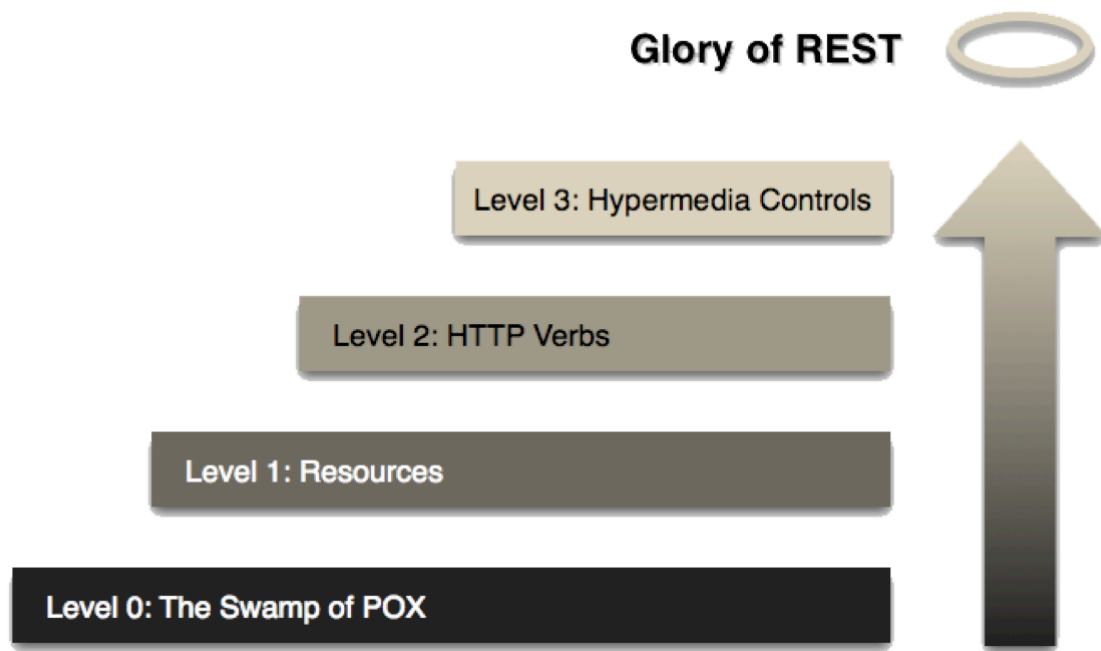
- `DELETE status/:id`
- `GET status/:id`
- `POST direct_message` or `PUT direct_message/:id`



The maturity levels of **REST**



Richardson's Maturity Model



Level 0: The Swamp of Pox

~ nothing to do with REST ~

- One **URI**, one **HTTP** method
- **XML-RPC** / **SOAP** / **POX**
- Giant 'black box', is what eBay uses.

```
POST /appointmentService HTTP/1.1  
[various other headers]
```

```
<appointmentRequest>  
  <slot doctor = "mjones" start = "1400" end = "1450"/>  
  <patient id = "jsmith"/>  
</appointmentRequest>
```

Level 0: eBay

~ not an easy to use API ~

```
POST http://svcs.ebay.com/services/search/FindingService/v1
```

```
<findItemsByKeywordsRequest xmlns="http://www.ebay.com/marketplace/
search/v1/services">
  <affiliate>
    <networkId>9</networkId>
    <trackingId>1234567890</trackingId>
    <customId>k-man</customId>
  </affiliate>
  <sortOrder>EndTime</sortOrder>
  <paginationInput>
    <entriesPerPage>2</entriesPerPage>
  </paginationInput>
  <keywords>camalots</keywords>
</findItemsByKeywordsRequest>
```

Level 1: Resources

~ lots of APIs start out this way ~

- Each resource has an **unique URI**
- Single HTTP verb (usually POST or GET)
- Verbs have no meaning, used to **tunnel over HTTP**
- Early versions of Flickr, del.icio.us and Amazon

```
POST /slots/1234 HTTP/1.1  
[various other headers]
```

```
<appointmentRequest>  
  <patient id = "jsmith"/>  
</appointmentRequest>
```

