Course notes
Restful Web services

Neila BEN LAKHAL (PhD @ Tokyo Institute of Technology)
By Neila.benlakhal@enicarthage.rnu.tn

What is REST?

- REST is a term coined by Roy Fielding in his Ph.D [1]. Dissertation to describe an architecture style of networked systems.
- An acronym standing for **REpresentational State Transfer.**
- It Defines the architectural style for building large-scale distributed ***hypermedia*** systems :
  - Example of distributed hypermedia system: **WWW**
- By “architectural style” -- is completely independent of HTTP and the Web. It can be used with HTTP and the Web, but it doesn't have to be.
- REST is not a "standard". There will never be a W3C recommendation for REST, for example.

[1]Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures, PhD Thesis, University of California, Irvine, 2000
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Why REST?

- The idea is that, rather than using complex mechanisms such as CORBA, or RPC to connect between machines, simple **HTTP** is used to make calls between machines.
- Why ?
  - To serve data to **machines** in the same way the web serves data to **humans**
- In many ways, the **WWW** itself, based on HTTP, can be viewed as a REST-based architecture.
- How ?

Web vs. programmable Web

- WEB: serves data (HTML) to human
  - *The Web is full of data: book information, messages, etc.*
  - *It is also full of services: search engines, online stores, etc.*

  ->Rather than installing all this data and all these programs on your own computer, you install one program—a web browser—and access the data and services through it.
- Programmable web: serves data (XML) to machines
  - *The programmable web is just the same. The main difference is that instead of arranging its data in attractive HTML pages, the programmable web usually serves plain XML documents*
  - *It is not necessarily for human consumption. Its data is intended as input to a software program.*

Web vs. programmable Web

- There are a large number of formats available, including HTML, XML, JSON, RSS, Atom, CSV and many other custom formats.
  - **Plain-Old-XML** (**POX**) and **JavaScript Object Notation** (**JSON**) are often popular.
- In the same way documents are inhabitants of the Web, services are inhabits of the programmable web.
- Services can be classified :
  - By the technologies they use (URIs, SOAP etc.),
  - By the underlying architectures and design philosophies.
- What they all have in commune : All are based on **HTTP**.

HTTP, the protocol that all web services have in common.

Overview of HTTP:

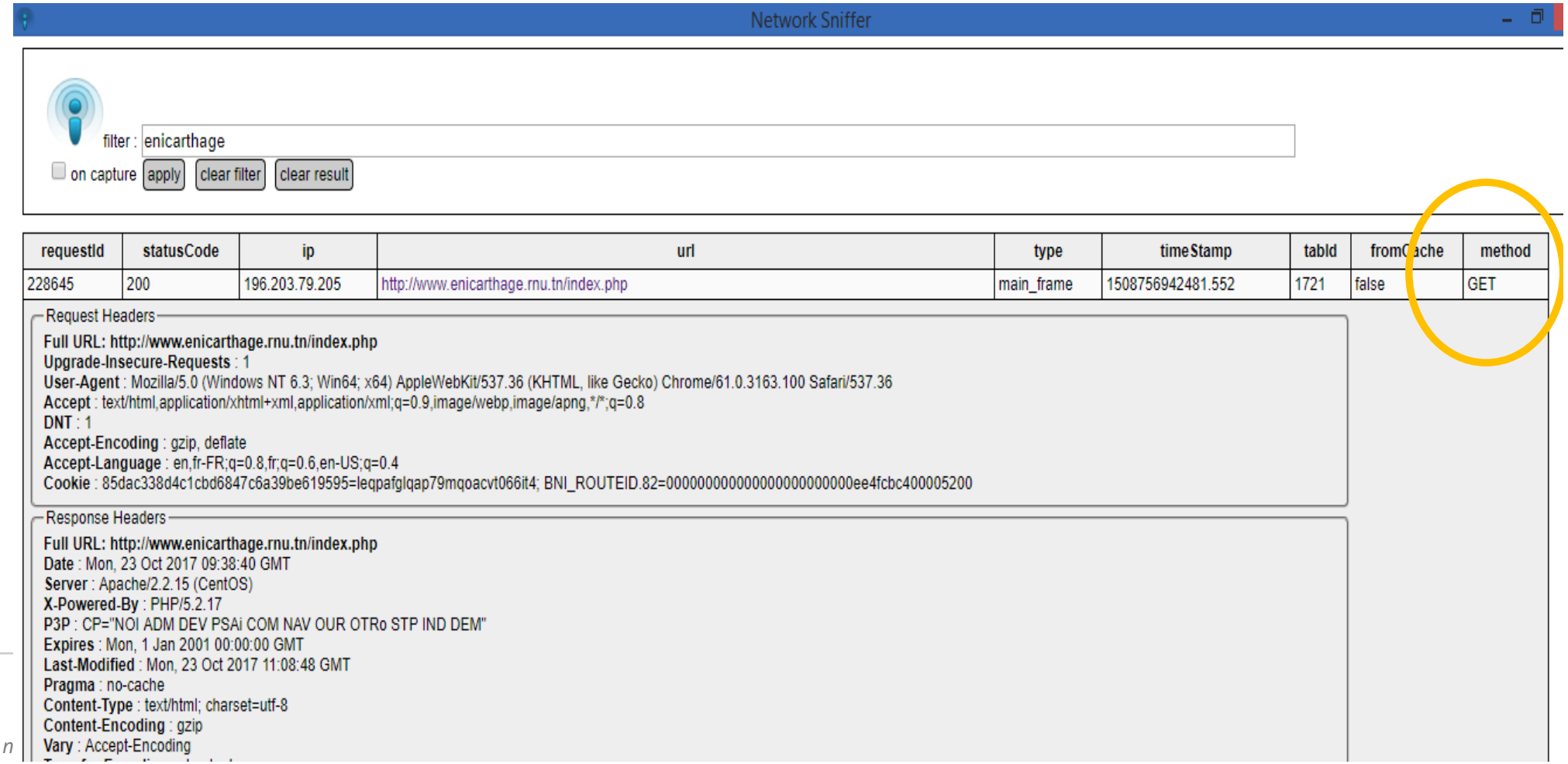
- HTTP is a document-based protocol, in which the client puts a document in an envelope and sends it to the server. The server returns the favor by putting a response document in an envelope and sending it to the client.

- HTTP has strict standards for what the envelopes should look like, but it doesn't much care what goes inside.

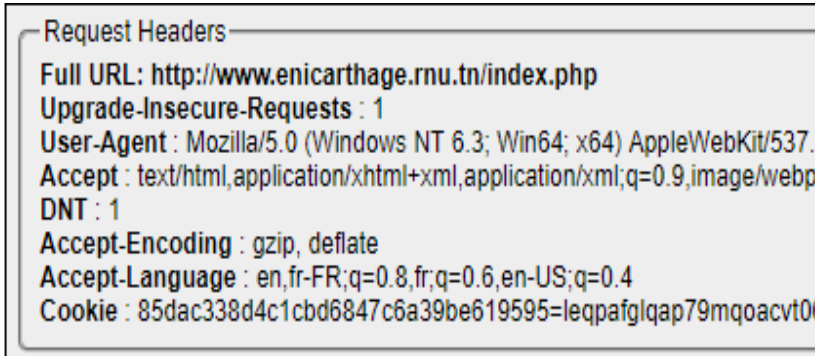
HTTP message structure

Request Line	Status Line
General Header	
Request Header	Response Header
Entity Header	
Empty Line	
Message Body (entity body or encoded entity body	

- Example :An HTTP GET request http://www.enicarthage.rnu.tn/index.php



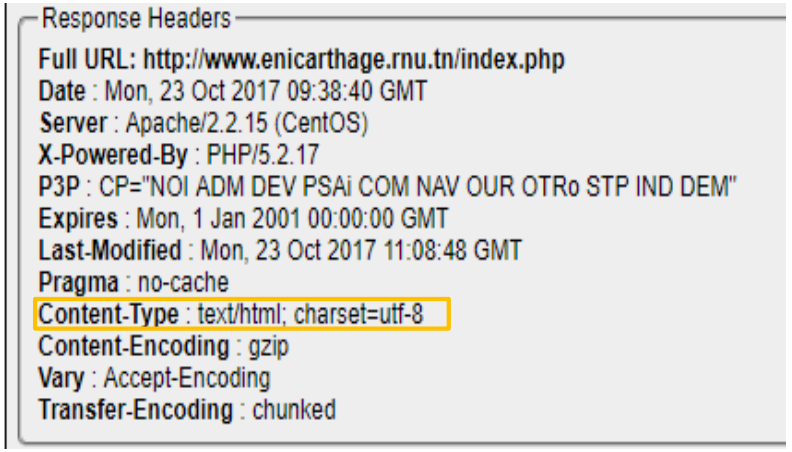
HTTP request (Web document)



- **HTTP method** : In this request, the method is "GET."
  - In REST, it is called the "**HTTP verb**" **or "HTTP action**."
  - The name of the HTTP method is like a method name in a programming language: it indicates how the client expects the server to process this envelope. In this case, the client (web browser) is trying to **GET** some information from the server (www.enicarthage.rnu.tn).
- **Path** : This is the portion of the URI to the right of the hostname.
  - In this example : /index.php
- **Request headers** : key-value pairs that act like informational stickers.
  - Examples : Host, User-Agent, Accept, and so on.
- **Entity-body** : also called the document or **representation**: This is the document that is inside the envelope.
  - *This particular request has no entity body, which means the envelope is empty! This is typical for a GET request, where all the information needed to complete the request is in the path and the headers.*

neila.benlakhal@gmail.com

HTTP response (Web document)



- The response can be divided into three parts:
  - **HTTP response code** : This numeric code tells the client the request status; (200: “OK”).
  - **Response headers** : These are informational stickers slapped onto the envelope.
  - **Entity-body or representation** :
    - This is the document inside the envelope. The rest of the response is just an envelope with stickers on it, telling the web browser how to deal with the document.
    - The most important is : Content-Type In this case, the media type is text/html. This lets the web browser render the entity-body as an HTML document: **a web page.**

