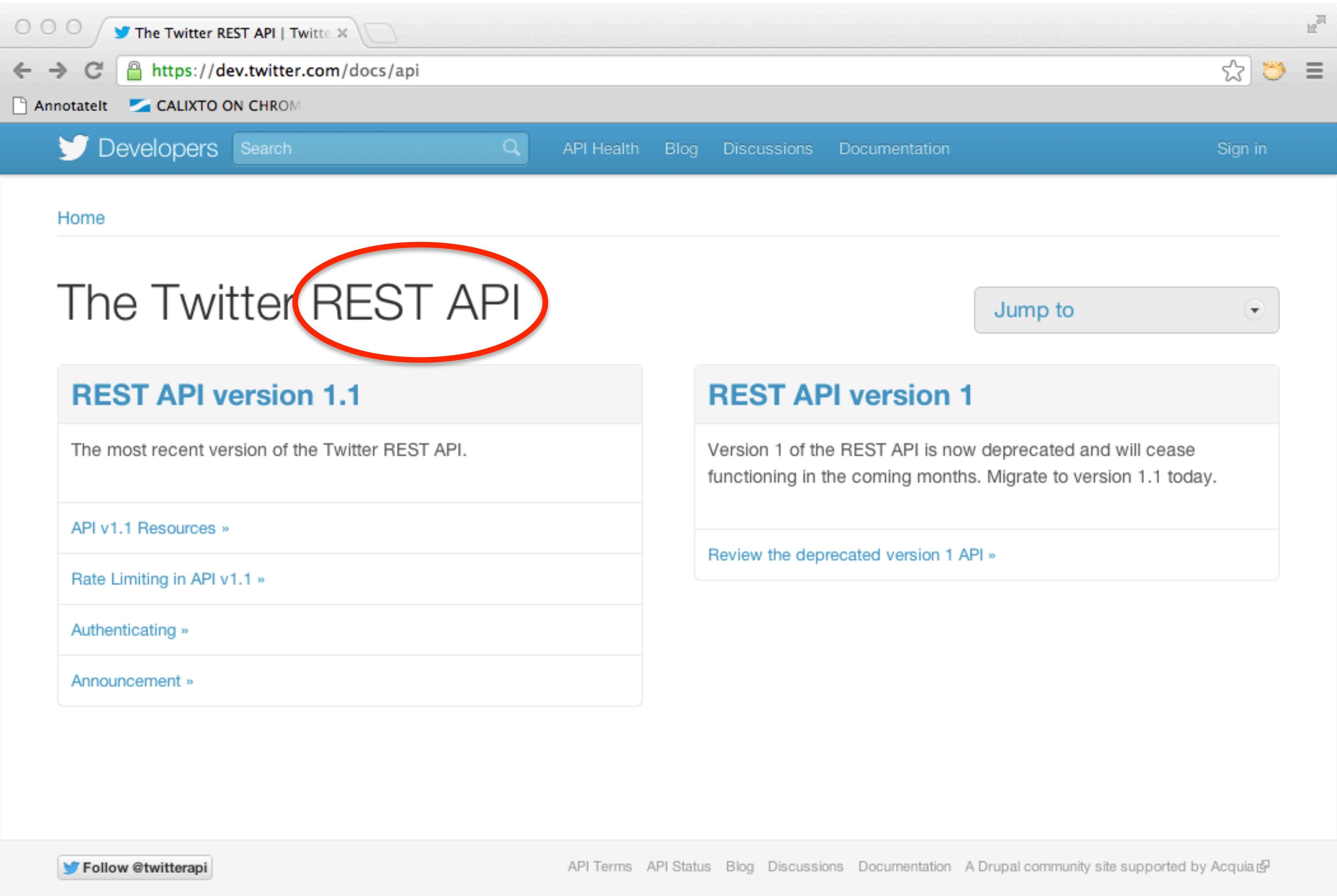
INFO/CS 4302
Web Information Systems

FT 2012
Week 7: RESTful Webservice APIs

- Bernhard Haslhofer -



The Twitter REST API | Twitter

https://dev.twitter.com/docs/api

AnnotateIt CALIXTO ON CHROM

Developers Search API Health Blog Discussions Documentation Sign in

Home

The Twitter REST API

Jump to

REST API version 1.1

The most recent version of the Twitter REST API.

API v1.1 Resources »

Rate Limiting in API v1.1 »

Authenticating »

Announcement »

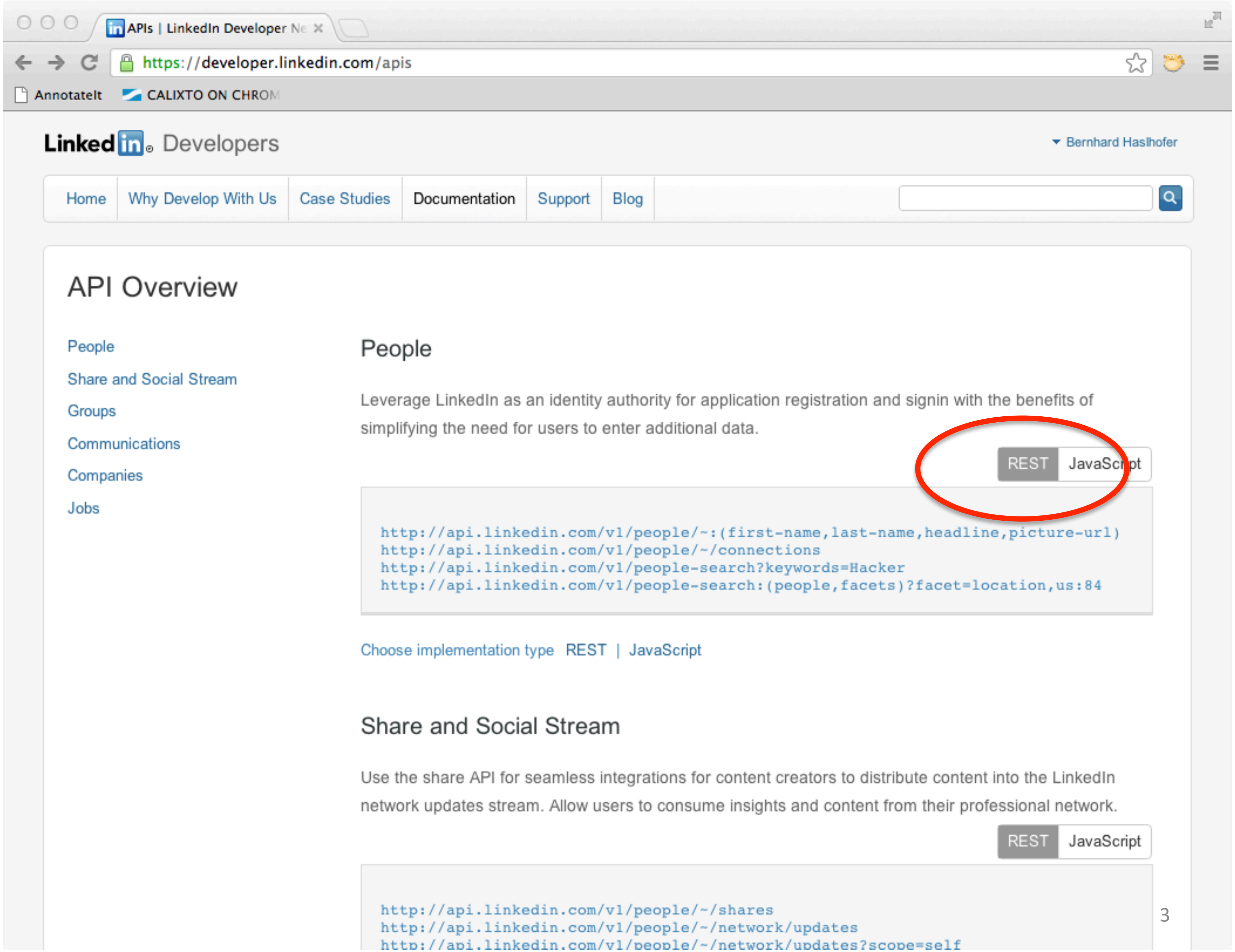
REST API version 1

Version 1 of the REST API is now deprecated and will cease functioning in the coming months. Migrate to version 1.1 today.

Review the deprecated version 1 API »

Follow @twitterapi

API Terms API Status Blog Discussions Documentation A Drupal community site supported by Acquia



API Overview

People
Share and Social Stream
Groups
Communications
Companies
Jobs

People

Leverage LinkedIn as an identity authority for application registration and signin with the benefits of simplifying the need for users to enter additional data.