Level 2: HTTP Verbs

~ close but no sigar ~

- Many URIs, using multiple verbs
- Correct use of response codes
- Exposes state, not behavior
- Crud services, can be useful e.g Amazon S3

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```

```
HTTP/1.1 200 OK
<openSlotList>
  <slot id = "1234" start = "1400" end = "1450"/>
  <slot id = "5678" start = "1600" end = "1650"/>
</openSlotList>
```

Level 3: Hypermedia controls

~ True RESTful ~

- Resources are **self-describing**
- **Hypermedia As The Engine Of Application State (HATEOAS)**
- Exposes **state** and **behavior**

```
<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400"
    end = "1450"/>
  <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/addTest"
    uri = "/slots/1234/appointment/tests"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
    uri = "/patients/jsmith/contactInfo"/>
</appointment>
```


So, are level 0, 1 and 2 RESTful?

“What needs to be done to make the **REST** architectural style clear on the notion that **hypertext is a constraint**? In other words, if the engine of application state (and hence the API) **is not being driven by hypertext**, then **it cannot be RESTful** and cannot be a REST API. **Period.** Is there some broken manual somewhere that needs to be fixed?”

Roy T. Fielding

Level 2 is easy, how do we do **HATEOAS**?

~ Worst acronym ever! ~



HATEOAS?

“The next control state of an application resides in the representation of the first requested resource, ... The application state is controlled and stored by the user agent ... anticipate changes to that state (e.g., link maps and prefetching of representations) ... The model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions in the current set of representations.”

Roy T. Fielding

Say what?



The key to **HATEOAS** is simple

- **Hypermedia / Mime-types / Media-types :**
 - Describes a **current state**
 - Compare it with a web page
 - Can be seen as the contract
- **Links:**
 - Describe the transition to the **next state**
 - Compare it with hyperlinks
- **HATEOAS makes **surfing** the web possible**
- **Jim Webber: “Hypermedia Describes Protocols” (**HYDEPR**)**



“In each response message, include the links for the next request message”

HATEOAS Part 1: Links

~ AtomPub ~

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Example Feed</title>
  <subtitle>A subtitle.</subtitle>
  <link href="http://example.org/feed/"
        rel="self" />
  <link href="http://example.org/" />
  <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>John Doe</name>
    <email>johndoe@example.com</email>
  </author>
  <entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2003/12/13/atom03" />
    <link rel="alternate" type="text/html"
          href="http://example.org/2003/12/13/atom03.html"/>
```

HATEOAS Part 1: Links

~ Netflix API ~

```
<link
  href="http://.../catalog/titles/series/70023522/cast"
  rel="http://schemas.netflix.com/catalog/people"
  title="cast">
<cast>
  <link href="http://api.netflix.com/catalog/people/30011713"
    rel="http://schemas.netflix.com/catalog/person"
    title="Steve Carell"/>
  <link href="http://api.netflix.com/catalog/people/30014922"
    rel="http://schemas.netflix.com/catalog/person"
    title="John Krasinski"/>
  <link href="http://api.netflix.com/catalog/people/20047634"
    rel="http://schemas.netflix.com/catalog/person"
    title="Jenna Fischer"/>
</cast>
</link>
```


HATEOAS Part 1: Twitter Example

~ Before using links ~