HTTP Request

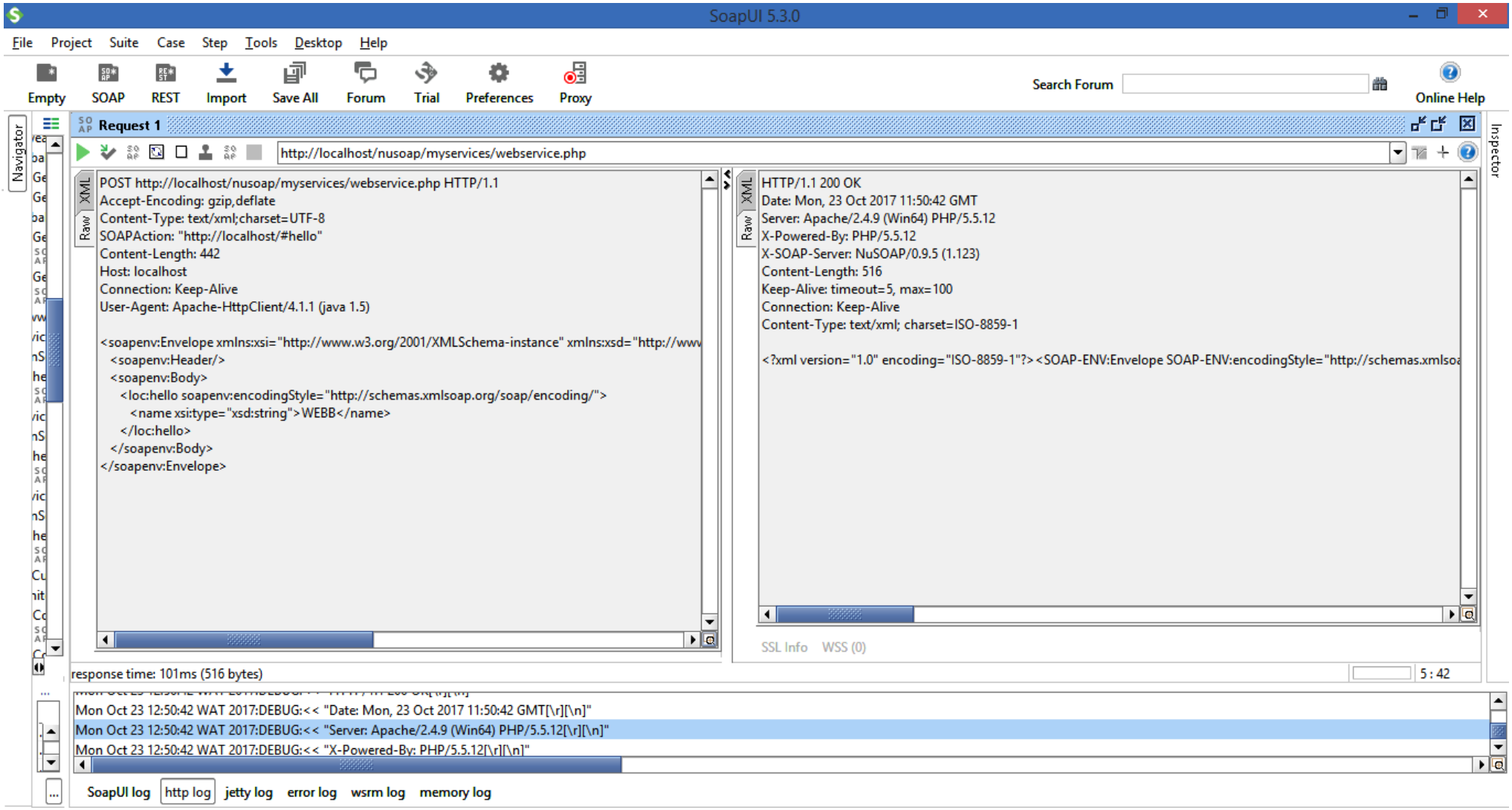
- How to tell the server what the client want to do ?
- The HTTP Request can be :
  - Call for a Web service:
    - __REST
    - __SOAP
  - Call for a Web document.
  - Its depends on the **HTTP method (verb)**

Method information of a service

- Can use either the HTTP standardized methods or specific method :
  - Use the ***HTTP most common methods*** : GET, PUT, DELETE, and POST:
    - GET (Query the state, idempotent, can be cached)
    - POST (Update a resource or create child resource)
    - PUT (Transfer the state on existing/new resource)
    - DELETE (Delete a resource)
  - Use ***programmer defined methods*** usually in the **URI path** or the **request document** (e.g., search, update, lookup, etc.)

Method information : SOAP-style

- Typical SOAP service keeps its method information in the e**ntity-body** and in the **HTTP header**.
- The URI never vary in a SOAP service.
- The method information goes into :
  - The request headers : (soapaction)
  - The entity-body : the SOAP envelope body.
  - ➔ **SOAP services : RPC-style architecture.**


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Navigator
Request 1
http://localhost/nusoap/myservices/webservice.php
XML Raw
POST http://localhost/nusoap/myservices/webservice.php HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: "http://localhost/#hello"
Content-Length: 442
Host: localhost
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www
<soapenv:Header/>
<soapenv:Body>
<loc:hello soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<name xsi:type="xsd:string">WEBB</name>
</loc:hello>
</soapenv:Body>
</soapenv:Envelope>
XML Raw
HTTP/1.1 200 OK
Date: Mon, 23 Oct 2017 11:50:42 GMT
Server: Apache/2.4.9 (Win64) PHP/5.5.12
X-Powered-By: PHP/5.5.12
X-SOAP-Server: NuSOAP/0.9.5 (1.123)
Content-Length: 516
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/xml; charset=ISO-8859-1
<?xml version="1.0" encoding="ISO-8859-1"?><SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoa
SSL Info WSS (0)
response time: 101ms (516 bytes)
5 : 42
Inspector
Mon Oct 23 12:50:42 WAT 2017:DEBUG:<< "Date: Mon, 23 Oct 2017 11:50:42 GMT[\r][\n]"
Mon Oct 23 12:50:42 WAT 2017:DEBUG:<< "Server: Apache/2.4.9 (Win64) PHP/5.5.12[\r][\n]"
Mon Oct 23 12:50:42 WAT 2017:DEBUG:<< "X-Powered-By: PHP/5.5.12[\r][\n]"
SoapUI log
http log
jetty log
error log
wsrm log
memory log

Method information : REST-style

- REQUEST:

GET http://api.fixer.io/latest?symbols=USD%2CGBP HTTP/1.1

- The HTTP method is : **GET**
- The requested resource is in the HTTP header:

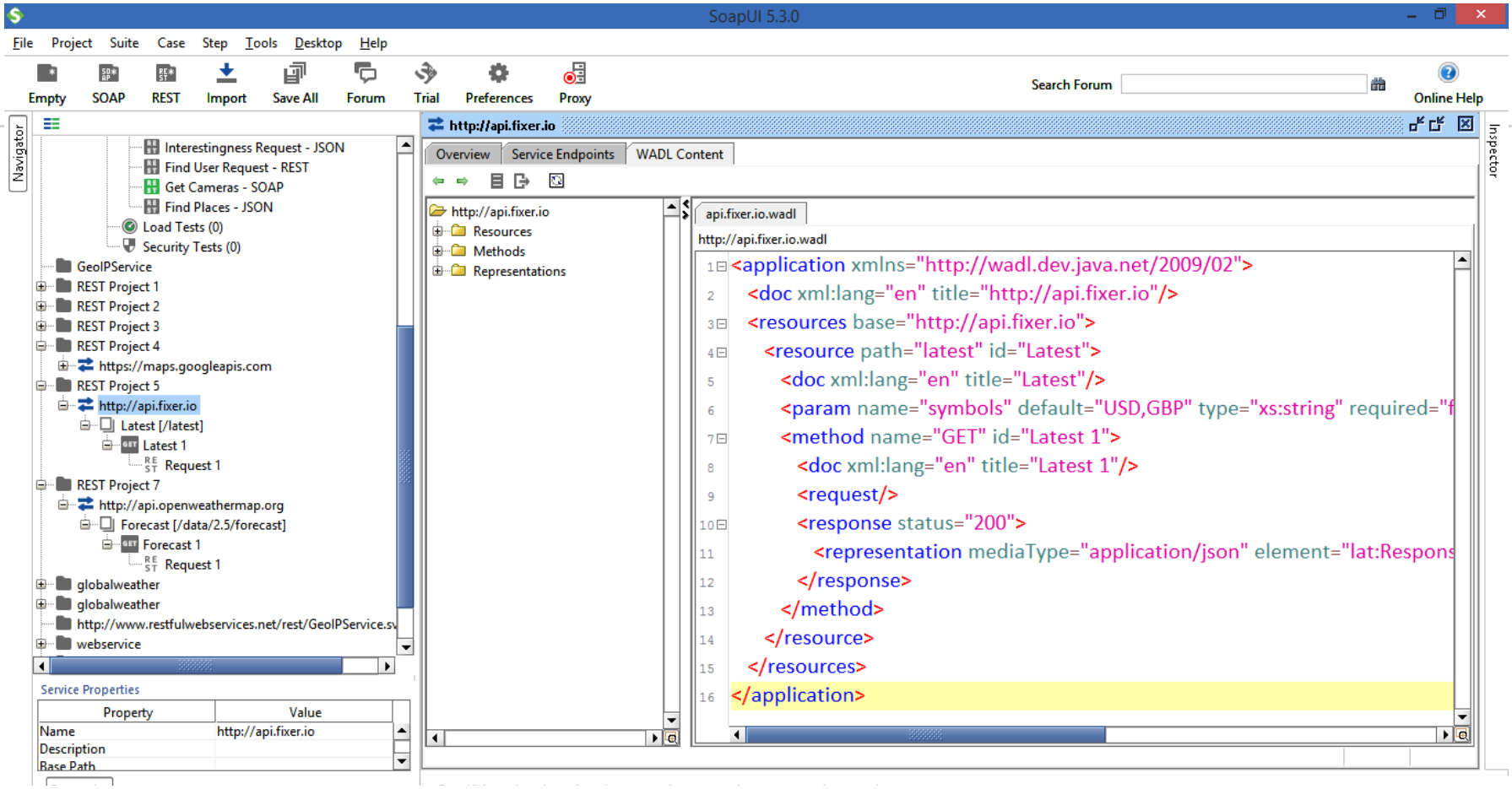
  http://api.fixer.io/latest?symbols=USD%2CGBP (URI)

→ **RESTful service.**

REST service WADL

- The Web Application Description Language (WADL) is an XML vocabulary used to describe RESTful web services. As with WSDL, a generic client can load a WADL file and be immediately equipped to access the full functionality of the corresponding web service.
- Since RESTful services have simpler interfaces, WADL is not nearly as necessary to these services as WSDL is to RPC-style SOAP services.
- WADL is not a standard.

