REPRESENTATIONAL STATE TRANSFER

Research Center for Assistive Information and Communication Solutions

APPLIED SCIENCE BY FRAUNHOFER – MADE IN PORTUGAL

© Fraunhofer Portugal

Fraunhofer PORTUGAL

REST

The Birth of REST

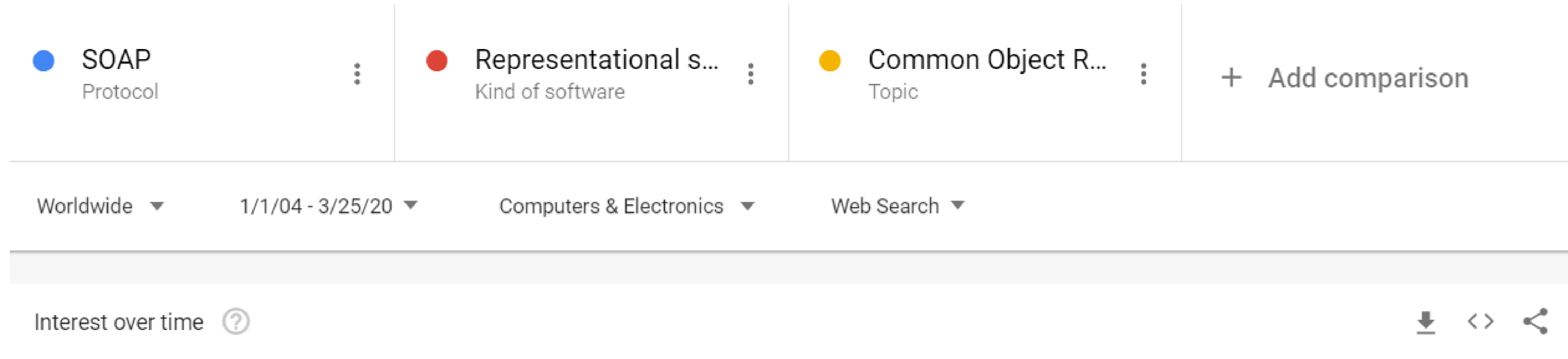
- SOAP was notorious for being complex to build, complex to use, and near-impossible to debug. And the alternative, CORBA, was even worse. The problem was that there was no standard for how APIs should be designed and used. APIs were not designed to be accessible; they were only designed to be flexible.
- But a small group of expert developers recognized the true potential of web APIs. Thanks to this small group, led by Roy Fielding, REST was coined, and the API landscape changed forever.
- In 2000, Roy Fielding and his colleagues had one objective: create a standard so that any server could talk to any other server in the world.

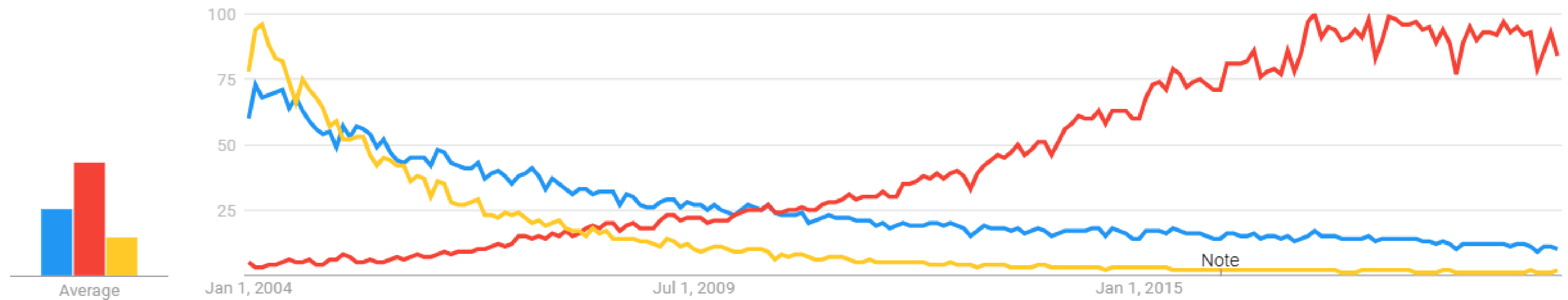
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

© Fraunhofer Portugal

REST

The Birth of REST





© Fraunhofer Portugal
Fraunhofer PORTUGAL

REST

What is?