```
GET .../followers/ids.json?cursor=-1&screen_name=josdirksen

{
  "previous_cursor": 0,
  "previous_cursor_str": 0,
  "ids": [
    12345678,
    87654321,
    11223344
  ],
  "next_cursor": 0,
  "next_cursor_str": "0"
}
```

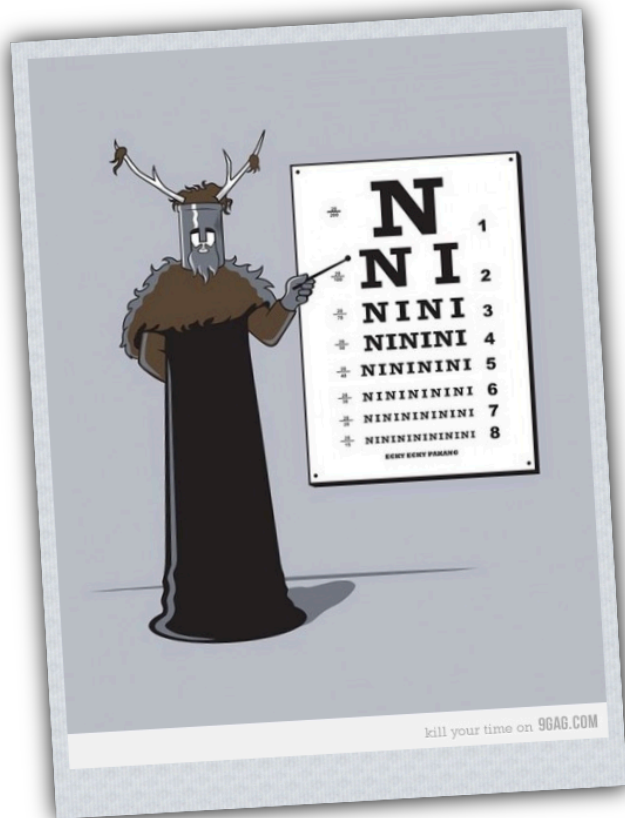
HATEOAS Part 1: Twitter Example

~ With links ~

```
GET .../followers/ids.json?cursor=-1&screen_name=josdirksen

{
  "previous_cursor": 0,
  "id": {
    "name": "John Smit",
    "id": "12345678"
    "links" : [
      { "rel": "User info",
        "href": "https://.../user/12345678"},
      { "rel": "Follow user",
        "href": "https://.../friendship/12345678"}
    ]
  } // and add other links: tweet to, send direct message,
  ..// block, report for spam, add or remove from list
}
```

HATEOAS Part 2: Media-types



WHERE:

People know **where** to find your resource using **URIs**.

HOW:

They know **how** to interact by following **links**.

WHAT:

But **what** do the resources look like, **what** do they need to post?

~ not all media-types are equal, some are more equal than others ~

- 



HATEOAS Part 2: Media-types

~ Custom media types ~

200 OK

Content-Type: **application/vnd.bank.org.account+xml**; charset=UTF-8

```
<accounts xmlns="urn:org:bank:accounts">
  <account>
    <id>AZA12093</id>
    <link href="http://bank.org/account/AZA12093" rel="self"/>
    <link rel="http://bank.org/rel/transfer"
          type="application/vnd.bank.org.transfer+xml"
          href="http://bank.org/transfers"/>
    <link rel="http://bank.org/rel/customer"
          type="application/vnd.bank.org.customer+xml"
          href="http://bank.org/customer/7t676323a"/>
    <balance currency="USD">993.95</balance>
  </account>
</accounts>
```

And that is what **HATEOAS** means!

- **Media-types** describe the resources.
- Actions are executed by following **links**.
- Each new response reflects a **state**.
- It is good to create **custom** media-types.
- Creates self-describing **APIs**.
- Clients '**explore**' your API just as they browse the web.

“Media-types describes a domain specific application protocol”



HATEOAS Part 2: Twitter Example

~ With links, but no media-type ~