WADL example (generated in SOAPUI when service URI imported in SOAPUI)



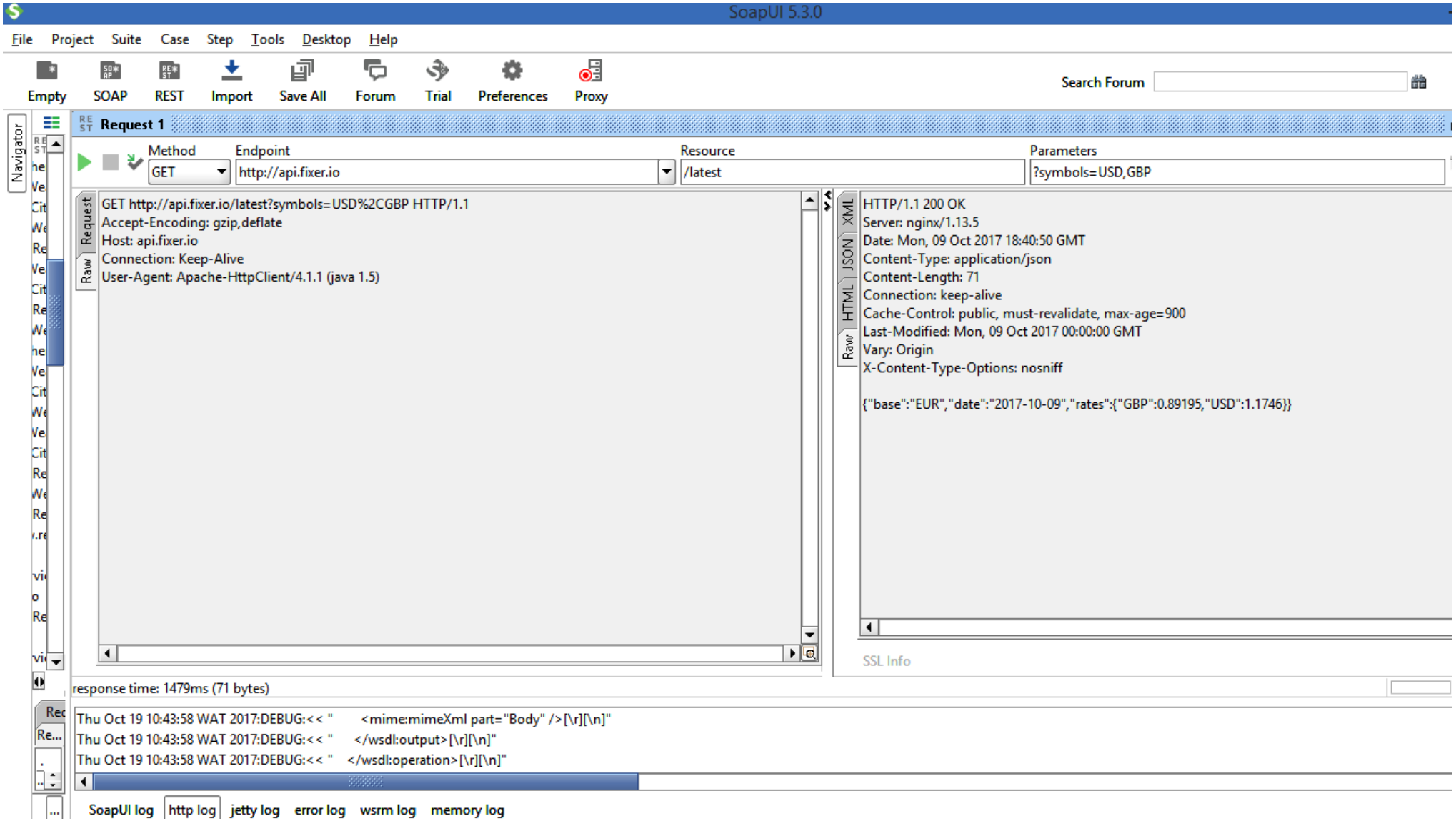
https://www.w3.org/Submission/wadl/

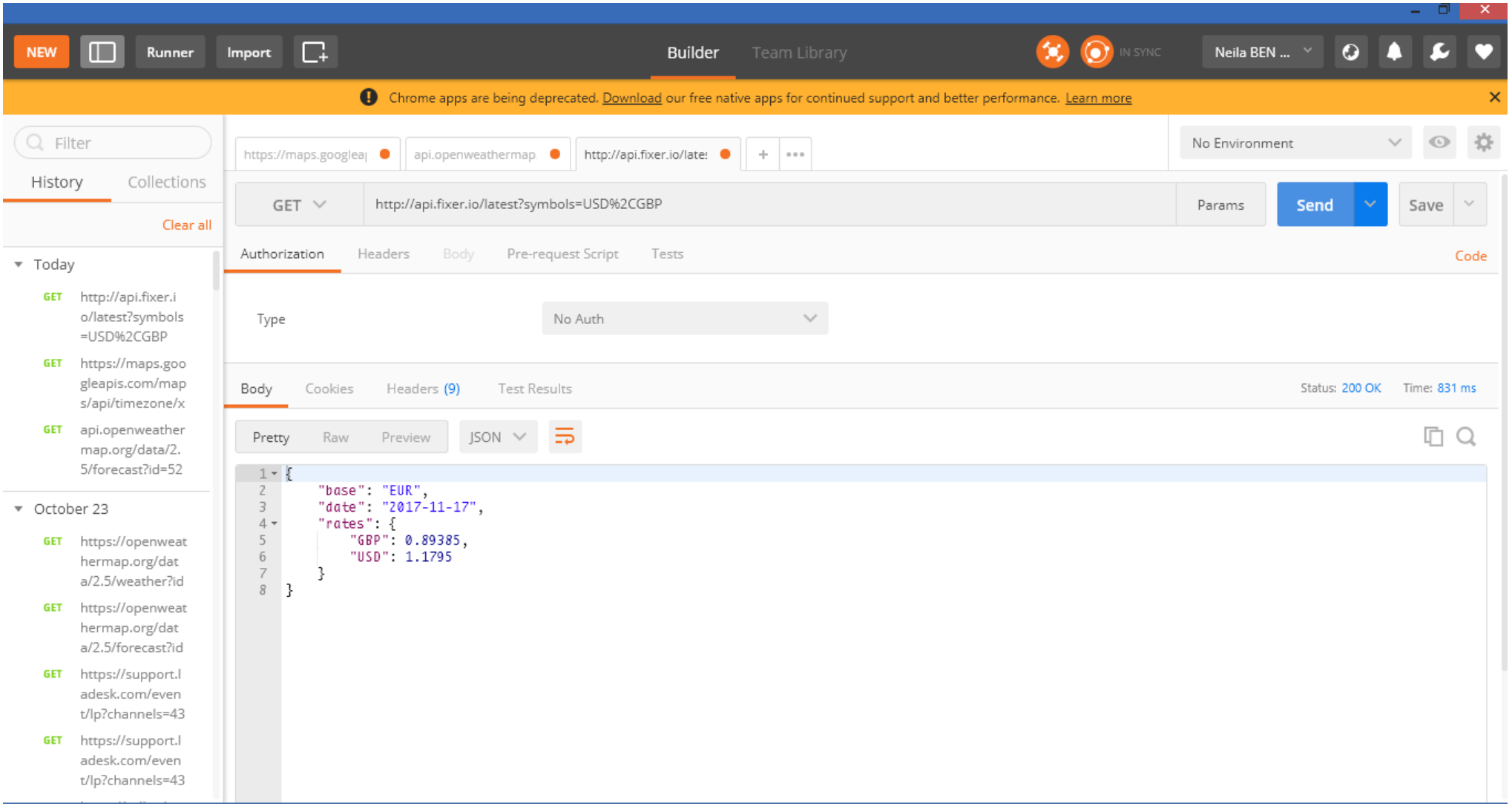
WADL structure

- The ***<application>*** element is the root for the WADL metadata.
- It contains :
  - 0..N ***<doc>*** elements (documentations) (optional)
  - 0..1 ***<grammars>*** elements : It acts as the container for schemas that describe the service's resource. The ***<grammars>*** element typically refers to XML schema.
  - 0..N of the following :
    - ***<resources>*** element that contains 0..N ***<resource>***
    - ***<resource>*** element contains ***<method>***
    - ***<method>*** contains : ***<request>*** and ***<response>*** elements
    - ***<response>*** element contains ***<representation>*** elements and eventually ***<fault>*** element.
    - ***<param>*** elements placed in ***<method>*** or ***<resource>*** element.