- Representational State Transfer (REST) is a term coined by Roy Fielding to describe an architecture style of networked systems.

“REST architectural style for distributed hypermedia systems, describing the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, while contrasting them to the constraints of other architectural styles”

Roy Fielding

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

© Fraunhofer Portugal

REST
In a nutshell

- How to represent resources differently
- Client-server communication
- How to manipulate resources
- Simple, interoperable and flexible way of writing web services

Rest is NOT

- A Protocol
- A standard
- A Replacement for SOAP

© Fraunhofer Portugal
Fraunhofer PORTUGAL

REST

Constraints

- **Client-server**: Interactions between components are based on the request-response pattern, where clients send a request and servers reply-back with a response.
- **Uniform interface:** unambiguous, simple, and standard interface respected by all components. Using HTTP verbs (GET, PUT, POST, DELETE), URIs as our resources, and get an HTTP response with a status and a body.
- **Stateless:** Each request is self-descriptive and has enough context for the server to process that message. Client context and state should be kept only on the client, not on the server.
- **Cacheable:** Unless denoted, a client can cache any representation to boost their loading time.
- **Layered system:** Possible to use intermediary servers to further improve scalability and response times.

© Fraunhofer Portugal

REST

Principles

- **Addressable resources:** The key abstraction of information is a resource, named by a URI. Any information that can be named can be a resource
- **Manipulation of resources through representations:** All interactions are context-free: each interaction contains all the information necessary to understand the request, independent of any requests that may have preceded it
- **Self-descriptive messages:** The representation of a resource is a sequence of bytes, plus representation metadata to describe those bytes. The form of the representation can be negotiated between REST components
- **Hypermedia as the engine application state:** use links to connected related ideas and each possible state of the device needs to be RESTful resource with its own unique URL

© Fraunhofer Portugal

REST - Principles

Addressable resources

- Every HTTP request happens in complete isolation
  - Server NEVER relies on information from prior requests
  - There is no specific ‘ordering’ of client requests (i.e. page 2 may be requested before page 1)
  - If the server restarts a client can resend the request and continue from it left off
- Possible states of a server are also resources and should be given their own URIs

```
<scheme> “:” <authority><path> [ “?” query ] [ “#” fragment ]
```

URL syntax

  - `<scheme>` can be either http or https
  - `<authority>` hostname and optionally port number and access credentials
  - `<path>` hierarchical path to the resource
  - `[query]` and `[fragment]` optionally it can be added query parameters and fragments

© Fraunhofer Portugal
Fraunhofer PORTUGAL

REST - Principles

Addressable resources

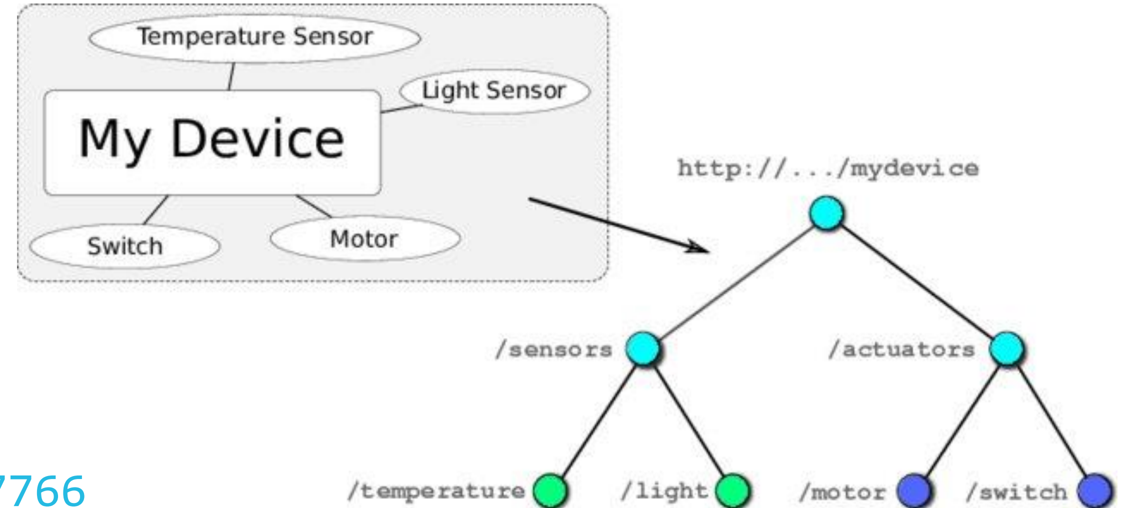
- Examples of URLs
  - https://www.google.com/search?q=porto
  - https://192.168.1.100:8080
  - https://192.168.1.100:8080/api/devices/
- Representing actual properties
  - `# Device named light1`
  - http://lapd2021.ddns.net/api/devices/lamp-1
  - `# Sample No. 7766 from January 2019`
  - https://webofthings.org/samples/2019/01/7766