REST JavaScript

```
http://api.linkedin.com/v1/people/~:(first-name,last-name,headline,picture-url)
http://api.linkedin.com/v1/people/~/connections
http://api.linkedin.com/v1/people-search?keywords=Hacker
http://api.linkedin.com/v1/people-search:(people,facets)?facet=location,us:84
```

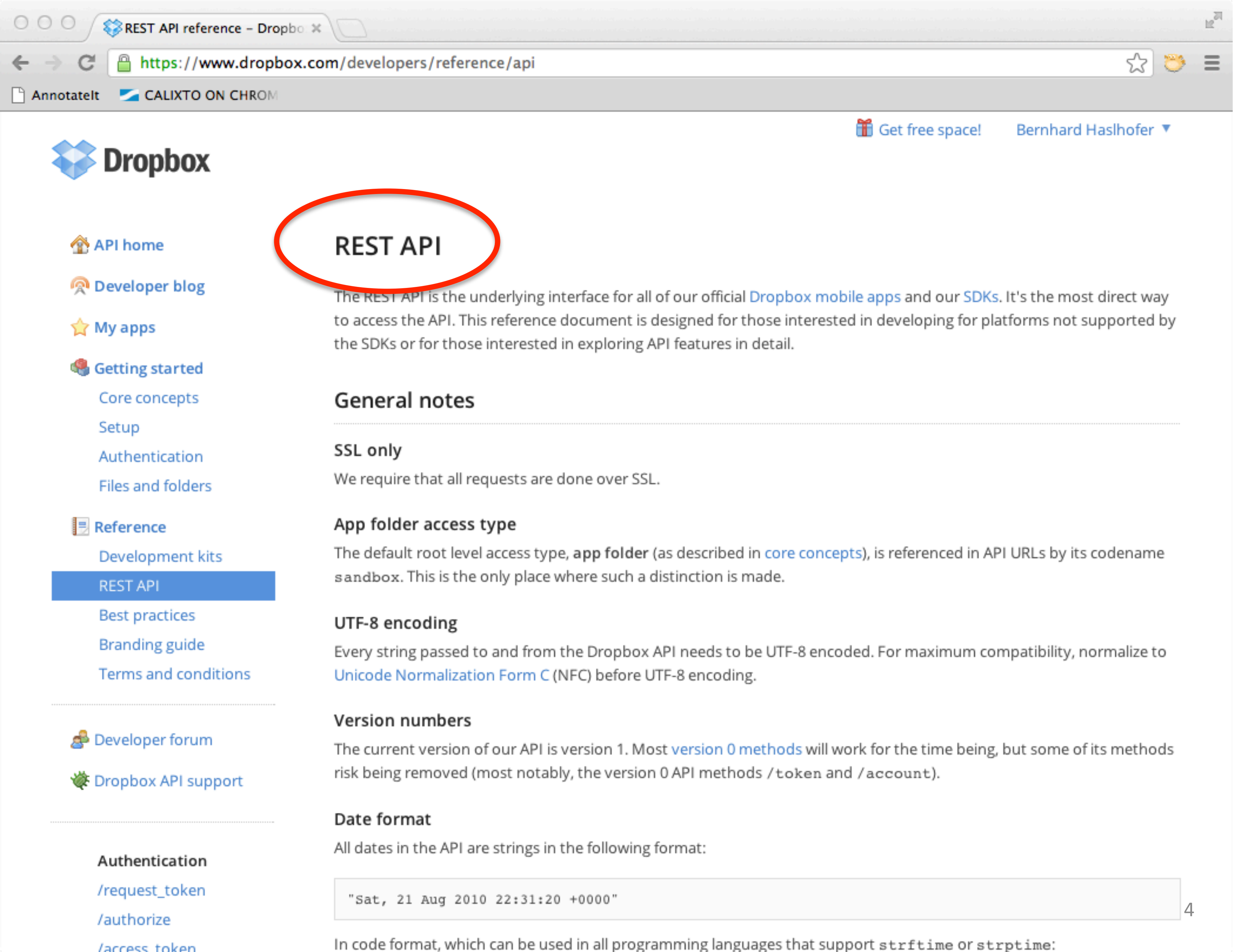
Choose implementation type REST | JavaScript

Share and Social Stream

Use the share API for seamless integrations for content creators to distribute content into the LinkedIn network updates stream. Allow users to consume insights and content from their professional network.

REST JavaScript

```
http://api.linkedin.com/v1/people/~/shares
http://api.linkedin.com/v1/people/~/network/updates
http://api.linkedin.com/v1/people/~/network/updates?scope=self
```



REST API

The REST API is the underlying interface for all of our official Dropbox mobile apps and our SDKs. It's the most direct way to access the API. This reference document is designed for those interested in developing for platforms not supported by the SDKs or for those interested in exploring API features in detail.

General notes

SSL only

We require that all requests are done over SSL.

App folder access type

The default root level access type, **app folder** (as described in core concepts), is referenced in API URLs by its codename `sandbox`. This is the only place where such a distinction is made.

UTF-8 encoding

Every string passed to and from the Dropbox API needs to be UTF-8 encoded. For maximum compatibility, normalize to Unicode Normalization Form C (NFC) before UTF-8 encoding.

Version numbers

The current version of our API is version 1. Most version 0 methods will work for the time being, but some of its methods risk being removed (most notably, the version 0 API methods `/token` and `/account`).

Date format

All dates in the API are strings in the following format:

```
"Sat, 21 Aug 2010 22:31:20 +0000"
```

In code format, which can be used in all programming languages that support `strftime` or `strptime`:

Source: http://www.blogperfume.com/new-27-circular-social-media-icons-in-3-sizes/

Plan for today...

- Recap - Web Fundamentals
- APIs, Web Services
- Group Brainstorming
- RESTful APIs – Architectural principles
- Questions, Housekeeping, ...

RECAP – WEB FUNDAMENTALS

Web Fundamentals

Internet ≠ World Wide Web

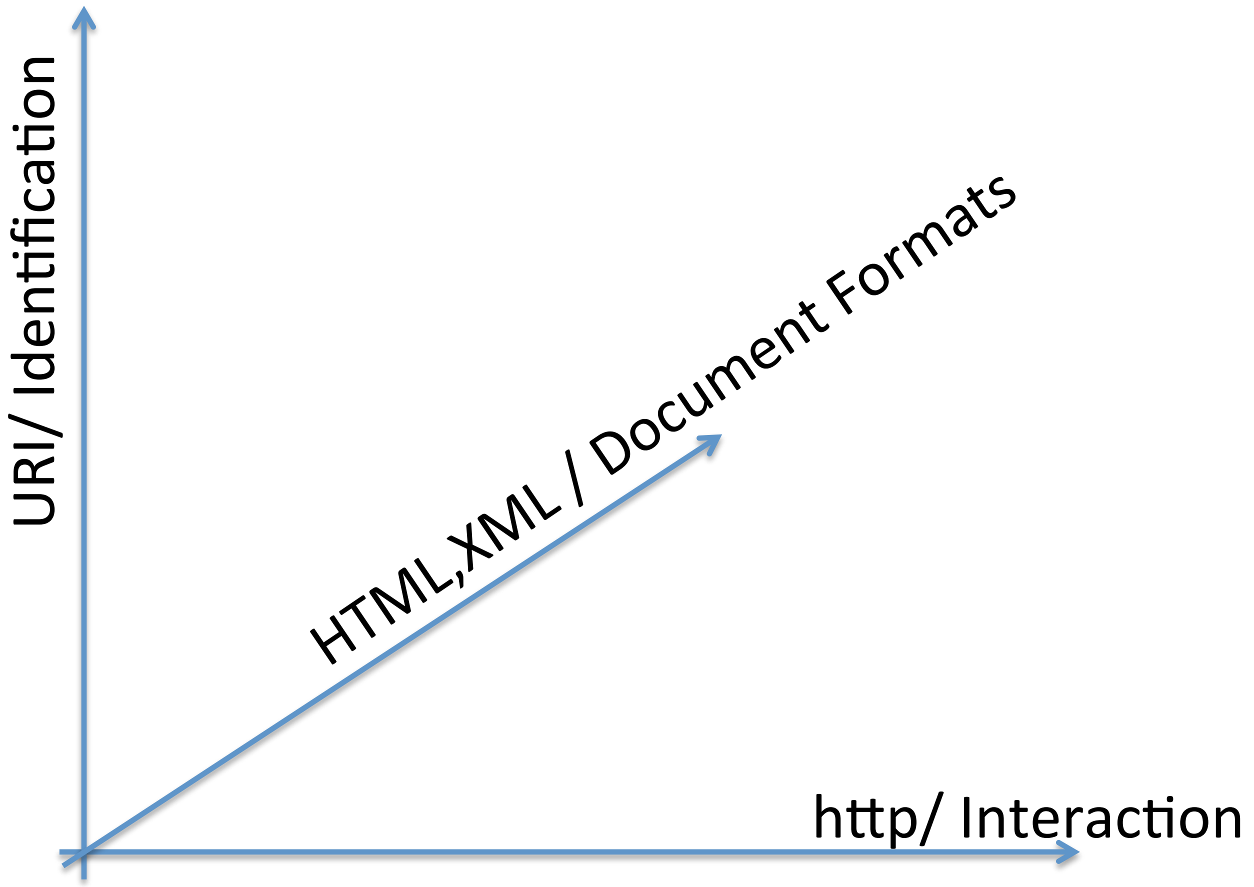
Web Fundamentals

- Key Architectural Components
  - Identification: ???
  - Interaction: ???
  - Standardized Document Formats: ???, ???, ???