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://research.sun.com/wadl/2006/10 wadl.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ex="http://www.example.org/types"
    xmlns="http://research.sun.com/wadl/2006/10">
    <grammars>
      <include href="ticker.xsd"/>
    </grammars>
    <resources base="http://www.example.org/services/">
      <resource path="getStockQuote">
        <method name="GET">
          <request>
            <param name="symbol" style="query" type="xsd:string"/>
          </request>
          <response>
            <representation mediaType="application/xml" element="ex:quoteResponse"/>
            <fault status="400" mediaType="application/xml" element="ex:error"/>
          </response>
        </method>
      </resource>
    </resources>
</application>
```

GET http://www.example.org/services/getStockQuote/symbol?


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Navigator
Request 1
Method
GET
Endpoint
http://api.fixer.io
Resource
/latest
Parameters
?symbols=USD,GBP
Request
Raw
GET http://api.fixer.io/latest?symbols=USD%2CGBP HTTP/1.1
Accept-Encoding: gzip,deflate
Host: api.fixer.io
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
XML
JSON
HTML
Raw
HTTP/1.1 200 OK
Server: nginx/1.13.5
Date: Mon, 09 Oct 2017 18:40:50 GMT
Content-Type: application/json
Content-Length: 71
Connection: keep-alive
Cache-Control: public, must-revalidate, max-age=900
Last-Modified: Mon, 09 Oct 2017 00:00:00 GMT
Vary: Origin
X-Content-Type-Options: nosniff
{"base":"EUR","date":"2017-10-09","rates":{"GBP":0.89195,"USD":1.1746}}
SSL Info
response time: 1479ms (71 bytes)
Thu Oct 19 10:43:58 WAT 2017:DEBUG:<< " <mime:mimeXml part="Body" />[\r][\n]"
Thu Oct 19 10:43:58 WAT 2017:DEBUG:<< " </wsdl:output>[\r][\n]"
Thu Oct 19 10:43:58 WAT 2017:DEBUG:<< " </wsdl:operation>[\r][\n]"
SoapUI log
http log
jetty log
error log
wsrm log
memory log


NEW
Runner
Import
Builder
Team Library
IN SYNC
Neila BEN ...
Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more
Filter
History
Collections
Clear all
Today
GET http://api.fixer.io/latest?symbols=USD%2CGBP
GET https://maps.googleapis.com/maps/api/timezone/x
GET api.openweathermap.org/data/2.5/forecast?id=52
October 23
GET https://openweathermap.org/data/2.5/weather?id
GET https://openweathermap.org/data/2.5/forecast?id
GET https://support.ladesk.com/event/lp?channels=43
GET https://support.ladesk.com/event/lp?channels=43
https://maps.googlea
api.openweathermap
http://api.fixer.io/late
No Environment
GET
http://api.fixer.io/latest?symbols=USD%2CGBP
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Code
Type
No Auth
Body
Cookies
Headers (9)
Test Results
Status: 200 OK
Time: 831 ms
Pretty
Raw
Preview
JSON
{
"base": "EUR",
"date": "2017-11-17",
"rates": {
"GBP": 0.89385,
"USD": 1.1795
}
}

Method information : REST-RPC style service

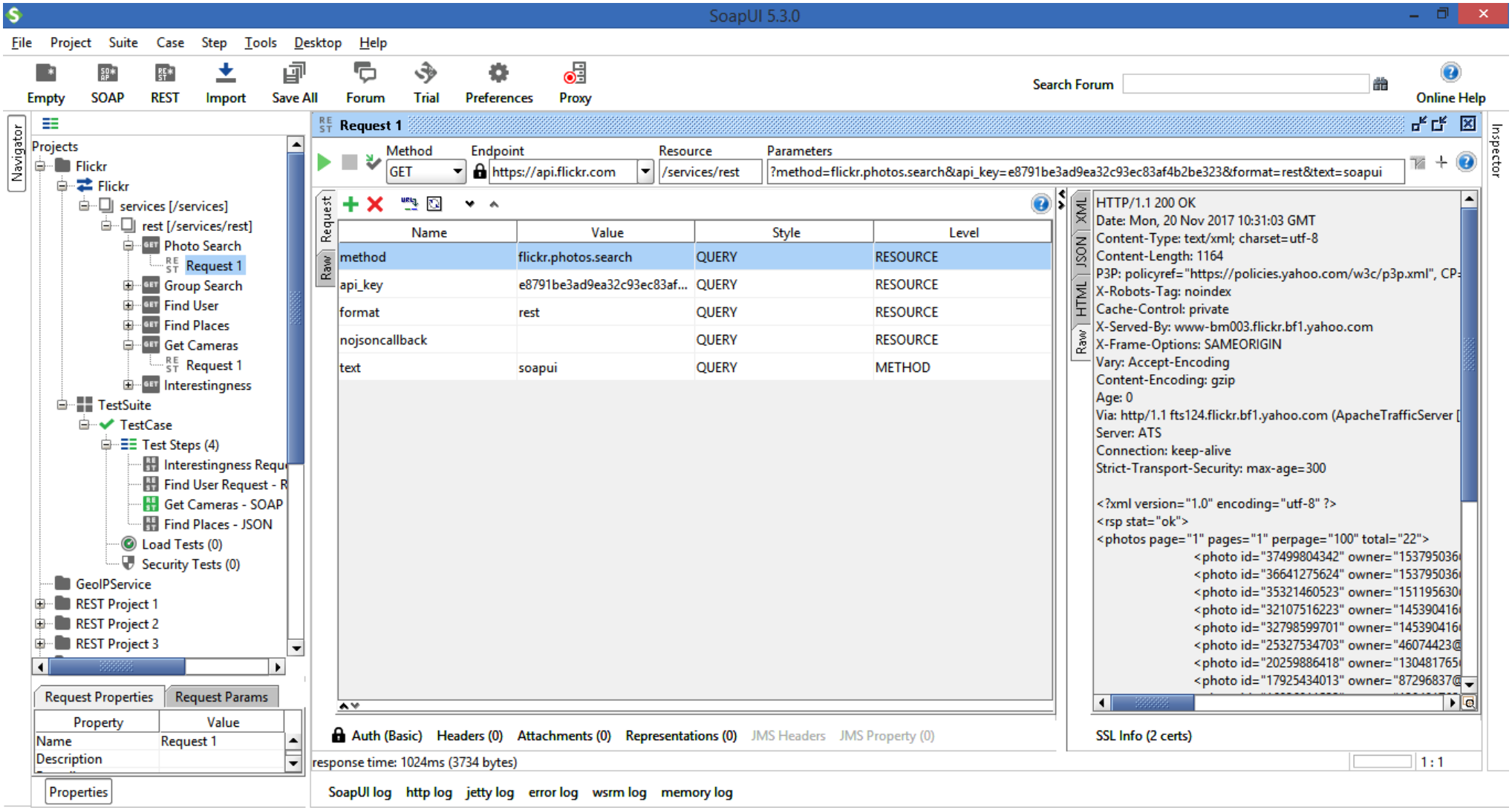
- https://www.flickr.com/services/api/

GET https://api.flickr.com/services/rest?method=flickr.photos.search&api_key=XXX&format=rest&text=soapui

  - The method is flickr.photos.search
  - Other methods: flickr.people.findByEmail, etc.

- Here, Flickr is sticking the method information in the **method** query variable and is ***not*** really using the HTTP method.

→**Hybrid REST-RPC oriented service**


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Navigator
Projects
Flickr
Flickr
services [/services]
rest [/services/rest]
Photo Search
Request 1
Group Search
Find User
Find Places
Get Cameras
Request 1
Interestingness
TestSuite
TestCase
Test Steps (4)
Interestingness Requ
Find User Request - R
Get Cameras - SOAP
Find Places - JSON
Load Tests (0)
Security Tests (0)
GeoIPService
REST Project 1
REST Project 2
REST Project 3
Request Properties
Request Params
Property
Value
Name
Request 1
Description
Properties
Request 1
Method
Endpoint
Resource
Parameters
GET
https://api.flickr.com
/services/rest
?method=flickr.photos.search&api_key=e8791be3ad9ea32c93ec83af4b2be323&format=rest&text=soapui
Request
Raw
Name
Value
Style
Level
method
flickr.photos.search
QUERY
RESOURCE
api_key
e8791be3ad9ea32c93ec83af...
QUERY
RESOURCE
format
rest
QUERY
RESOURCE
nojsoncallback
QUERY
RESOURCE
text
soapui
QUERY
METHOD
Auth (Basic)
Headers (0)
Attachments (0)
Representations (0)
JMS Headers
JMS Property (0)
response time: 1024ms (3734 bytes)
XML
JSON
HTML
Raw
HTTP/1.1 200 OK
Date: Mon, 20 Nov 2017 10:31:03 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 1164
P3P: policyref="https://policies.yahoo.com/w3c/p3p.xml", CP
X-Robots-Tag: noindex
Cache-Control: private
X-Served-By: www-bm003.flickr.bf1.yahoo.com
X-Frame-Options: SAMEORIGIN
Vary: Accept-Encoding
Content-Encoding: gzip
Age: 0
Via: http/1.1 fts124.flickr.bf1.yahoo.com (ApacheTrafficServer
Server: ATS
Connection: keep-alive
Strict-Transport-Security: max-age=300
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
<photos page="1" pages="1" perpage="100" total="22">
<photo id="37499804342" owner="153795036
<photo id="36641275624" owner="153795036
<photo id="35321460523" owner="151195630
<photo id="32107516223" owner="145390416
<photo id="32798599701" owner="145390416
<photo id="25327534703" owner="46074423@
<photo id="20259886418" owner="130481765
<photo id="17925434013" owner="87296837@
SSL Info (2 certs)
1 : 1
Inspector
SoapUI log
http log
jetty log
error log
wsrm log
memory log

Types of service in the programmable Web

- REST-style resource oriented service
  - Static web sites
  - REST web services : api.fixer.io etc.
- RPC-style service
  - SOAP web services
- REST-RPC hybrid service
  - Most web applications
  - Many APIs e.g., Flickr web API etc. (programmable Web)

Examples of (good)RESTful, resource-oriented web services

- Most read-only web services that don't use SOAP
- Static web sites
- Many web applications, especially read-only ones like search engines

REST architectural style principles

- The REST architectural style is based on 4 principles:
  1. ***Resource Identification through URI,***
  2. ***Uniform Interface for all resources,***
  3. ***Self-Descriptive Message representations and***
  4. ***Stateful interactions through hyperlinks.***

REST principles

1. *Resource identification through URI.*

- A RESTful Web service exposes a set of resources which identify the targets of the interaction with its clients.
- Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

Representational State Transfer?

◉Why is it called **Representational State Transfer?**

- The Web is comprised of **resources**.
- A **resource** is any item of interest : A resource is anything that’s important enough to be referenced as a thing in itself.

Resource ?

- A **resource** is something that can be stored on a computer and represented as a stream of bits: *a document, a row in a database, or the result of running an algorithm*.
- A **resource** may be a physical object like *an apple, or an abstract concept like courage, etc.*
- **What makes a resource a resource in REST**?
  - It has to have at least one **URI.** The **URI** is the name and address of a resource.
  - If a piece of information doesn't have a URI, it's not a resource and it's not really on the Web, except as a bit of data describing some other resource.

Example of resources | URI

- The latest version of the software release (identified by the URI: http://www.example.com/software/releases/latest.tar.gz)
- Some information about jellyfish (http://www.example.com/search/Jellyfish)
- The next prime number after 1024 (http://www.example.com/nextprime/1024)
  The next five prime numbers after 1024 (http://www.example.com/next-5-primes/1024)
- Etc.

URI ?

- URI - **U**niform **R**esource **I**dentifier
- Internet Standard for resource naming and identification (originally from 1994, revised until 2014)
- A compact sequence of characters that identifies an abstract or physical ***resource***.

- Examples:

http://tools.ietf.org/html/rfc3986

Why is it called Representational State Transfer?

- Boeing may define a 787 ***resource***.
- Clients may access that ***resource*** with this URL: http://www.boeing.com/aircraft/787
- A ***representation*** of the ***resource*** is returned (e.g., Boeing787.html).
- The ***representation*** places the client application in a ***state***.
- The result of the client traversing a hyperlink in Boeing787.html is another **resource** accessed.
- The new ***representation*** places the client application into yet another ***state***.

→The client application changes (transfers) ***state*** with each ***resource representation***.

REST principles

2. *Uniform interface*.

- Resources are manipulated using a fixed set of operations : the **HTTP verbs**.
- The most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE.
- These correspond to create, read, update, and delete (or CRUD) operations.
- There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Methods : examples

Examples :

- The POST verb is most-often utilized to **create** new resources

POST http://www.example.com/customers/12345/orders

- The HTTP GET method is used to **read** (or retrieve) a representation of a resource

GET http://www.example.com/customers/12345

- For more examples :

http://www.restapitutorial.com/lessons/httpmethods.html

REST principles

3. *Self-Descriptive Message Representations*

Resources are decoupled from their representation so that their content can be accessed in a variety of formats (e.g., HTML, XML, plain text, PDF, JPEG, etc.

REST principles

4. *Stateful interactions through hyperlinks.*

Every interaction with a resource is ***stateless***, i.e., request messages are self-contained.
Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, e.g., URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.

Statelessness?

- Means that every HTTP request happens in complete isolation: When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request.
  - → The server never relies on information from previous requests.
  - → If that information was important, the client would have sent it again in this request.

How to build a REST-style service client ?

- Service request: is an HTTP request
  - Need for an ***HTTP library***
- Service response : is an HTTP response with an entity-body where the entity body is generally an XML/Json response.
- Need for :
  - ***XML parser***
  - ***Decode Json response***

REST-style service client

- As every REST-style service is an HTTP request, every programming language can handle it through an HTTP library
- Every modern programming language has one or more libraries for making HTTP requests.
- To build a fully general web service client you need an HTTP library with these features:
  - Support at least the 5 main HTTP methods: GET, HEAD, POST, PUT and DELETE
  - Support HTTPS and SSL certificate validation. Web services, like web sites, use HTTPS to secure communication with their clients.
  - Allow the programmer to customize the data sent as the entity-body of a PUT or POST request.
  - Allow the programmer to customize a request's HTTP headers.
  - Give the programmer access to the response code and headers of an HTTP response; not just access to the entity-body.
  - Be able to communicate through an HTTP proxy.

Examples of HTTP libraries

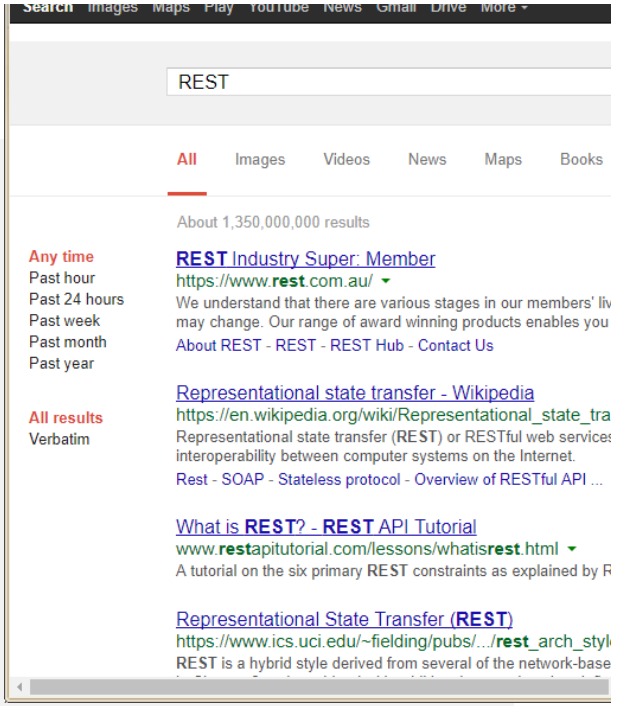
- PHP: **libcurl (**PHP comes with a binding to the C library libcurl)
- JS: XMLHttpRequest (HTTP client library for JavaScript)
- C#: System.Web.HTTPWebRequest
- Java: HttpClient (The Java standard library comes with an HTTP client, java.net.HttpURLConnection/Restlet,…)
- Python: httplib2
- Etc.

cURL/PHP

- PHP supports libcurl, a library that allows to connect and communicate to many different types of servers with many different types of protocols.
- libcurl supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading (this can also be done with PHP's ftp extension), HTTP form based upload, proxies, cookies, and user+password authentication.
- **PHP/cURL**: The module for PHP that makes it possible for PHP programs to use libcurl.

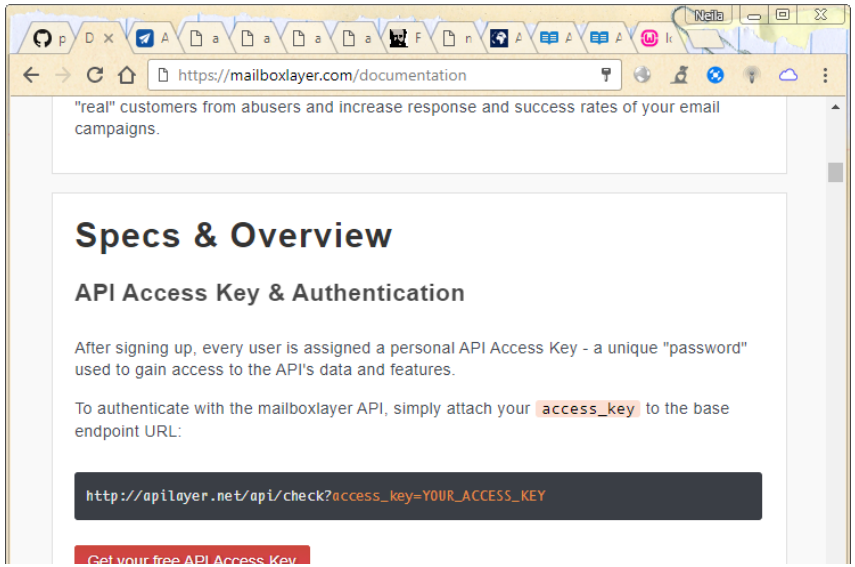
Retrieving Web page in PHP Using Curl