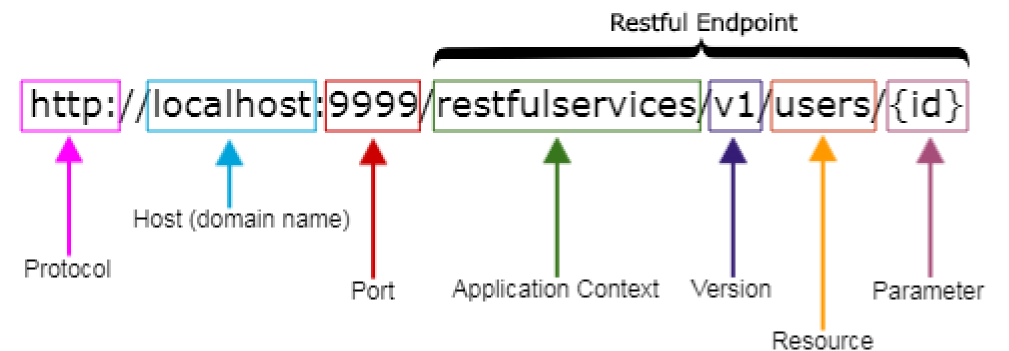
© Fraunhofer Portugal
Fraunhofer PORTUGAL

REST - Principles

Addressable resources

- Resources can be organized in hierarchy
  - `# List of devices in an building 4`
  - https://192.168.1.100:8080/building4/devices
  - `# Sample No. 7766 from January 2019`
  - https://webofthings.org/samples/2019/01/7766
  - `# List of sensor of device 24`
  - https://devices.webofthings.io/24/sensors





© Fraunhofer Portugal
Fraunhofer PORTUGAL

REST - Principles

Addressable resources

- Design rules for an addressable web resource
  - **Web Things must have an HTTP server:** support HTTP version 1.1 but ideally v2 as well. The server does not need to be hosted on the device
  - **Web Things should use HTTPS:** a thing should offer only secure connections specially when accessible from the internet
  - **Web Things must have a root resource accessible via an HTTP URL:** client applications must have a URL to send HTTP requests
  - **Web Things must expose their properties using a hierarchical structure:** to facilitate the discovery of resources

© Fraunhofer Portugal

REST - Principles

Manipulation of resources through representations

- Resources are NOT data – they are an abstraction of how the information/data is split up for presentation/consumption
- The web server must respond to a request by sending a series of bytes in a specific file format, in a specific language – i.e. a representation of the resource
  - Formats: XML/JSON, HTML, PDF, PPT, DOCX...
  - Languages: English, Spanish, Hindi, Portuguese…

© Fraunhofer Portugal

REST - Principles

Manipulation of resources through representations

- Style 1: Distinct URI for each representation:
  - ex.com/press-release/2012-11.en (English)
  - ex.com/press-release/2012.11.fr (French)
  - …and so on
- Style 2: Content Negotiation
  - Expose Platonic form URI:
    - ex.com/press-release/2012-11
  - Client sets specific HTTP request headers to signal what representations it's willing to accept
    - **Accept:** Acceptable file formats
    - **Accept-Language:** Preferred language