Web Fundamentals

- Key Architectural Components
  - Identification: URI
  - Interaction: HTTP
  - Standardized Document Formats: HTML, XML, JSON, etc.

Principle 'Orthogonal Specifications'

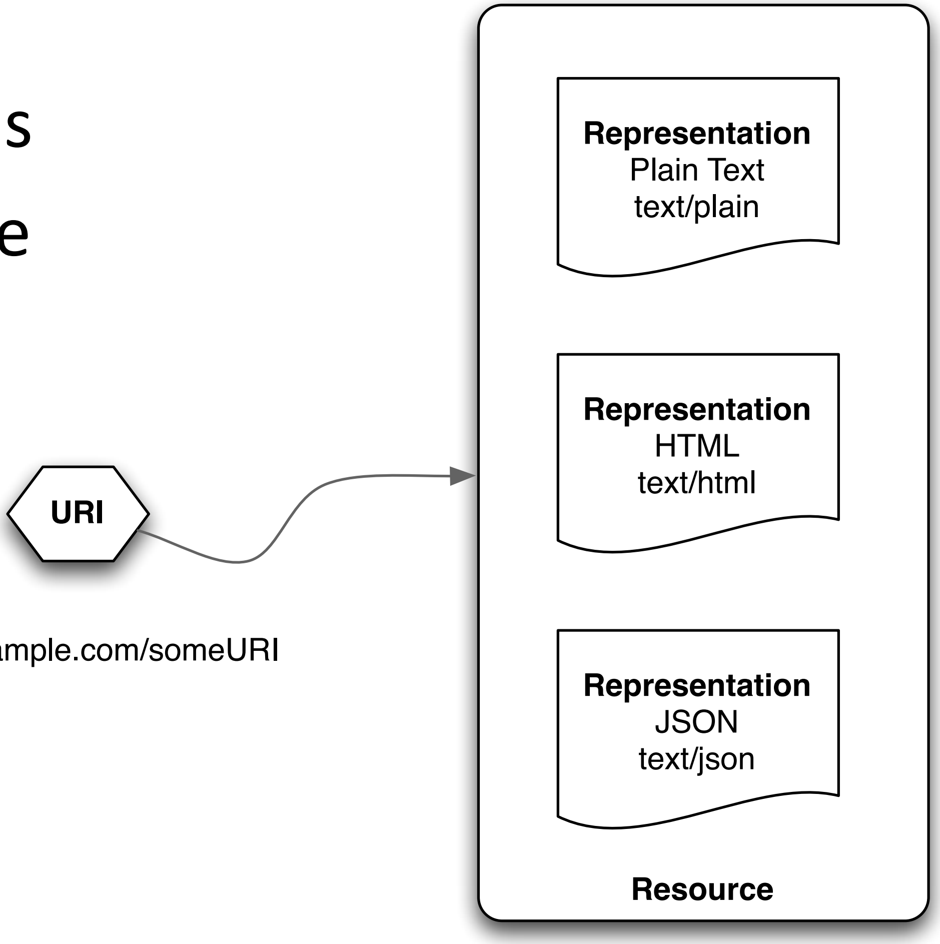


URIs / Resources

- URIs identify interesting things
  - documents on the Web
  - relevant aspects of a data set
- HTTP URIs name and address resources in Web-based systems
  - a URI names and identifies one resource
  - a resource can have more than one name
    - http://foo.com/software/latest
    - http://foo.com/software/v1.4

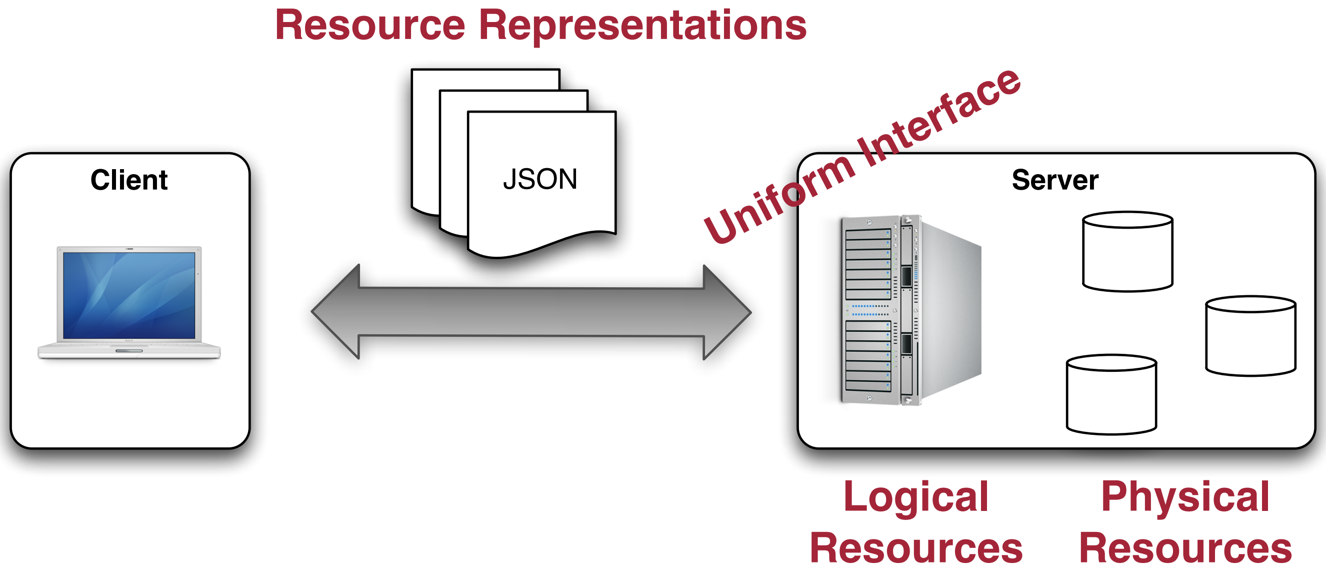
Resource Representation

- A resource can have several representations
- Representations can be in any format
  - HTML
  - XML
  - JSON
  - ...

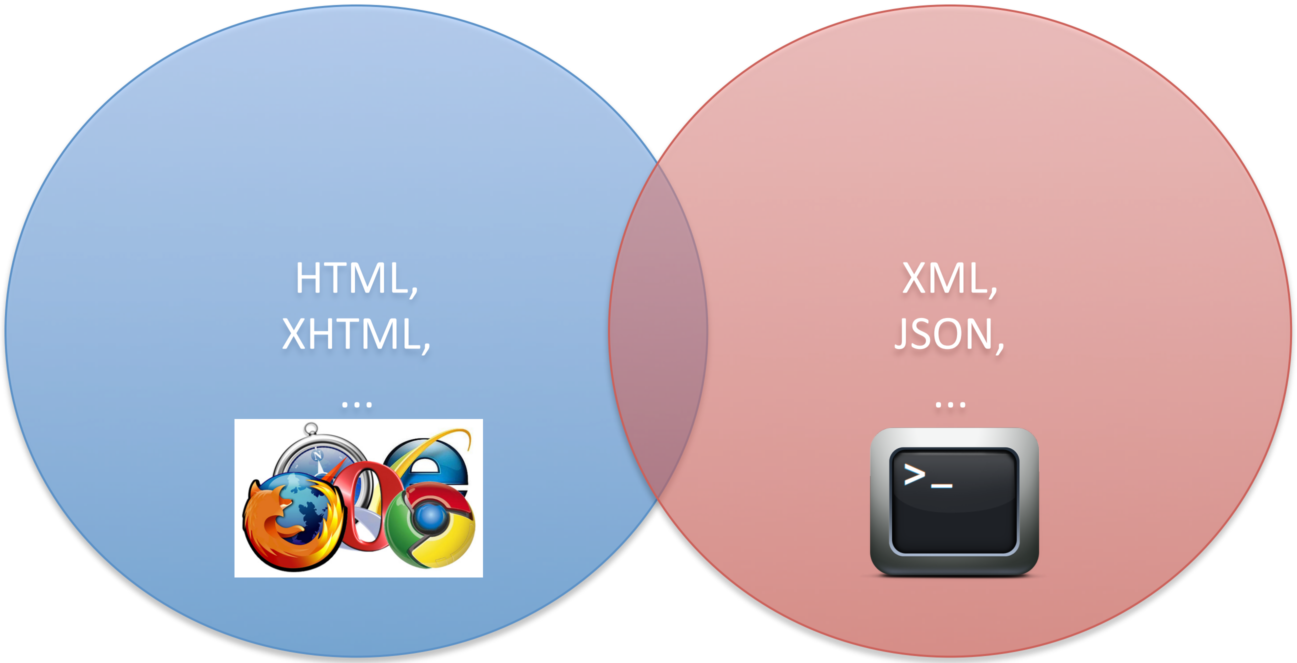


Interacting with Resources

- We deal with resource representations
  - not the resources themselves (pass by value)
  - representations can be in any format (defined by media-type)
- Each resource implements a standard uniform interface (HTTP)
  - a small set of verbs applied to a large set of nouns
  - verbs are universal and not invented on a per-application basis