```php
<?php
//step1 Initialize a curl session use curl_init().
$cSession = curl_init();
//step2 set option for CURLOPT_URL.
//This value is the URL which we are sending the request to.
//Append a search term "REST" using parameter "q=".
//Set option for CURLOPT_RETURNTRANSFER:
//true will tell curl to return the string instead of printing it out.
//Set option for CURLOPT_HEADER,
//false will tell curl to ignore the header in the return value.
curl_setopt($cSession,CURLOPT_URL,"http://www.google.com/search?q=REST");
curl_setopt($cSession,CURLOPT_RETURNTRANSFER,true); curl_setopt($cSession,CURLOPT_HEADER, false);
//step3: Execute the curl session using curl_exec().
$result=curl_exec($cSession);
//step4: Close the curl session we have created.
curl_close($cSession);
//step5: Output the return string.
echo $result; ?>
```



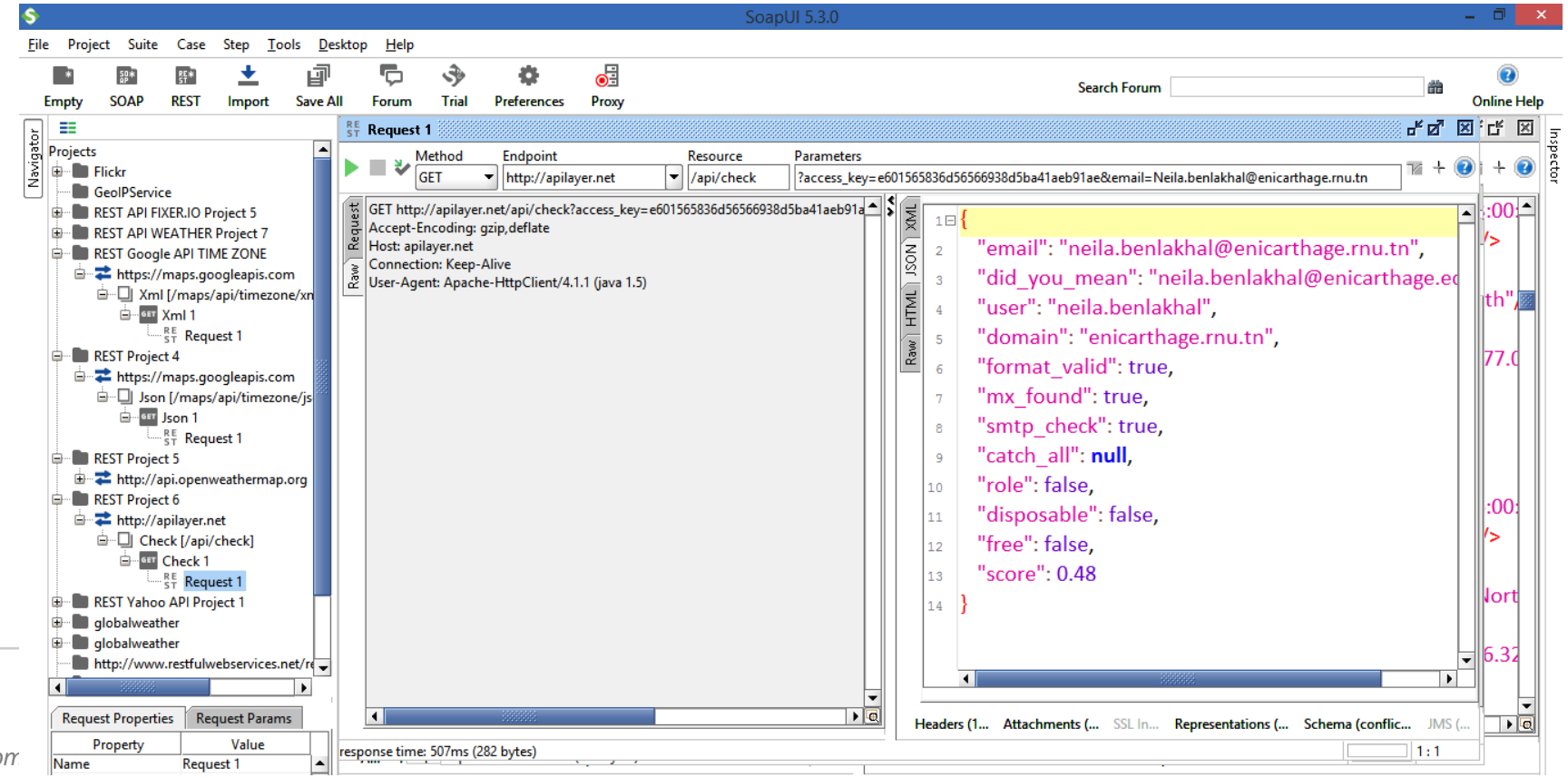
Curl client/restful web services (Json)

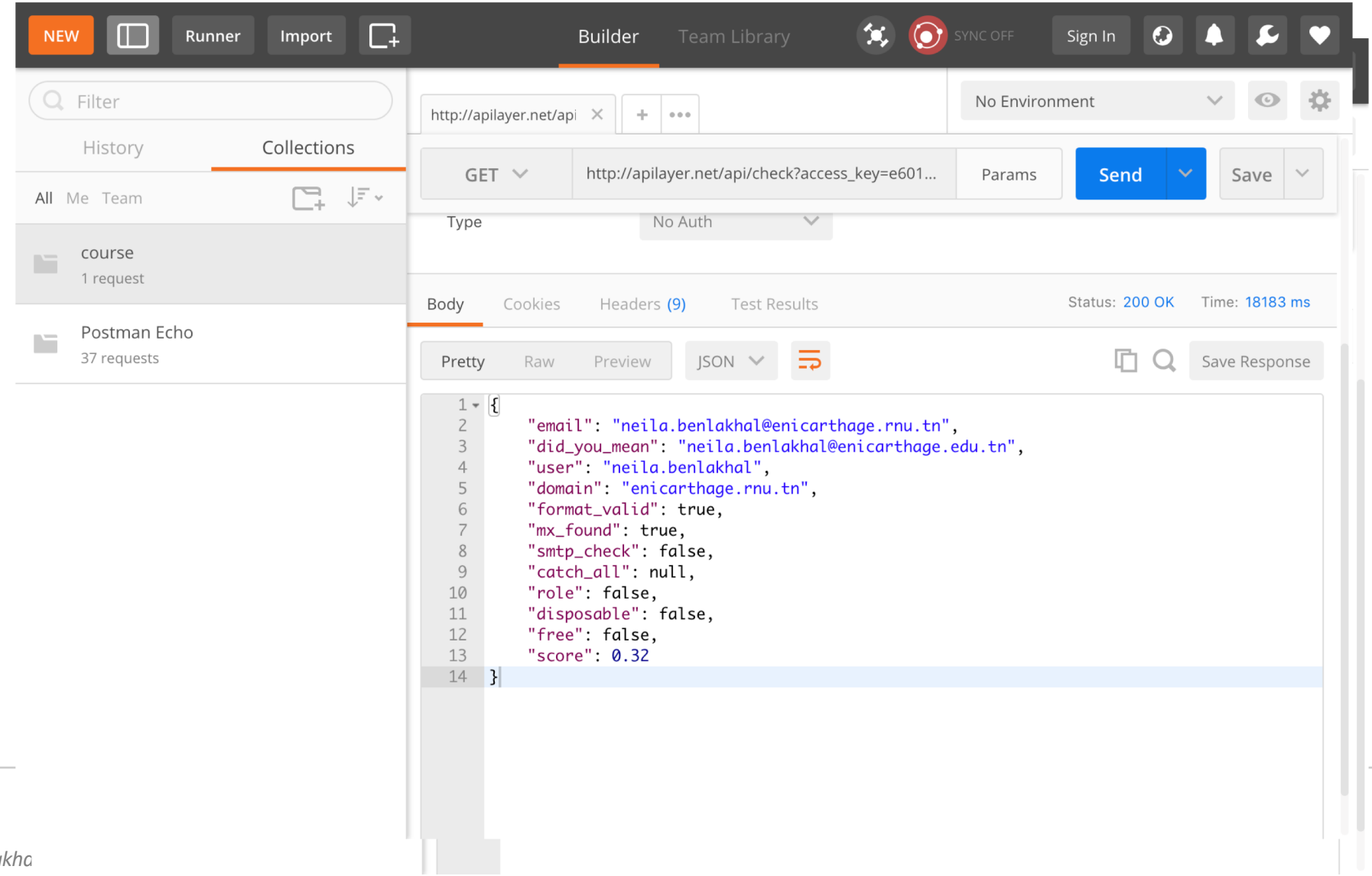
- https://mailboxlayer.com/documentation
- Mailboxlayer offers a simple REST-based JSON API enabling you to thoroughly check and verify email addresses right at the point of entry into your system.



Test via SOAPUI

- GET http://apilayer.net/api/check?access_key=e601565836d56566938d5ba41aeb91ae&email=Neila.benlakhal@enicarthage.rnu.tn




NEW
Runner
Import
Builder
Team Library
SYNC OFF
Sign In
Filter
History
Collections
All Me Team
course
1 request
Postman Echo
37 requests
http://apilayer.net/api
No Environment
GET
http://apilayer.net/api/check?access_key=e601...
Params
Send
Save
Type
No Auth
Body
Cookies
Headers (9)
Test Results
Status: 200 OK
Time: 18183 ms
Pretty
Raw
Preview
JSON
Save Response
{
"email": "neila.benlakhal@enicarthage.rnu.tn",
"did_you_mean": "neila.benlakhal@enicarthage.edu.tn",
"user": "neila.benlakhal",
"domain": "enicarthage.rnu.tn",
"format_valid": true,
"mx_found": true,
"smtp_check": false,
"catch_all": null,
"role": false,
"disposable": false,
"free": false,
"score": 0.32
}

Test via a client (Php/Curl)