```
GET .../followers/ids.json?cursor=-1&screen_name=josdirksen

{
  "previous_cursor": 0,
  "id": {
    "name": "John Smit",
    "id": "12345678"
    "links" : [
      { "rel": "User info",
        "href": "https://.../user/12345678"} ,
      { "rel": "Follow user",
        "href": "https://.../friendship/12345678"}
    ]
  }
  ...
}
```


HATEOAS Part 2: Twitter Example

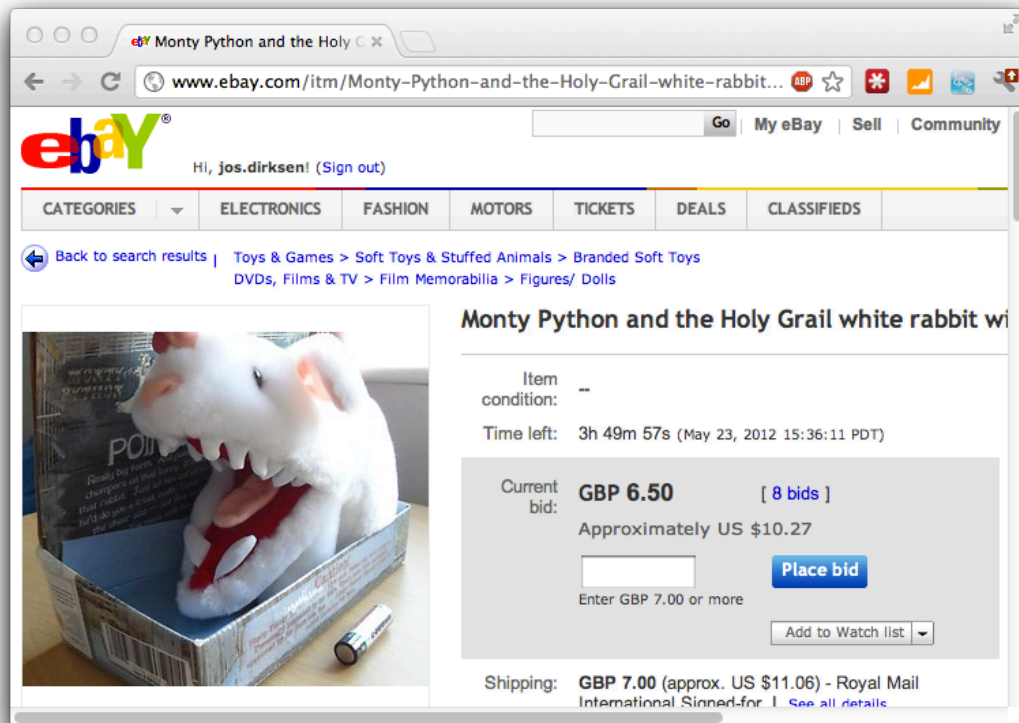
~ With links & media-type ~

```
GET .../followers/ids.json?cursor=-1&screen_name=josdirksen
```

```
{
  "previous_cursor": 0,
  "id": {
    "name": "John Smit",
    "id": "12345678"
    "links" : [
      { "type": "application/vnd.twitter.com.user",
        "rel": "User info",
        "href": "https://.../user/12345678"}},
      { "type": "application/vnd.twitter.com.user.follow",
        "rel": "Follow user",
        "href": "https://.../friendship/12345678"}
    ] // and add other options: tweet to, send direct message,
      // block, report for spam, add or remove from list
  } // This is how you create a self-describing API.
}
```

Case: eBay

~ API should guide the user ~



Common scenario: bidding on item

1. **Add item to watch list:** keep track of the item.
2. **Get user details:** find out more about the buyer.
3. **Get user feedback:** is seller trustworthy?
4. **Make a bid:** place a bid for the item.

eBay: API should help the client

~ just like a browser does using links ~

Hi, jos.dirksen! (Sign out)

CATEGORIES **ELECTRONICS** **FASHION** **MOTORS** **TICKETS** **DEALS** **CLASSIFIEDS** **eBay Buyer Protection** **Add to watch list**

Back to search results | Toys & Games > Soft Toys & Stuffed Animals > Branded Soft Toys
DVDs, Films & TV > Film Memorabilia > Figures/ Dolls

Monty Python and the Holy Grail white rabbit with big pointy teeth soft toy

Item condition: --

Time left: 3h 49m 57s (May 23, 2012 15:36:11 PDT)

Current bid: **GBP 6.50** [8 bids]
Approximately US \$10.27

Place bid

Enter GBP 7.00 or more **Add to Watch list**

Shipping: **GBP 7.00** (approx. US \$11.06) - Royal Mail International Signed-for | [See all details](#)
See details about International shipping here. [?](#)
Item location: **Cornwall, UK, United Kingdom**
Ships to: **N. and S. America, Europe, Asia, Australia**
[See exclusions](#)

Delivery: Estimated Delivery within 4-8 business days [?](#)
Seller ships within 1 day after [receiving cleared payment](#).

Share: [Facebook](#) [Twitter](#) [Pinterest](#) | [Add to Watch list](#)

Seller information
dangermouse_rm (2063 ★) **me**
100% Positive feedback
.....
[Save this seller](#)
[See other items](#)

Place bid **Get User details** **Look at user feedback**

Click to view larger image and other views

eBay: add to watchlist

~ API should tell us what to do ~