Document/Data Formats



Display data

Transport and store data

APIS, WEB SERVICES

APIs

What is an API?

and

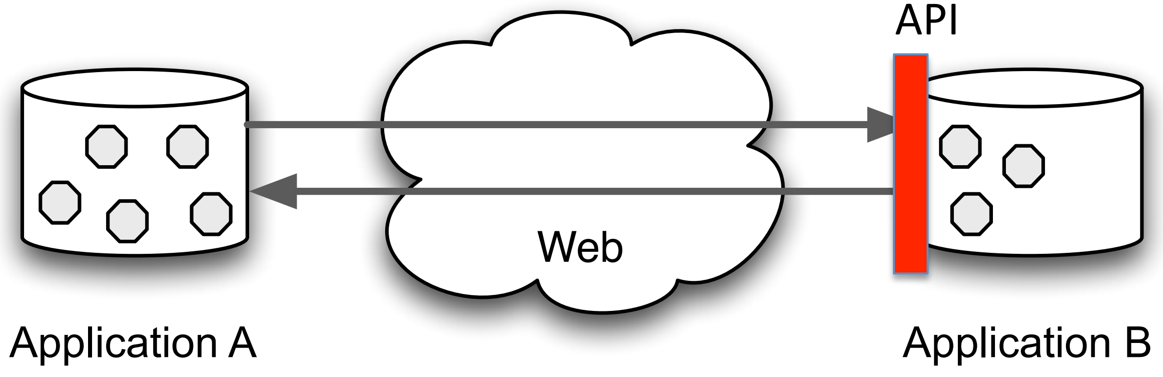
Why do we need APIs?

(Web) APIs

- Application Programming Interface
- Specifies how software components communicate with each other
  - e.g., Java API, 3rd party library APIs
  - usually come with documentation, howtos

- Web API: specify how applications communicate with other over the Web (HTTP, URI, XML, etc.)

Web Services



- Example operations:
  - Publish image on Flickr
  - Order a book at Amazon
  - Post a message on your friend’s Facebook wall
  - Update user photo on foursquare

Web Services

- “Web Services” ≅ “Web APIs”
- Build on the design principles and architectural components of the Web
- Provide certain operations
- Exchange structured data in standard formats (JSON, XML, etc)

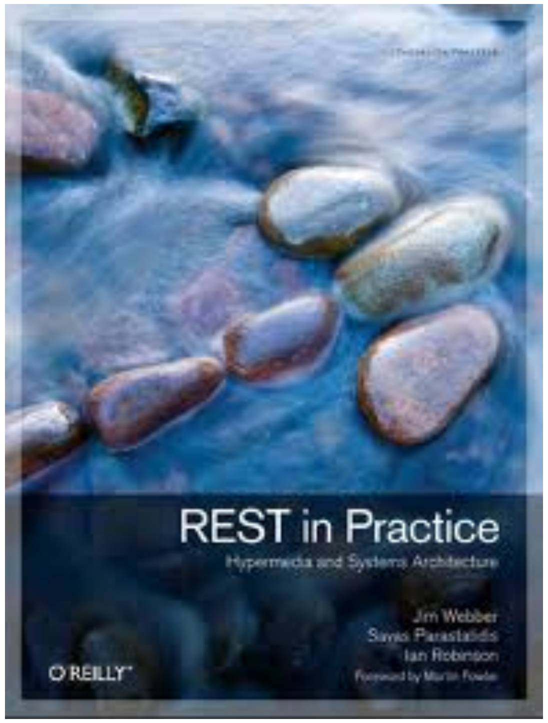
GROUP BRAINSTORMING

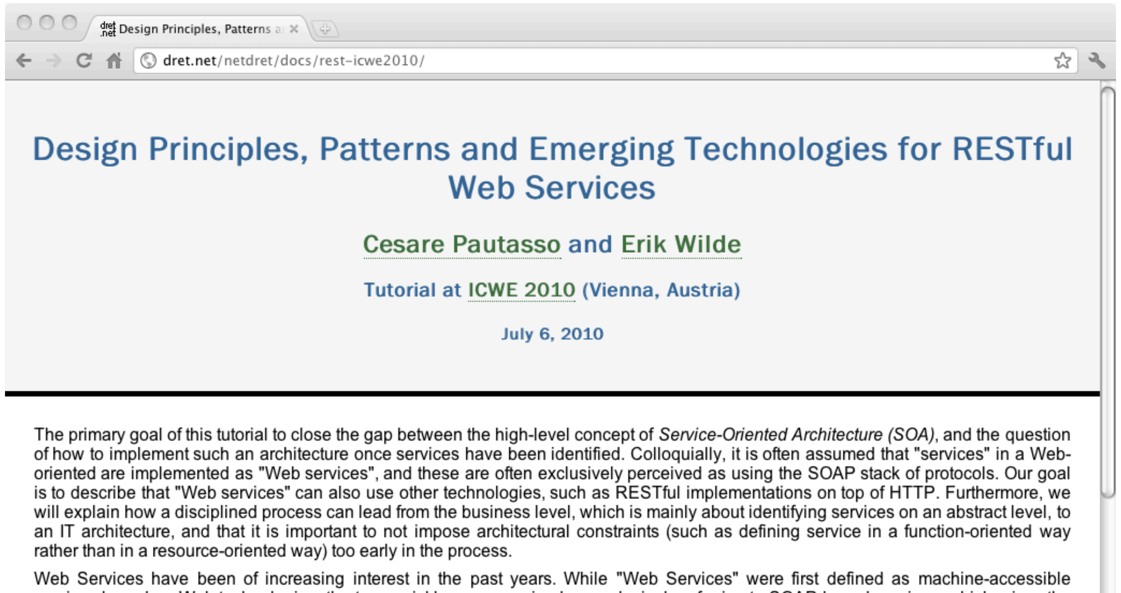
Instructions

- Form groups of 5
- 10 min:
  - discuss known or possible Web API operations (functions)
  - collect operations in the form:
    - [verb][noun] at [service]
  - one person per group should write them down at: http://bit.ly/info4302-api-brainstorming

RESTFUL APIS – ARCHITECTURAL PRINCIPLES

Web Services for the Real World
RESTful Web Services
O'REILLY
Leonard Richardson & Sam Ruby




Design Principles, Patterns a
dret.net/netdret/docs/rest-icwe2010/

Design Principles, Patterns and Emerging Technologies for RESTful Web Services

Cesare Pautasso and Erik Wilde

Tutorial at ICWE 2010 (Vienna, Austria)

July 6, 2010

The primary goal of this tutorial to close the gap between the high-level concept of *Service-Oriented Architecture (SOA)*, and the question of how to implement such an architecture once services have been identified. Colloquially, it is often assumed that "services" in a Web-oriented are implemented as "Web services", and these are often exclusively perceived as using the SOAP stack of protocols. Our goal is to describe that "Web services" can also use other technologies, such as RESTful implementations on top of HTTP. Furthermore, we will explain how a disciplined process can lead from the business level, which is mainly about identifying services on an abstract level, to an IT architecture, and that it is important to not impose architectural constraints (such as defining service in a function-oriented way rather than in a resource-oriented way) too early in the process.