```php
<?php // set API Access Key
$access_key = 'e601565836d56566938d5ba41aeb91ae';
// set email address $email_address =
'neila.benlakhal@enicarthage.rnu.tn';
// Initialize CURL:
$ch =
curl_init('http://apilayer.net/api/check?access_key='.$access_key.'&ema
il='.$email_address.'');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
// Store the data:
$json = curl_exec($ch);
curl_close($ch);
// Decode JSON response:
$validationResult = json_decode($json, true);
var_dump(json_decode($json, true));
// Access and use your preferred validation result objects
print "format is valid? " .$validationResult['format_valid']."<br/>";
print "SMTP?".$validationResult['smtp_check']."<br/>";
print "Domain? ".$validationResult['domain'];
?>
```

Pour supporter les autres méthodes de HTTP/client Curl:

```php
<?php
function callAPI($method, $url, $data){
   $curl = curl_init();
   switch ($method){
      case "POST":curl_setopt($curl, CURLOPT_POST, 1);
         if ($data)
            curl_setopt($curl, CURLOPT_POSTFIELDS, $data); break;
      case "PUT":curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
         if ($data)
            curl_setopt($curl, CURLOPT_POSTFIELDS, $data);break;
      default:
         if ($data)
            $url = sprintf("%s?%s", $url, http_build_query($data));
   }// OPTIONS:
   curl_setopt($curl, CURLOPT_URL, $url);
   curl_setopt($curl, CURLOPT_HTTPHEADER, array(
      'APIKEY: 111111111111111111111','Content-Type: application/json'));
   curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
   curl_setopt($curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
   // EXECUTE:
   $result = curl_exec($curl);
   if(!$result){die("Connection Failure");}
   curl_close($curl);
   return $result;
}?>
```

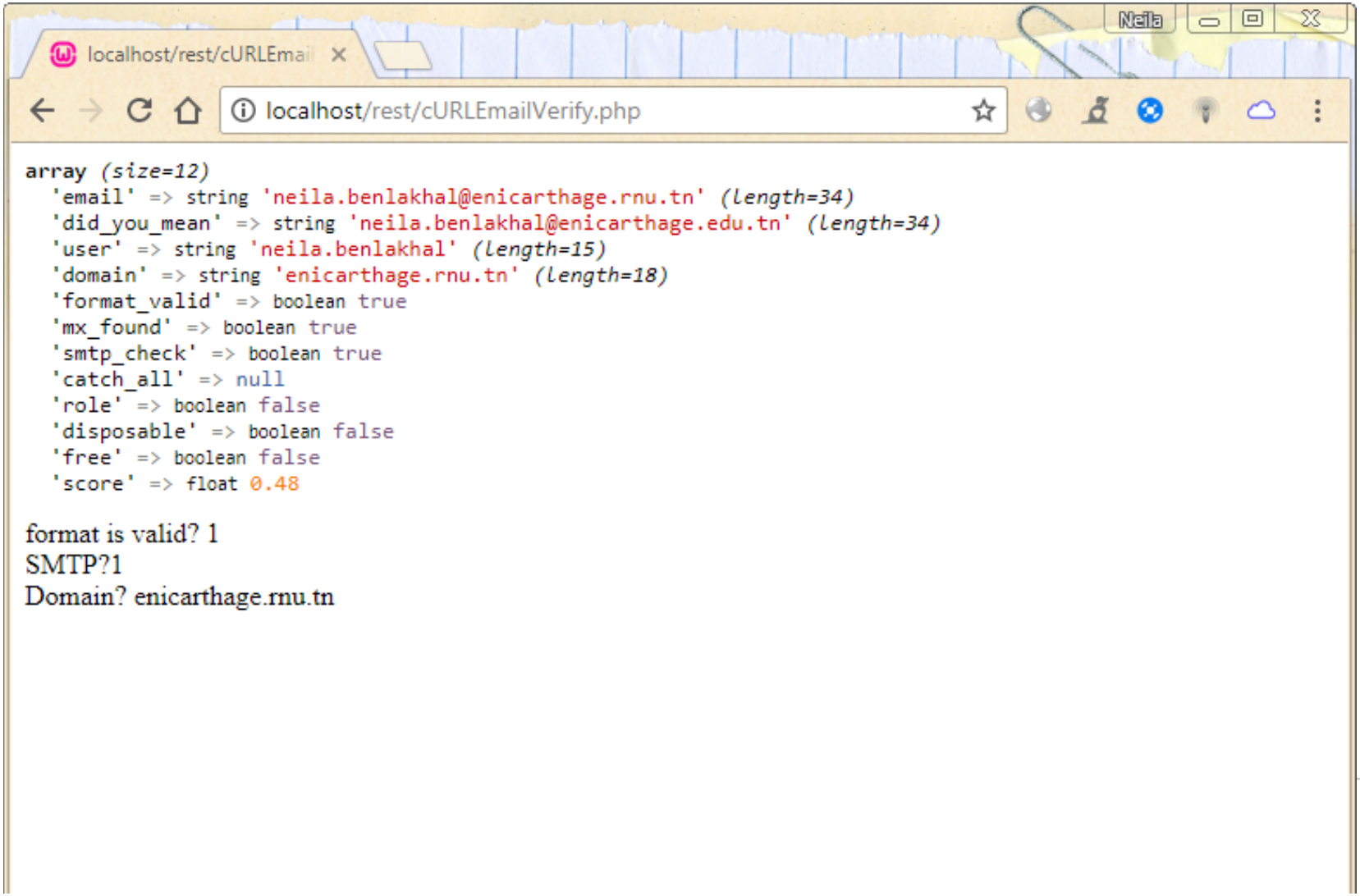