© Fraunhofer Portugal

Fraunhofer PORTUGAL

REST - Principles

Manipulation of resources through representations

- Simple HTTP request and response
  - `$ curl -i -v -u admin:lapd2021 -H "Accept: application/json" -H "Content-Type: application/json" -X GET` http://lapd2021.ddns.net/api/devices/lamp-1
- Clients specify representation format
  - `> Accept: application/json`
- Servers specify representation format
  - `> Content-Type: application/json`

clouduser@i-163: ~

```
clouduser@i-163:~$
clouduser@i-163:~$ curl -i -v -u demo:demo -H "Accept: application/json" -H "Con
* Hostname was NOT found in DNS cache
*   Trying 185.82.21.117...
* Connected to demo.pimatic.org (185.82.21.117) port 80 (#0)
* Server auth using Basic with user 'demo'
> GET /api/devices/light1 HTTP/1.1
> Authorization: Basic ZGVtbzpkZW1v
> User-Agent: curl/7.35.0
> Host: demo.pimatic.org
> Accept: application/json
> Content-Type: application/json
>
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
* Server nginx/1.4.6 (Ubuntu) is not blacklisted
< Server: nginx/1.4.6 (Ubuntu)
Server: nginx/1.4.6 (Ubuntu)
< Date: Wed, 27 Feb 2019 15:37:07 GMT
Date: Wed, 27 Feb 2019 15:37:07 GMT
< Content-Type: application/json; charset=utf-8
Content-Type: application/json; charset=utf-8
< Content-Length: 1718
Content-Length: 1718
< Connection: keep-alive
Connection: keep-alive
< X-Powered-By: Express
X-Powered-By: Express
< Cache-Control: no-cache, no-store, must-revalidate
Cache-Control: no-cache, no-store, must-revalidate
< Pragma: no-cache
Pragma: no-cache
< Expires: 0
Expires: 0
< ETag: W/"6b6-XTjPgE+7jKVhihleogepJg"
ETag: W/"6b6-XTjPgE+7jKVhihleogepJg"
< Set-Cookie: pimatic.sess=eyJ1c2VybmFtZSI6ImRlbW8iLCJsb2dpblRva2VuIjoiYzdhMWNhO
Set-Cookie: pimatic.sess=eyJ1c2VybmFtZSI6ImRlbW8iLCJsb2dpblRva2VuIjoiYzdhMWNhODc
< Set-Cookie: pimatic.sess.sig=V1t336cxEZk8aFoP8h8SEkFSBkE; path=/; httponly
Set-Cookie: pimatic.sess.sig=V1t336cxEZk8aFoP8h8SEkFSBkE; path=/; httponly

<
{"device":{"id":"light1","name":"Lamp 1","template":"switch","attributes":[{"des
":1550915499699,"v":true},{"t":1550961112730,"v":false},{"t":1550961113357,"v":
":1551074966724,"v":true},{"t":1551116249344,"v":false},{"t":1551116643954,"v":t
1551188564431,"v":true},{"t":1551188565049,"v":false},{"t":1551188568000,"v":tr
1551212377057,"v":true},{"t":1551221357680,"v":false},{"t":1551221647843,"v"* Co
true},{"t":1551221657886,"v":false},{"t":1551262997772,"v":true},{"t":155126299
nOff"},{"description":"Changes the switch to on or off","params":{"state":{"typ
h","returns":{"state":{"type":"boolean"}},"name":"getState"}],"config":{"id":"l
$
```

Request

Response headers

Response body

© Fraunhofer Portugal

Fraunhofer PORTUGAL

REST - Principles

Manipulation of resources through representations

- Design rules for content negotiation
  - **Web Things must support JSON as their default representation:** can support other types as long it supports JSON. Always use CamelCase for objects names in JSON payloads (e.g., lastValue instead of Last-Value or last_value)
  - **Web Things support UTF8 encoding for requests and responses:** can support many encoding formats (e.g., Chinese or Russian) but it has to support UTF8 for any resource
  - **Web Things may offer an HTML interface/representation (UI):** besides the computer-friendly API devices should also offer human friendly user interfaces

© Fraunhofer Portugal

REST - Principles

Self-descriptive messages - HTTP Request/Response

- HTTP Provides 4 basic methods for CRUD (create, read, update, delete) operations:
  - **GET**: Retrieve representation of resource
  - **PUT**: Update/modify existing resource (or create a new resource)
  - **POST**: Create a new resource
  - **DELETE**: Delete an existing resource
- Another 3 less commonly used methods:
  - **HEAD**: Fetch meta-data of representation only (i.e. a metadata representation)
  - **OPTIONS**: Check which HTTP methods a particular resource supports
  - **PATCH**: used for partial modifications to a existing resource i.e. delta updates.