Web Services have been of increasing interest in the past years. While "Web Services" were first defined as machine-accessible

RESTful Webservices

- REST = Representational State Transfer
  - Based on Chapter 5 of Roy Fielding's 2000 PhD thesis (it is in your reading list!)
- An architectural style for building loosely coupled systems
- The Web itself is an instance of that style
- Web Services can be built on top of it

The Resource-Oriented Architecture

- A set of design principles for building RESTful Web Services
  - Addressability
  - Uniform interface
  - Connectedness
  - Statelessness

Web Services for the Real World
RESTful
Web Services
O'REILLY
Leonard Richardson & Sam Ruby

Addressability

- An addressable application
  - exposes the interesting aspects of its dataset as resources
  - exposes a URI for every piece of information it might serve
  - which is usually an infinite number of URIs

Addressability

- A resource
  - is anything that is important enough to be referenced as a thing in itself
  - usually something
    - you want to serve information about
    - that can be represented as a stream of bits
      - actors
      - movies
  - a resource must have at least one name (URI)

Addressability

- Resource names (URIs)
  - the URI is the name and address of a resource
  - a resource's URI should be descriptive

```
http://example.com/movies

instead of

http://example.com/overview.php?list=all,type=movie
```

The Resource-Oriented Architecture

- A set of design principles for building RESTful Web Services
  - Addressability
  - Uniform interface
  - Connectedness
  - Statelessness

Web Services for the Real World
RESTful
Web Services
O'REILLY
Leonard Richardson & Sam Ruby

Uniform Interface

- The same set of operations applies to everything (every resource)
- A small set of verbs (methods) applied to a large set of nouns (resources)
  - verbs are universal and not invented on a per-application base
- Natural language works in the same way (new verbs rarely enter language)

Uniform Interface

- HTTP defines a small set of verbs (methods) for acting on URI-identified resources

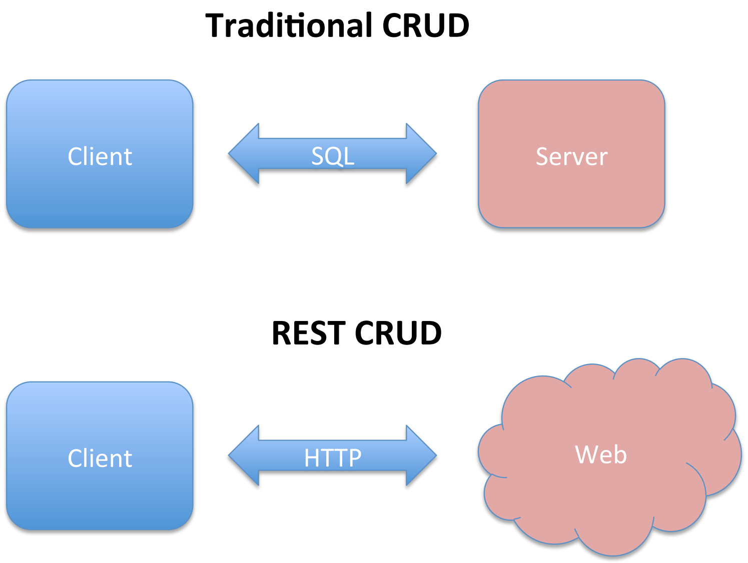
Which methods (verbs) are defined in HTTP?

Uniform Interface

- RESTful Web Services use HTTP to its full extent
  - Methods: GET, POST, PUT, DELETE, (...)
  - Request headers: Authorization, Content-Type, Last-Modified
  - Response Codes: 200 OK, 304 Not Modified, 401 Unauthorized, 500 Internal Server Error
  - Body: an envelope for data to be transported from A to B

Uniform Interface

- With HTTP we have all methods we need to manipulate Web resources (**CRUD** interface)
  - **Create** = POST (or PUT)
  - **Read** = GET
  - **Update** = PUT
  - **Delete** = DELETE



Mapping Web Service Operations to CRUD

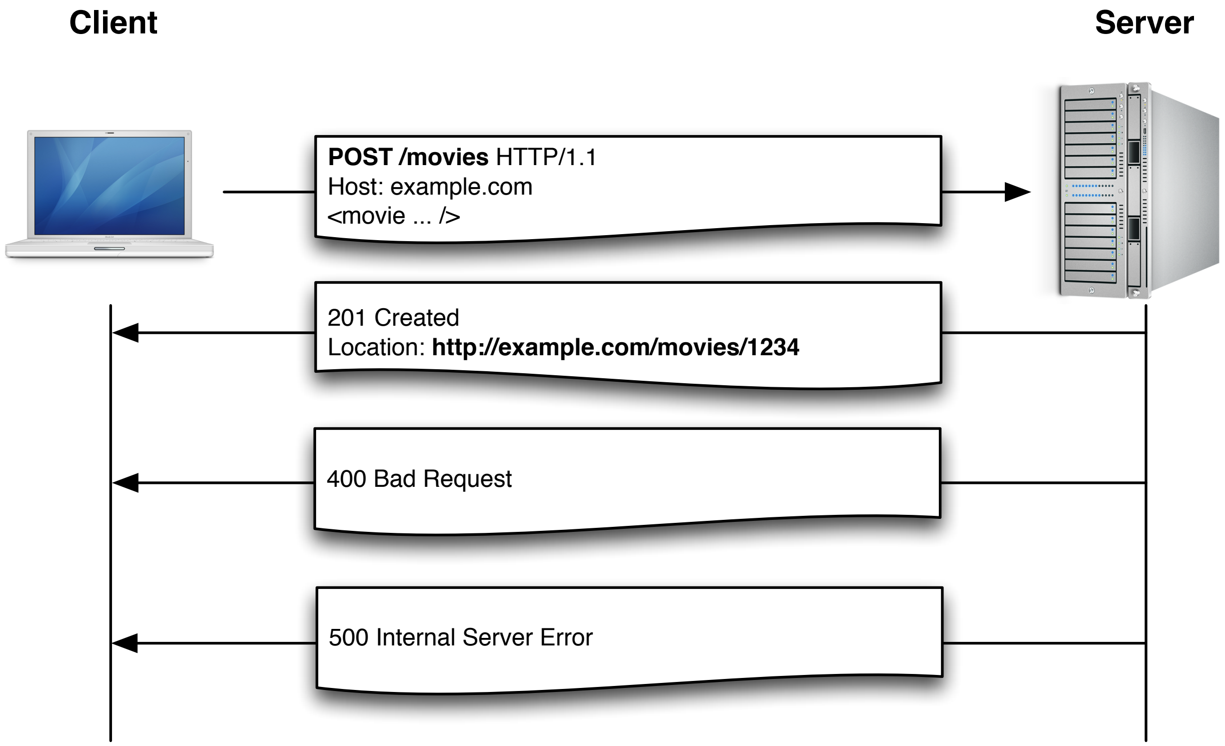
- C(reate):
  - order at Etsy, message on Facebook wall, ???
- R(read):
  - ???
- U(pdate):
  - user account on Etsy, ???
- D(elete):
  - order at Etsy, ???

Safe and Idempotent Behavior

- Safe methods can be ignored or repeated without side-effects: GET and HEAD
- Idempotent methods can be repeated without side-effects: PUT and DELETE
- Unsafe and non-idempotent methods should be treated with care: POST

Uniform Interface

- **CREATE** a new resource with HTTP POST


Client
Server
POST /movies HTTP/1.1
Host: example.com
<movie ... />
201 Created
Location: **http://example.com/movies/1234**
400 Bad Request
500 Internal Server Error

Example POST Request

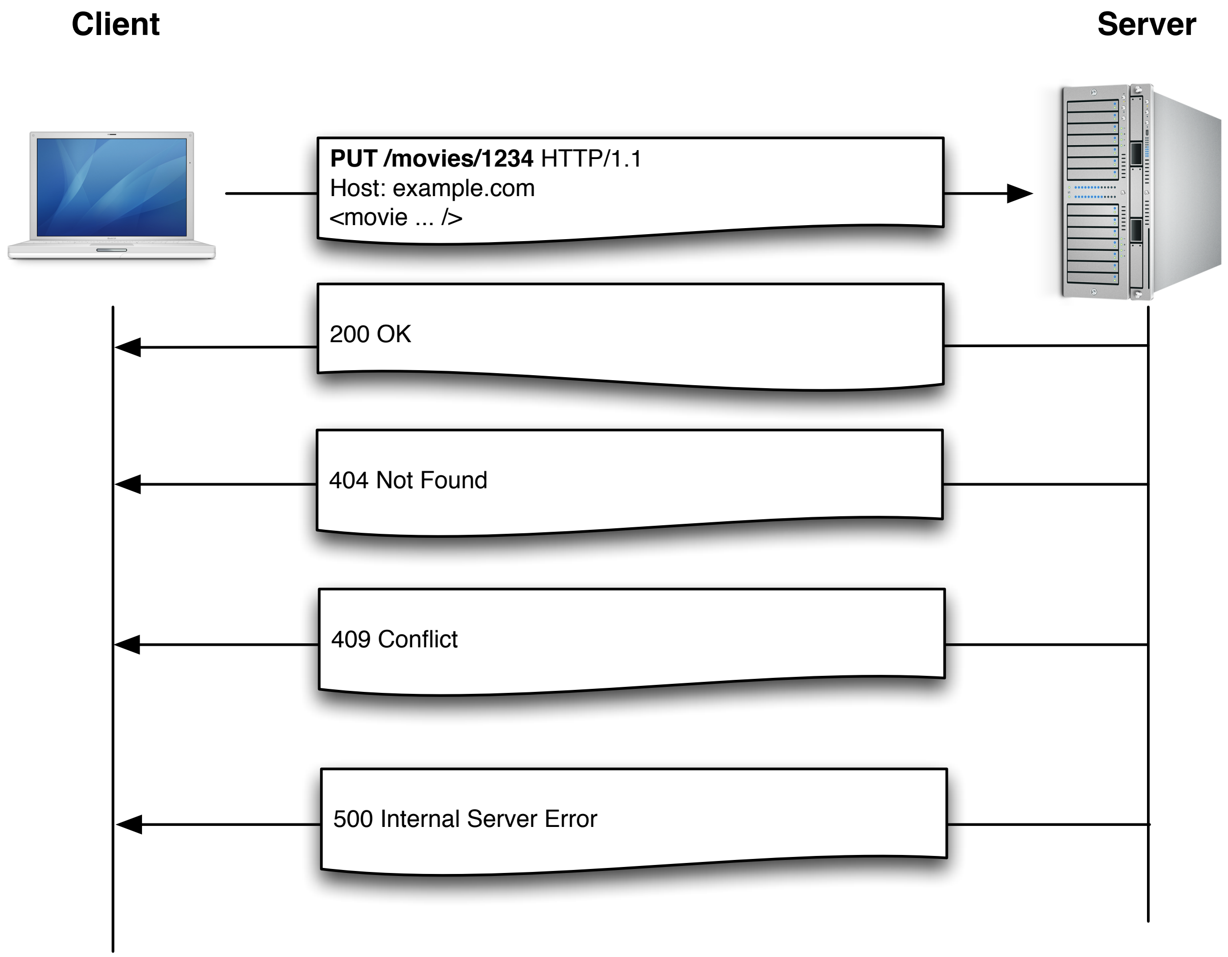
```
POST /movies HTTP/1.1
Host: example.com
...

<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
</movie>
```