https://www.weichieprojects.com/blog/curl-api-calls-with-php/#!

cURL GET request

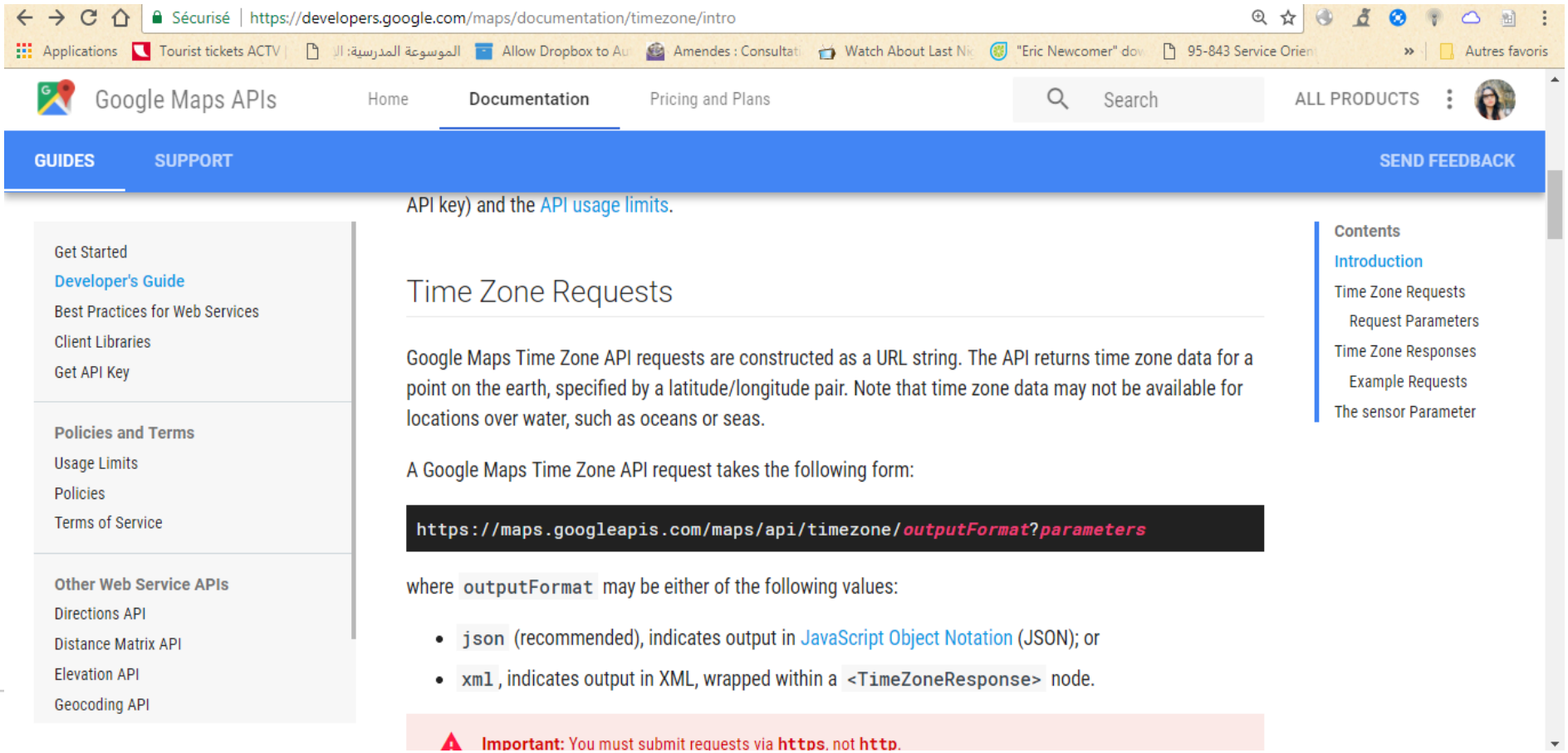
```
<?php
$get_data = callAPI('GET',
'https://api.example.com/get_url/'.$user['User']['customer_id'
], false);
$response = json_decode($get_data, true);
$errors = $response['response']['errors'];
$data = $response['response']['data'][0];
?>
```

cURL POST request

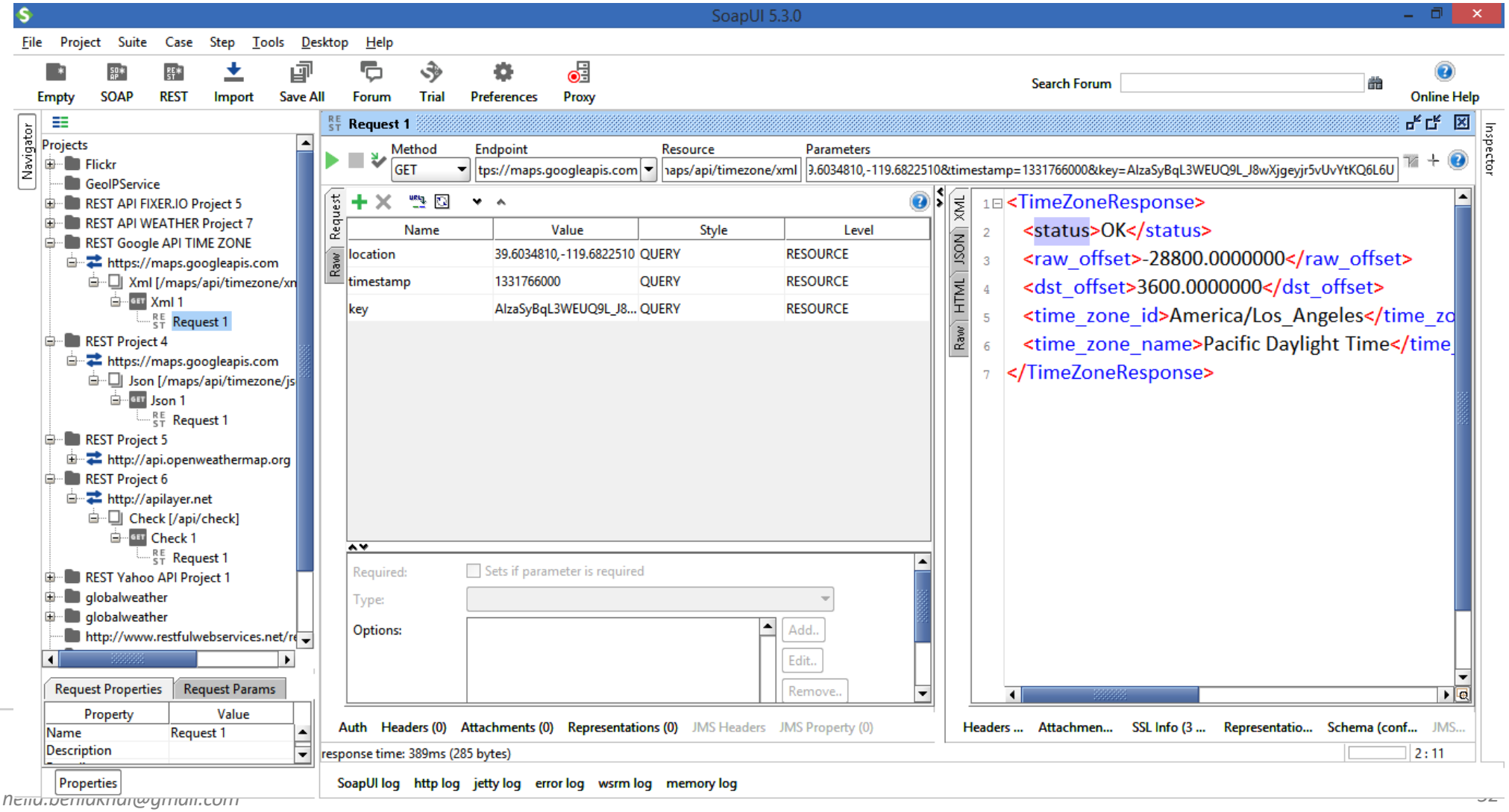
```php
<?php
$data_array =  array(
      "customer"          => $user['User']['customer_id'],
      "payment"           => array(
            "number"            => $this->request->data['account'],
            "routing"           => $this->request->data['routing'],
            "method"            => $this->request->data['method']
      ),
);
$make_call = callAPI('POST', 'https://api.example.com/post_url/',
json_encode($data_array));
$response = json_decode($make_call, true);
$errors   = $response['response']['errors'];
$data     = $response['response']['data'][0];
?>
```


Neila
localhost/rest/cURLEmai
localhost/rest/cURLEmailVerify.php
array (size=12)
'email' => string 'neila.benlakhal@enicarthage.rnu.tn' (length=34)
'did_you_mean' => string 'neila.benlakhal@enicarthage.edu.tn' (length=34)
'user' => string 'neila.benlakhal' (length=15)
'domain' => string 'enicarthage.rnu.tn' (length=18)
'format_valid' => boolean true
'mx_found' => boolean true
'smtp_check' => boolean true
'catch_all' => null
'role' => boolean false
'disposable' => boolean false
'free' => boolean false
'score' => float 0.48
format is valid? 1
SMTP?1
Domain? enicarthage.rnu.tn

- https://developers.google.com/maps/documentation/timezone/intro



Test via SOAPUI

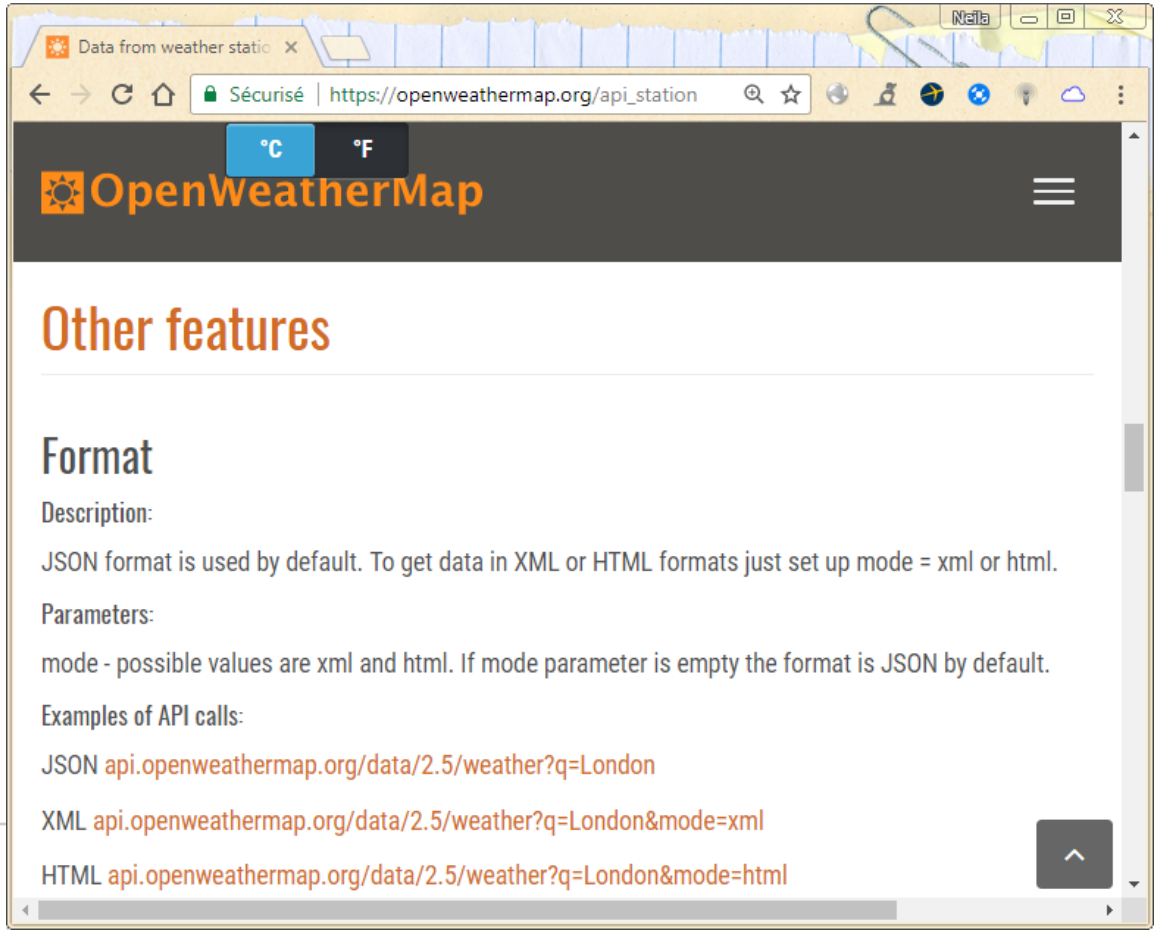


Client PhP (sans curl/ XML Parser)

