RESTful Web Services

Marco Fabbri
marco.fabbri28@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM—Università di Bologna

DS 2013/2014
Distributed Systems – 15th of May 2014
What is REST?

REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations. This is achieved by placing constraints on connector semantics where other styles have focused on component semantics. REST enables the caching and reuse of interactions, dynamic substitutability of components, and processing of actions by intermediaries, thereby meeting the needs of an Internet-scale distributed hypermedia system.
What is REST?

- REST is not a standard
- REST is not a protocol
- REST is an architectural style for networked applications
- REST defines a set of simple principles (loosely followed by most API implementations)
Advantages of REST

• Cacheable
• Stateless
• Scalable
• Fault-tolerant
• Loosely coupled
The Constraints of REST

1. Client-server
2. Stateless server
3. Cache
4. Uniform interface
   1. Identification of resources
   2. Manipulation of resources through representations
   3. Self-descriptive messages
   4. Hypermedia as the engine of application state or HATEOAS
5. Layered system
6. Code-on-demand (optional)
Client-Cache-Stateless-Server Architecture

- Each request must contain all information.
- No stored context on the server.
- Client has the right to reuse response data.
The first three constraints—client-server, stateless server, and cache—combine to give us the robust client-cache-stateless-server architecture that allows today’s Web to scale. Requests travel to and exchange information with the server. The server does not store data from client requests, but the client can reuse response data, sent by the server, by storing it in a local cache.
Code-on-Demand Architecture

- Add features to a deployed client, which provides for improved extensibility and configurability.
- Better user-perceived performance and efficiency.

Client \[\text{code} \] → Stateless Server
The optional code-on-demand constraint allows clients to request and execute code from servers. This, in turn, allows the server to deploy new features to clients. The result is improved extensibility and configurability for servers, and improved performance and efficiency for clients. From an API perspective, JavaScript returns an HTTP response that is executed on a local Web browser.
Uniform Interface Constraint

• The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components.

• Generality (applied to the component interface)
(some) Principles of REST

- URL identifies a resource
- URLs have a hierarchy
- Methods perform operations on resources
- Operation must be implicit
- Hypermedia format to represent data
- Link relations to navigate
Uniform Interface Constraint
- Four Principles

1. Identification of resources
2. Manipulation of resources
3. Self-descriptive messages
4. HATEOAS
Identification of Resources

- /index.php?action=getarticle&id=5
- /default/article/5/4/6/size

Cacheable? NO

Scalable? NO

Readable? NO
Identification of Resources

• /articles
  We want all articles

• /articles/5/photos/4/comments/1
  We want the first comment of the fourth photo for the fifth article

• /articles/5/photos/4/comments
  We want all comments of the fourth photo for the fifth article

  Cacheable? YES

  Scalable? YES

  Readable? YES
Identification of Resources

Filtering through a Query String, not the URI

✗ /photos/order/size/limit/5
✗ /photos/limit/5/order/size
✓ /photos?order=size&limit=5
✓ /photos?limit=5&order=size
Manipulation of Resources

- Create
- Retrieve
- Update
- Delete

Please note: REST != CRUD
Manipulation of Resources

CRUD to HTTP verb mapping

• Create -> POST
• Retrieve -> GET
• Update -> PUT (or PATCH)
• Delete -> DELETE
Manipulation of Resources

Creating a Resource

• POST creates a new resource
  But the server decides on that resources URI

• Examples

  • WWW: Posting to Web log
    • Server decides URI of posting
      and any comments made on that post

  • Programmatic service: Creating a new employee record
Manipulation of Resources

Safe Methods

• Any client should be able to make the request . . . as many times as necessary

• GET, OPTIONS, HEAD
Manipulation of Resources

Idempotent Methods

• Guarantees that the client can repeat the request when it’s not certain

• $x++$ vs. $x=4$

• All methods except “POST”
Self-Descriptive Messages

• Stateless!

• All information for processing is available:
  • How? (method + content-type)
  • What? (URI)
  • When? (preconditions)
  • Who? (authentication)
Self-Descriptive Messages

How: Method

**GET** /article/1234 HTTP/1.1
Host: www.mycompany.com
Accept: application/vnd.mycompany.myapp-v1+json
Authorization: OAuth oauth_nonce="123" ...
If-None-Matched: absad12412414
Self-Descriptive Messages

How: Content-Type

GET /article/1234 HTTP/1.1
Host: www.mycompany.com
Accept: application/vnd.mycompany.myapp-v1+json
Authorization: OAuth oauth_nonce="123" ...
If-None-Matched: absad12412414
Self-Descriptive Messages

application/vnd.mycompany.myapp-v1+json

The vnd name space is for proprietary media types

(as opposed to the IANA registered ones)
We want to “talk” JSON, not XML or others
We want to “play” with API version 1.0 (not any other), also Header: Accept-Version

General notes:
Interpret requests generously
Be strict with responses
Self-Descriptive Messages

What

GET /article/1234 HTTP/1.1
Host: www.mycompany.co.m
Accept: application/vnd.mycompany.myapp-v1+json
Authorization: OAuth oauth_nonce="123" ...
If-None-Matched: absad12412414
Self-Descriptive Messages

When

GET /article/1234 HTTP/1.1
Host: www.mycompany.co.m
Accept: application/vnd.mycompany.myapp-v1+json
Authorization: OAuth oauth_nonce="123" ...
If-None-Matched: absad12412414
Self-Descriptive Messages

Who

GET /article/1234 HTTP/1.1
Host: www.mycompany.co.m
Accept: application/vnd.mycompany.myapp-v1+json
Authorization: OAuth oauth_nonce="123" ...
If-None-Matched: absad12412414
Self-Descriptive Messages

XML API vs JSON API

http://goo.gl/0QIhjU
Self-Descriptive Messages

WADL?

We can describe RESTful XML Web Services (similar to WSDL)

Web Application Description Language (WADL) (another XML grammar to describe HTTP-based web applications)

But we don’t need a static contract description → We want self-descriptive messages
HATEOAS

HATEOAS

= 

Hypermedia As The Engine Of Application State

(makes the API explorable)
Links

• Use links to allow clients to discover locations and operations

• Link relations are used to express options

• Clients do not need to know URLs

• This controls the state
Links

• Links contain (adapted from Atom format’s link definition): The target (href, mandatory)

• A short relationship indication (rel, mandatory) (e.g. “details”, “payment”, “cancel”)

• The content type needed for the request (type, optional) The HTTP method (method, optional)

• See also:

• http://www.subbu.org/blog/2008/10/generalized-linking
Example: Flight Booking API

- The HATEOAS links indicate state transitions