POST Semantics

- POST creates a new resource
- The server decides on the resource's URI
- POST is **not idempotent**
  - A sequence of two or more POST requests has side-effects
  - Human Web:
    - "Do you really want to post this form again?"
    - "Are you sure you want to purchase that item again?"
  - Programmatic Web:
    - if you post twice, you create two resources

Uniform Interface

- **CREATE** a new resource with HTTP PUT



Example PUT Request

```
PUT /movies/1234 HTTP/1.1
Host: example.com
...

<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
</movie>
```

PUT Semantics

- PUT creates a new resource
- The client decides on the resource's URI
- PUT is **idempotent**
  - multiple PUT requests have no side effects
  - but it changes the resource state

Create with PUT or POST?

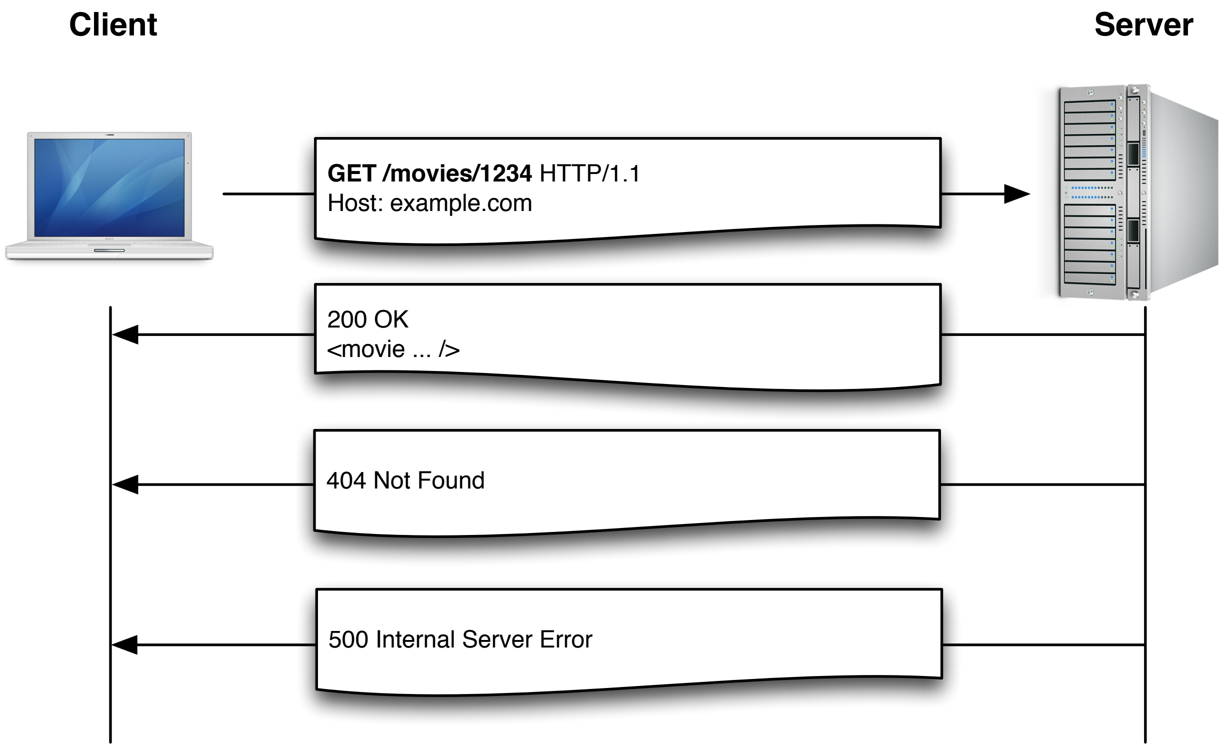
- The generic answer: it depends ☺
- Considerations
  - PUT if client
    - can decide on the URI
    - sends complete representation to the server
  - POST if server creates the URI (algorithmically)
  - some firewalls only allow GET and POST
  - POST is common practice

CREATE with PUT Example

```
# Create Amazon S3 bucket

PUT / HTTP/1.1
Host: colorpictures.s3.amazonaws.com
Content-Length: 0
Date: Wed, 01 Mar  2009 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

# Add Object to a bucket

PUT /my-image.jpg HTTP/1.1
Host: colorpictures.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
```

Uniform Interface

- **READ** an existing resource with HTTP GET



```
GET /movies/1234 HTTP/1.1
Host: example.com
```

```
200 OK
<movie ... />
```

```
404 Not Found
```

```
500 Internal Server Error
```

Example GET Request / Response

Request:

```
GET /movies/1234 HTTP/1.1
Host: example.com
Accept: application/xml
...
```

Response:

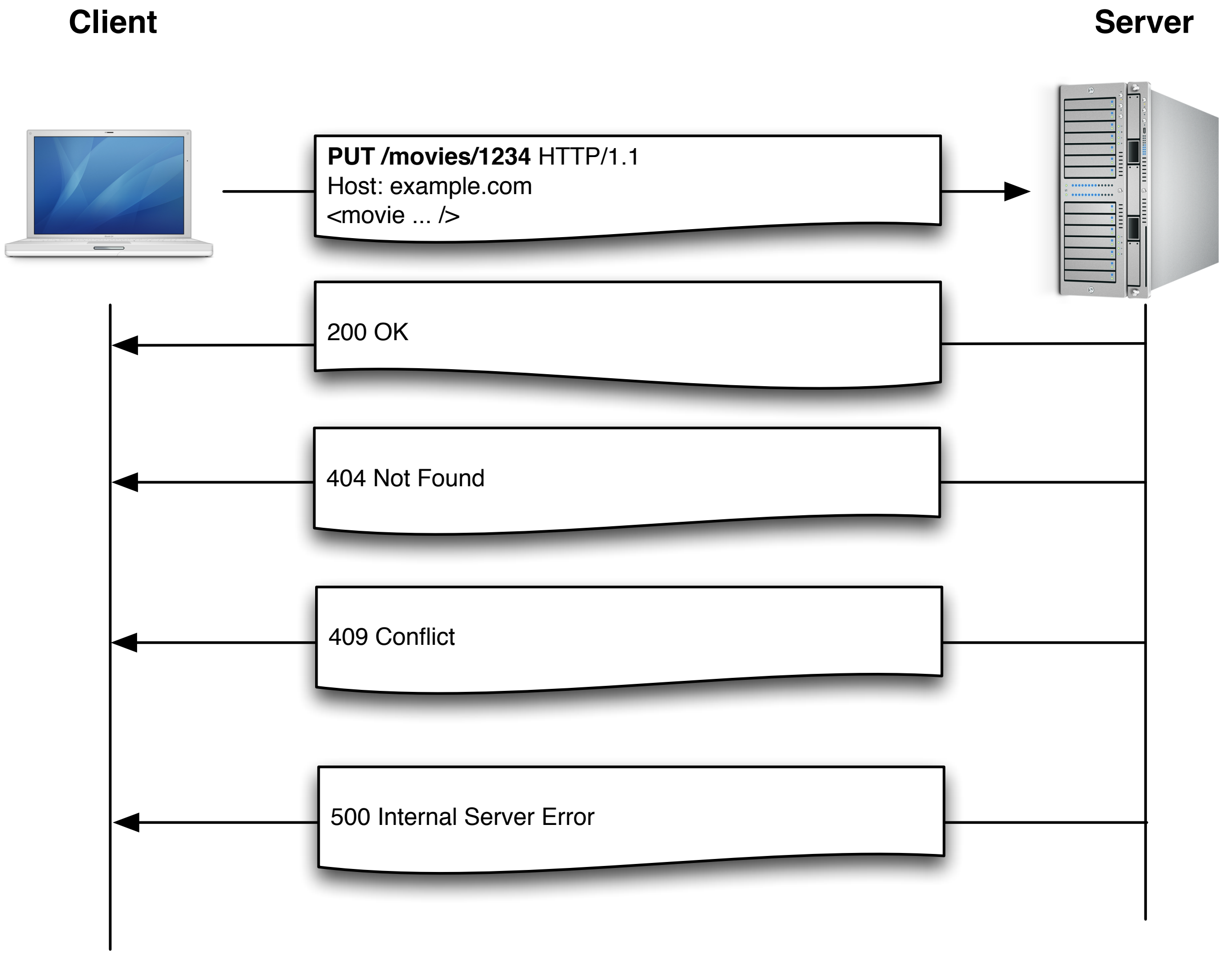
```
HTTP/1.1 200 OK
Date: ...
Content-Type: application/xml

<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
</movie>
```