```
GET .../item/180881974947
{
  "name" : "Monty Python and the Holy Grail white rabbit big pointy teeth",
  "id" : "180881974947",
  "start-price" : "6.50",
  "currency" : "GBP",
  ...
  "links" : [
    { "type": "application/vnd.ebay.item",
      "rel": "Add item to watchlist",
      "href": "https://.../user/12345678/watchlist/180881974947"},
    {
      // and a whole lot of other operations
    }
  ]
}
```

eBay: get user details

~ Follow links to get more information ~

```
GET .../item/180881974947
{
  "name" : "Monty Python and the Holy Grail white rabbit big pointy teeth",
  "id" : "180881974947",
  "start-price" : "6.50",
  "currency" : "GBP",
  // whole lot of other general item data
  "bidder" : {
    "name" : "dangermouse_rm",
    "link" : {
      "type" : "application/vnd.ebay.user",
      "rel" : "Get user details",
      "href" : "https://.../user/314512346523"
    }
  }
}
```

eBay: get user feedback

~ Depending on rights we get different links ~

```
GET .../user/314512346523
{
  "name" : "dangermouse_rm",
  "location" : "united kingdom",
  // whole lot of other general user data
  "feedbacks" : {
    "last_month" : "7,0,0"
    "last_6_month" : "19,0,0",
    "feedback" : {
      "title" : "Great communication and quick delivery. +++"
      "link" : {
        "type" : "application/vnd.ebay.feedback",
        "rel" : "Show feedback",
        "href" : "https://.../user/314512346523/feedback/1234567"
      }
    }
  }
  ...
}
```

eBay: place bid

~ Depending on rights we get different links ~

```
GET .../item/180881974947
{
  "name" : "Monty Python and the Holy Grail white rabbit big pointy teeth",
  "id" : "180881974947",
  "start-price" : "6.50",
  "currency" : "GBP",
  ...
  "links" : [
    { "type": "application/vnd.ebay.bid",
      "rel": "Place bid",
      "href": "https://.../user/12345678/bid/180881974947"},
    {
      // and a whole lot of other operations
    }
  ]
}
```

eBay: much easier!

~ and there was much rejoicing ~



Versioning without breaking



Three options for versioning

~ Media-types ~

- Media-type is versioned, directly in its **name**

Request:

GET /opengov/garbageschedule?location=Main%20Street HTTP/1.1

Accept: application/vnd.opengov.org.garbageschedule-v2+json

Response:

HTTP/1.1 200 OK

Content-Type: application/vnd.opengov.org.garbageschedule-v2+json

```
{"schedule":  
  "self": "schedule-2423",  
  "dayOfWeek": "Monday",  
  "oddOrEvenWeeks": "Odd"}
```

Three options for versioning

~ Add version qualifier ~

- Media-type stays the same, add a **qualifier**.

Request:

GET /opengov/garbageschedule?location=Main%20Street HTTP/1.1

Accept: **application/vnd.opengov.org.garbageschedule+json;v=1**

Response:

HTTP/1.1 200 OK

Content-Type: **application/vnd.opengov.org.garbageschedule+json;v=1**

```
{"schedule"  
  "self": "schedule-2423",  
  "dayOfWeek": "Monday",  
  "oddOrEvenWeeks": "Odd"}
```

Three options for versioning

~ The URI is versioned ~

- The version is added in the **URI** path

Request:

```
GET /opengov/v1/garbageschedule?location=Main%20Street HTTP/1.1  
Accept: application/vnd.opengov.org.garbageschedule+json
```

Response:

```
HTTP/1.1 200 OK  
Content-Type: application/vnd.opengov.org.garbageschedule+json
```