© Fraunhofer Portugal

REST - Principles

Self-descriptive messages - HTTP Request/Response

Method	Request Entity-Body/Representation	Response Entity-Body/Representation
GET	(Usually) Empty Representation/entity-body sent by client	Server returns representation of resource in HTTP Response
DELETE	(Usually) Empty Representation/entity-body sent by client	Server may return entity body with status message or nothing at all
PUT	(Usually) Client’s proposed representation of resource in entity-body	Server may respond back with status message or with copy of representation or nothing at all
POST	Client’s proposed representation of resource in entity-body	Server may respond back with status message or with copy of representation or nothing at all

© Fraunhofer Portugal

Fraunhofer PORTUGAL

REST - Principles

Self-descriptive messages - PUT vs. POST

- POST
  - Commonly used for creating subordinate resources existing in relation to some 'parent' resource
    - Parent: /weblogs/myweblog
    - Children: /weblogs/myweblog/entries/1
    - Parent: Table in DB; Child: Row in Table
- PUT
  - Usually used for modifying existing resources
  - May also be used for creating resources
- PUT vs. POST (for creation)
  - PUT: Client is in charge of deciding which URI resource should have
  - POST: Server is in charge of deciding which URI resource should have

© Fraunhofer Portugal

Fraunhofer PORTUGAL

REST Principles

Self-descriptive messages - PUT vs. POST

- What to do in case of partial updates or appending new data? PUT or POST?
  - PUT states: Send completely new representation overwriting current one
  - POST states: Create new resource
- In practice:
  - PUT for partial updates works fine. No evidence/claim for ‘why’ it can’t (or shouldn’t) be used as such
  - POST may also be used and some purists prefer this

© Fraunhofer Portugal

REST Principles

Self-descriptive messages – HTTP Status/Response Codes

- 2xx Success
  - **200 OK** – returns upon successful completion of a request
  - **201 Created** – returned when new resource is successfully created
  - **202 Accepted** – for asynchronous operations when operation is accepted but resource not created
- 3xx Redirection
- 4xx Client error
  - **401 Unauthorized** – requires user authentication / authorization failed using given credentials
  - **404 Not Found** – resource or document not found on the server
- 5xx Server error
  - **500 Internal Server Error** – error that prevented server from fulfilling request
  - **501 Service Unavailable** – request cannot be handle due to maintenance of temporary overload

© Fraunhofer Portugal

Fraunhofer
PORTUGAL

REST Principles

Hypermedia as the engine application state

- Hypermedia is the usage of different media formats (text, video, images, sounds) to create links between related ideas
  - Links can be used to retrieve more information about the sub-resources of a device

- REST requires the engine of application engine state to be hypermedia driven
  - Clients can retrieve the status of the current state and the possible transitions to other states
  - Applications can be stateful as long the client state is not stored in the server

© Fraunhofer Portugal

Fraunhofer PORTUGAL

Steps to a RESTful Architecture

Read the Requirements and turn them into resources

- Figure out the data set
- Split the data set into resources and for each kind of resource:
  - Name resources with URIs
  - Expose a subset of uniform interface
  - Design representation(s) accepted from client (Form-data, JSON, XML to be sent to server)
  - Design representation(s) served to client (file-format, language and/or (which) status message to be sent)
  - Consider typical course of events: sunny-day scenarios
  - Consider alternative/error conditions: rainy-day scenarios

© Fraunhofer Portugal

Fraunhofer PORTUGAL

References

- Dominique Guinard and Vlad Trifa. 2016. **Building the Web of Things: With Examples in Node.Js and Raspberry Pi** (1st ed.). Manning Publications Co., Greenwich, CT, USA.
- https://webofthings.org/team/
- https://pimatic.org/api/actions/ and https://pimatic.org/guide/api/ (API used for the assignment)

© Fraunhofer Portugal

Fraunhofer PORTUGAL