GET Semantics

- GET retrieves the representation (= the current state) of a resource
- GET is safe (implies idempotent)
  - does not change state of resource
  - has no side-effects
- If GET goes wrong
  - GET it again!
  - no problem because it safe (and idempotent)

Uniform Interface

- **UPDATE** an existing resource with HTTP PUT



```
PUT /movies/1234 HTTP/1.1
Host: example.com
<movie ... />
```

200 OK

404 Not Found

409 Conflict

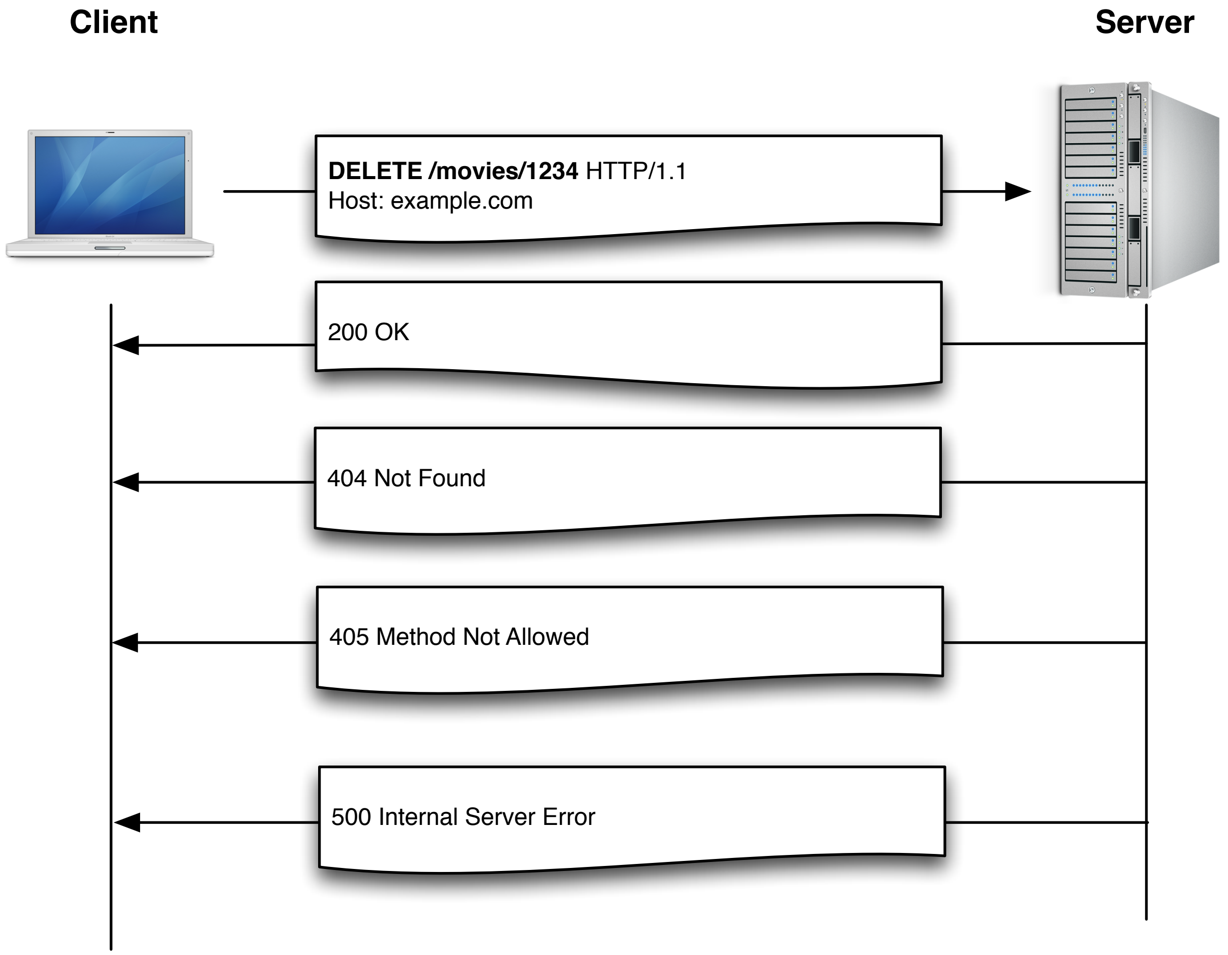
500 Internal Server Error

When PUT goes wrong

- If we get 5xx error, or some 4xx errors
  - simply PUT again!
  - no problem, because PUT is idempotent
- If we get errors indicating incompatible states then do some forward/backward compensation work and maybe PUT again
  - 409 Conflict (e.g., change your username to a name that is already taken)
  - 417 Expectation Failed (the server won't accept your representation – fix it, if possible)

Uniform Interface

- **DELETE** an existing resource with HTTP DELETE



DELETE Semantics

- Stop the resource from being accessible
  - logical delete
  - not necessarily physical
- If DELETE goes wrong
  - try it again!
  - DELETE is idempotent

The Resource-Oriented Architecture

- A set of design principles for building RESTful Web Services
  - Addressability
  - Uniform interface
  - Connectedness
  - Statelessness

Web Services for the Real World
RESTful Web Services
O'REILLY
Leonard Richardson & Sam Ruby

Connectedness

- In RESTful services, resource representations are hypermedia
- Served documents contain not just data, but also links to other resources

```
HTTP/1.1 200 OK
Date: ...
Content-Type: application/xml

<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
    <actor>http://example.com/actors/567</actor>
</movie>
```

The Resource-Oriented Architecture

- A set of design principles for building RESTful Web Services
  - Addressability
  - Uniform interface
  - Connectedness
  - Statelessness

Web Services for the Real World
RESTful
Web Services
O'REILLY
Leonard Richardson & Sam Ruby

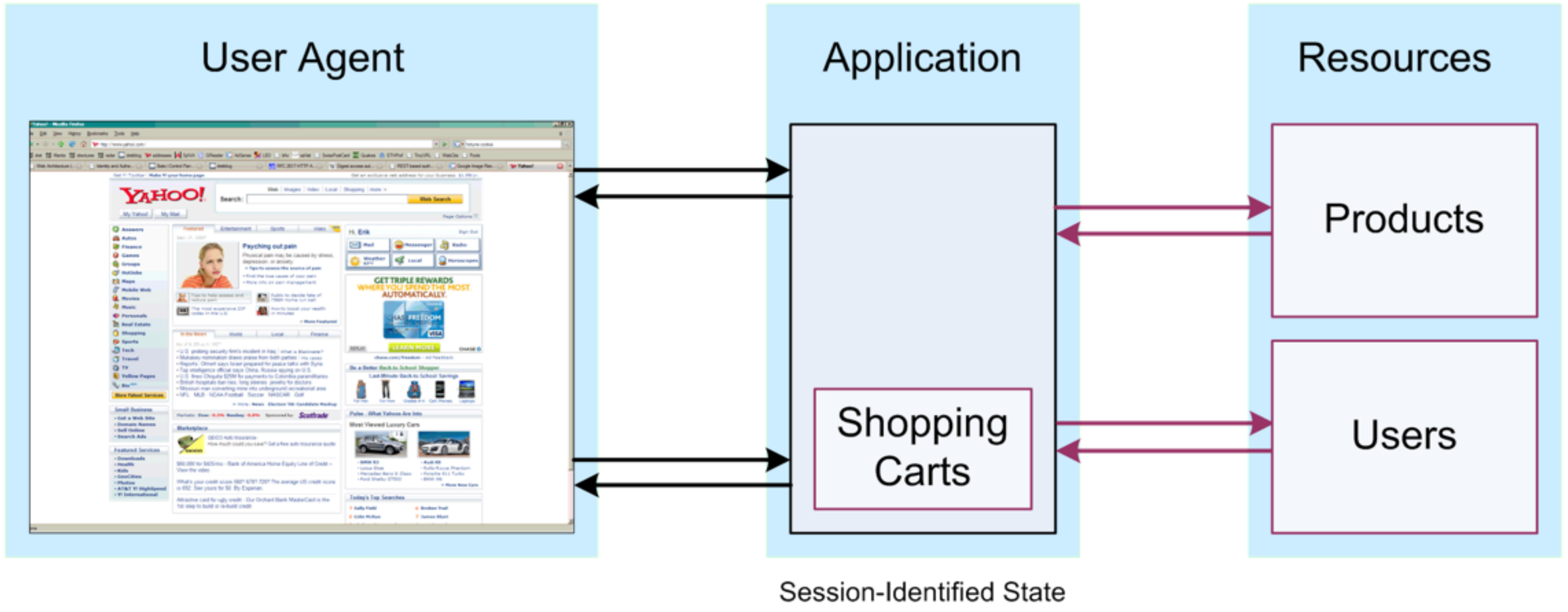
Statelessness

- Statelessness = every HTTP request executes in complete isolation
- The request contains all the information necessary for the server to fulfill that request
- The server never relies on information from a previous request
  - if information is important (e.g., user-authentication), the client must send it again