```
{"schedule"  
  "self": "schedule-2423",  
  "dayOfWeek": "Monday",  
  "oddOrEvenWeeks": "Odd"}
```

Which one is the best?

~ personal opinion ~



Matter of taste:

1. **Media-type** approach most **RESTful**, but requires work on client and server side.
2. **Qualifier**, second best, easier to implement. Less media-types to keep track off.
3. **URI**, most often used. Personally don't like it. Can have two URIs point to same resource.

When HTTPS isn't enough



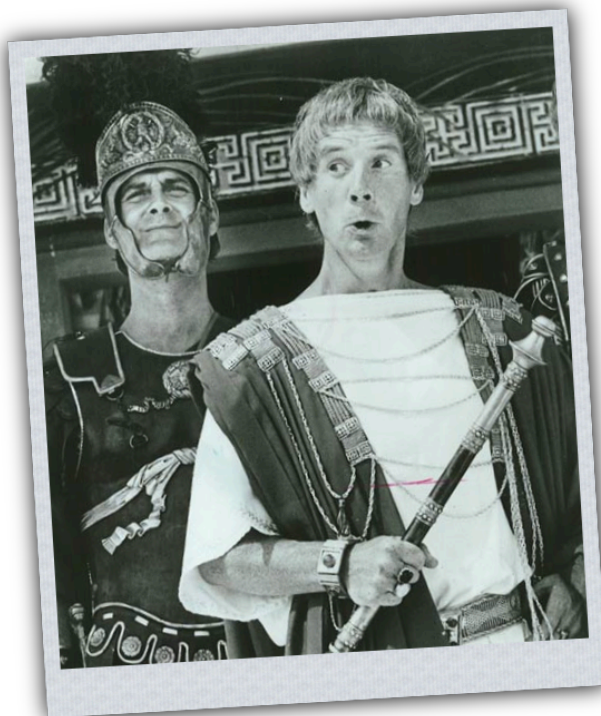
General idea about authentication in REST

“**REST** means working with the **standards of the web**, and the standard for 'secure' transfer on the web is **SSL**. Anything else is going to be kind of **funky** and require extra deployment effort for clients, which will have to have encryption libraries available.”

Highest rated answer on stackoverflow
regarding REST authentication schemes

Why do we need more?

- **HTTPS** doesn't fix man-in-the-middle attack
- It provides only **transport level** security
- Has no support for message **integrity** or **authenticity**
- REST assumes "**Layered System**"
- **OAuth** is nice (and complex) for authentication doesn't handle message integrity.
- REST doesn't have a **WS-Security** standard



HMAC based authentication

~ de facto standard ~

- Used by Google, Amazon AWS, Yahoo etc.
- Create a signature of the complete request, using shared secret.
- Add custom header with signature and signing user.
- Encrypt the following with this shared secret:
 - **URI**, to avoid executing on different resource,
 - **Verb**, indicates what we want to do with a resource,
 - **MD5-Header**, to detect changes to the content body
 - **Content-type**, indicates the type of resource
 - **Date header**, to avoid replay

HMAC based authentication

~ example ~

```
POST /resources/rest/geo/comment HTTP/1.1[\r][\n]
hmac: jos:+9tn0CLfxXFbzPmbYwq/KYuUSUI=[\r][\n]
Date: Mon, 26 Mar 2012 21:34:33 CEST[\r][\n]
Content-Md5: r52FDQv6V2GHN4neZBvXLQ==[\r][\n]
Content-Length: 69[\r][\n]
Content-Type: application/vnd.geo.comment+json;
charset=UTF-8[\r][\n]
Host: localhost:9000[\r][\n]
Connection: Keep-Alive[\r][\n]
User-Agent: Apache-HttpClient/4.1.3 (java 1.5)[\r][\n]
[\r][\n]
{"comment" : {"message":"blaat" , "from":"blaat" , "commentFor":
123}}
```

Parting guidelines

~ you may forget everything else I said ~

- Just doing URLs and Verbs doesn't make it RESTful
 - But that isn't necessary a bad thing.
- Use “**Links**” to describe “**HOW**” your service is used.
- Describe “**WHAT**” is expected using **media-types**.
 - This isn't a complete replacement of documentation
 - Don't use WADL
- Use **media-types** for versioning.
- Forget what **HATEOAS** stands for.

Q&A



AMSTERDAM

INTERNATIONAL
SOFTWARE DEVELOPMENT

CONFERENCE 2012



goto;

conference

Conference: May 24-25 / Training: May 26



THANKS FOR COMING

More information; look at <http://www.jpoint.nl> or contact me.