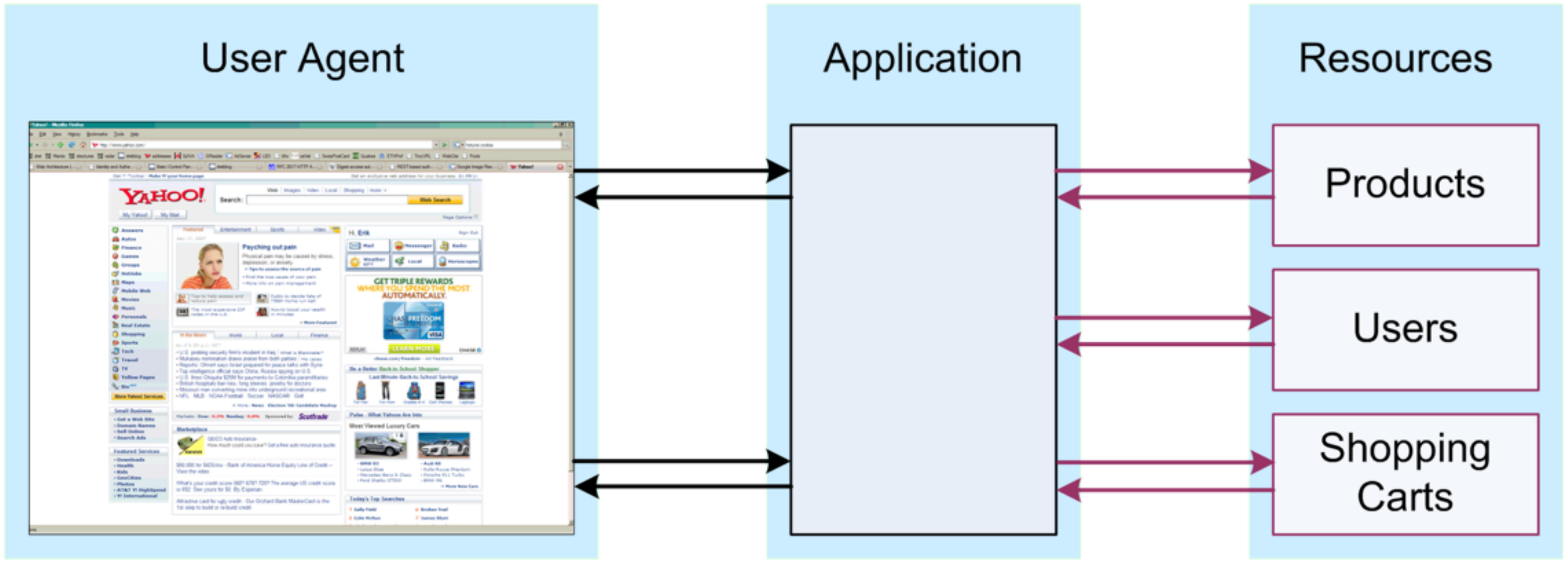
Statelessness

- This constraint does not say “stateless applications”!
  - for many RESTful applications, state is essential
  - e.g., shopping carts
- It means to move state to clients or resources
- State in resources
  - the same for every client working with the service
  - when a client changes resource state other clients see this change as well
- State in clients (e.g., cookies)
  - specific to client and has to be maintained by each client
  - makes sense for maintaining session state (login / logout)

State in the Application



Statelessness

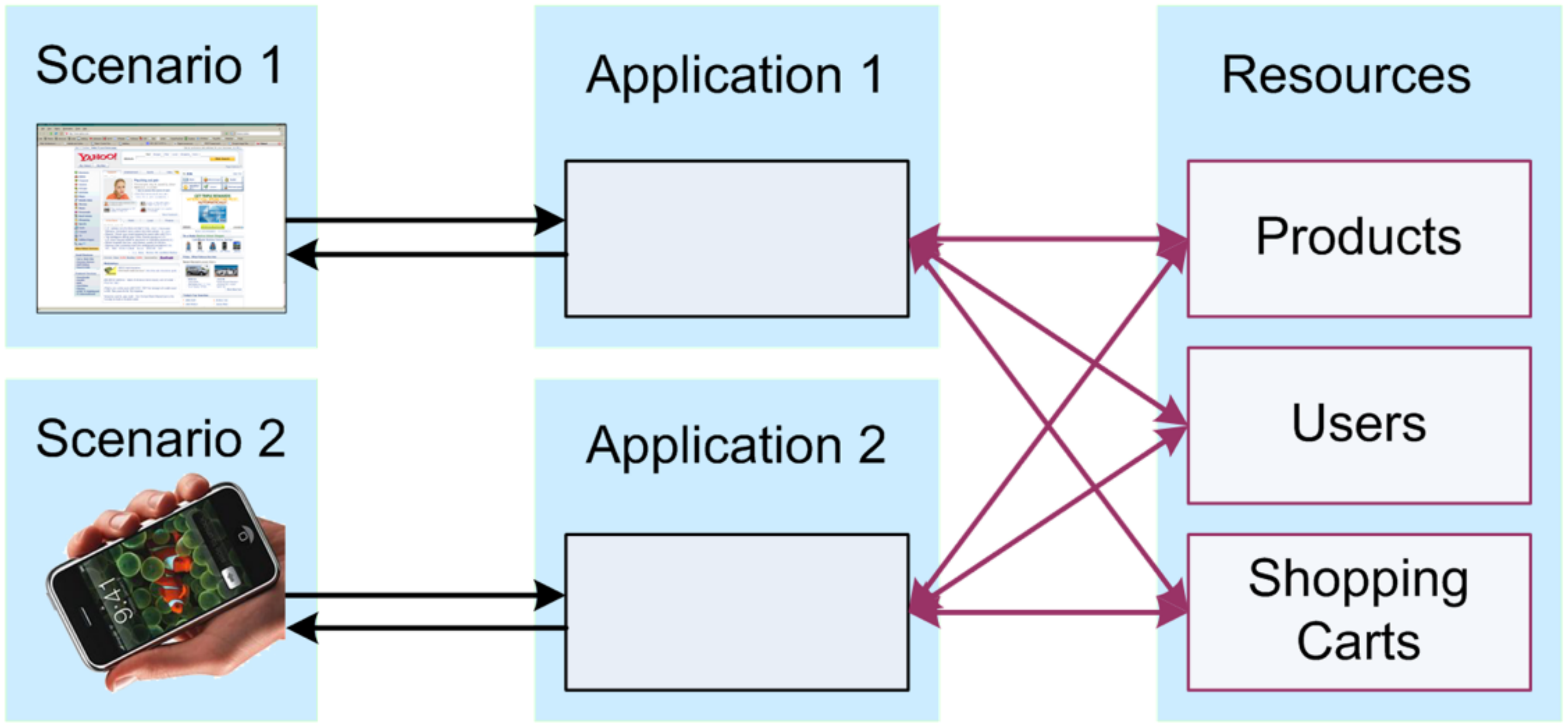


Application State (User Session)

State as a Resource

© Erik Wilde: http://dret.net/netdret/docs/rest-icwe2010/

Statelessness



© Erik Wilde: http://dret.net/netdret/docs/rest-icwe2010/

Tools and Frameworks

- Ruby on Rails - a framework for building RESTful Web applications
  - http://www.rubyonrails.org/
- Restlet - framework for mapping REST concepts to Java classes
  - http://www.restlet.org
- Django - framework for building RESTful Web applications in Python
- JAX-RC specification (http://jsr311.java.net/) provides a Java API for RESTful Web Services over the HTTP protocol.
- RESTEasy (http://www.jboss.org/resteasy/) - JBoss project that provides various frameworks for building RESTful Web Services and RESTful Java applications. Fully certified JAX-RC implementation.

Readings

- Fielding, Roy: Architectural Styles and the Design of Network-based Software Architectures (Chapters 4-6): http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Tutorial Design Principles, Patterns and Emerging Technologies for RESTful Web Services (Cesare Pautasso and Erik Wilde): http://dret.net/netdret/docs/rest-icwe2010/