```
<?php $url =
"https://maps.googleapis.com/maps/api/timezone/xml?location=38.908133
%2C-
77.047119&timestamp=1458000000&key=AIzaSyBqL3WEUQ9L_J8wXjgeyjr5vUvYtK
Q6L6U";
$result = file_get_contents($url);
//echo $result;//affichage brut du retour XML
// parcours avec l'API SimpleXML
$xml=simplexml_load_string($result) or die("Error: Cannot create
object"); //print_r($xml);// verification
echo "Status: ". $xml->status;
echo "raw_offset: ". $xml->status."<br/>";
echo "raw_offset: ".$xml->raw_offset."<br/>";
echo "dst_offset: ".$xml->dst_offset."<br/>";
 echo "time_zone_id: ".$xml->time_zone_id."<br/>";
echo "time_zone_name: ".$xml->time_zone_name."<br/>"; ?>
```

REST-style Web service client :

Rest-style resource oriented service :
https://openweathermap.org/api_station



REQUEST (SOAPUI)

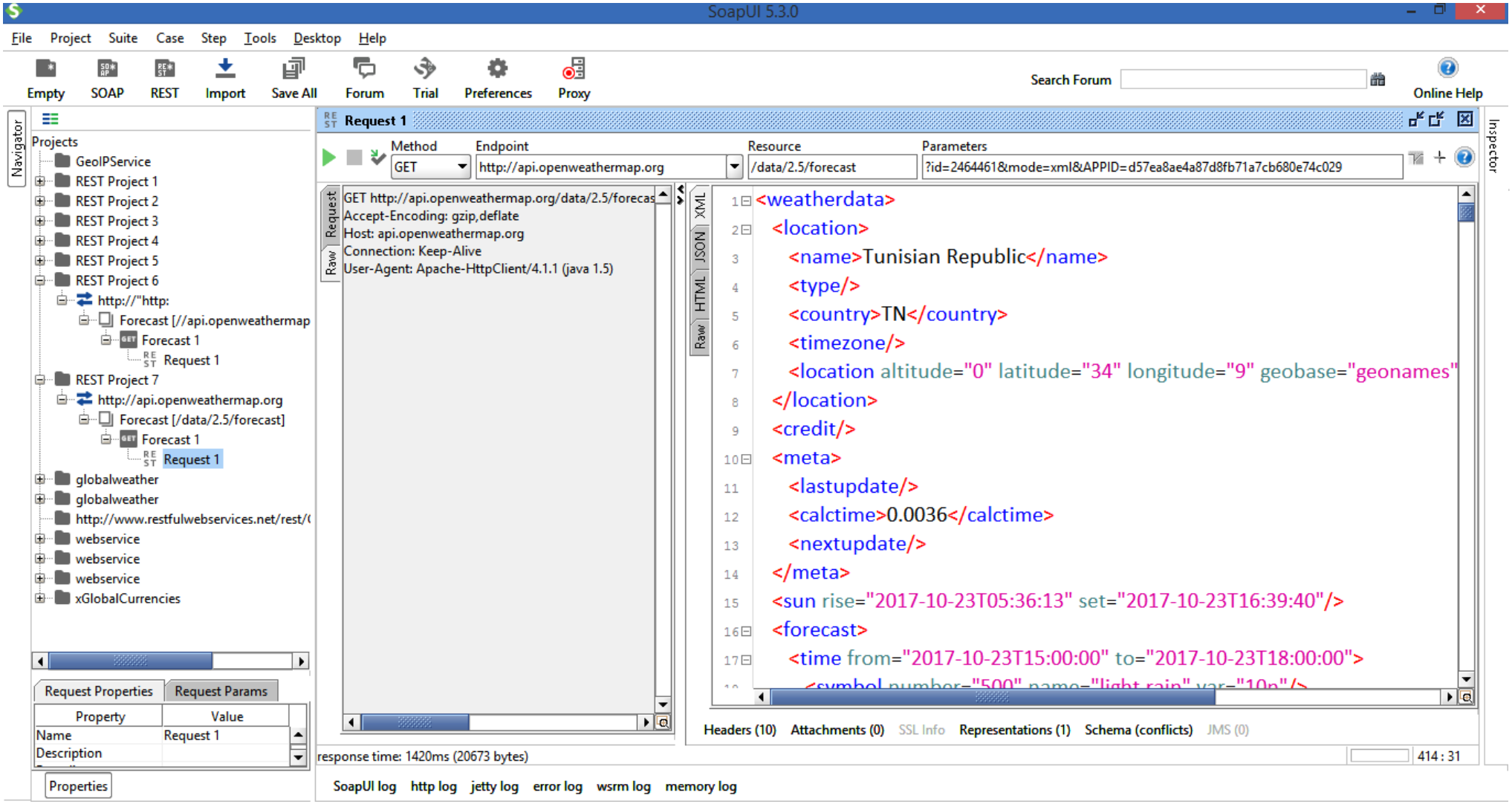
GET
http://api.openweathermap.org/data/2.5/forecast?id=2464461&**mode=xml**&APPID=d57ea8ae4a87d8fb71a7cb680e74c029 HTTP/1.1

Accept-Encoding: gzip,deflate

Host: api.openweathermap.org

Connection: Keep-Alive

User-Agent: Apache-HttpClient/4.1.1 (java 1.5)


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Navigator
Projects
GeoIPService
REST Project 1
REST Project 2
REST Project 3
REST Project 4
REST Project 5
REST Project 6
http://"http:
Forecast [//api.openweathermap
Forecast 1
Request 1
REST Project 7
http://api.openweathermap.org
Forecast [/data/2.5/forecast]
Forecast 1
Request 1
globalweather
globalweather
http://www.restfulwebservices.net/rest/(
webservice
webservice
webservice
xGlobalCurrencies
Request Properties
Request Params
Property
Value
Name
Request 1
Description
Properties
Request 1
Method
Endpoint
Resource
Parameters
GET
http://api.openweathermap.org
/data/2.5/forecast
?id=2464461&mode=xml&APPID=d57ea8ae4a87d8fb71a7cb680e74c029
Request
Raw
GET http://api.openweathermap.org/data/2.5/forecas
Accept-Encoding: gzip,deflate
Host: api.openweathermap.org
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
XML
JSON
HTML
Raw
<weatherdata>
<location>
<name>Tunisian Republic</name>
<type/>
<country>TN</country>
<timezone/>
<location altitude="0" latitude="34" longitude="9" geobase="geonames"
</location>
<credit/>
<meta>
<lastupdate/>
<calctime>0.0036</calctime>
<nextupdate/>
</meta>
<sun rise="2017-10-23T05:36:13" set="2017-10-23T16:39:40"/>
<forecast>
<time from="2017-10-23T15:00:00" to="2017-10-23T18:00:00">
Headers (10) Attachments (0) SSL Info Representations (1) Schema (conflicts) JMS (0)
response time: 1420ms (20673 bytes)
414 : 31
Inspector
SoapUI log http log jetty log error log wsrm log memory log

RESTFul service client in PhP: use file_get_contents 1/2

- For accessing services we need an HTTP client.
- We can use file_get_contents to access not only local files but also those located on remote servers over HTTP :

```
<?php
$url =
"http://api.openweathermap.org/data/2.5/forecast?id=2464
461&mode=xml&APPID=d57ea8ae4a87d8fb71a7cb680e74c029";
$result = file_get_contents($url);
echo $result; ?>
```

RESTFul service client in PhP: use file_get_contents 2/2

- However, file_get_contents uses GET, you would need considerable extra work to use a different verb like POST.
- Use HTTP client library such as CURL is better since it supports HTTP verbs such as POST and PUT in addition to GET.

References

- RESTful Web Services Paperback by Leonard Richardson , Oreilly, 2007.
- RESTful PHP Web Services by Samisa Abeysinghe, Packt, 2008.
- Internet & World Wide Web: How to Program, Fifth Edition, by Abbey Deitel, Harvey Deitel, Paul Deitel, Prentice Hall (chap 22) : how to create SOAP/REST Services in C#
- Architectural Styles and the Design of Network-based Software Architectures, by Roy Fielding PhD Thesis, University of California, Irvine, 2000
  https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Restful web services vs. "big"' web services: making the right architectural decision.by Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. In Proceedings of the 17th international conference on World Wide Web (WWW '08). ACM, New York, NY, USA, 805-814.
- http://www.restapitutorial.com/ (oct 2017)

Where to find REST WS

- https://www.programmableweb.com/
- Flickr, Yahoo, google Api, etc.
- https://github.com/toddmotto/public-apis
- Etc.

TP à faire

- Refaire les tests de Web services dans le cours en utilisant (SOAPUI/Postman)
- Apprendre à interpréter le contenu des requêtes/réponses HTTP et distinguer entre :
  - Appel vers : Web document | Rest service | SOAP service
  - Utiliser la console de Chrome (network sniffer)
- Générer le WADL des REST service dans SOAPUI
- Appeler un service REST via SOAPUI
- Écrire un client service REST : décoder réponse Json
- Écrire un client service REST : parser réponse XML

Rest Service ?

- REST service is:
- Platform-independent
- Language-independent
- Runs on top of HTTP, and
- Can easily be used in the presence of firewalls.

Rest service is not ?

- No built-in security features, encryption, these are added by building on top of HTTP:
  - For security, username/password tokens are often used.
  - For encryption, REST can be used on top of HTTPS (secure sockets).
- No State
  - The "ST" in "RE**ST**" stands for "State Transfer",
  - REST operations are self-contained, and each request carries with it (transfers) all the information (state) that the server needs in order to complete it.

REST vs SOAP

- **Why REST?**
  - REST uses standard HTTP – so it is much simpler
  - REST permits many different data formats - SOAP only permits XML
  - REST has better performance and scalability
  - Smaller messages – less overhead
  - Mobile developers know REST
- **Why SOAP?**
  - Wider support for transaction support
  - Better support for transaction support/security - WS-Security, WS-AtomicTransaction, WS-ReliableMessaging
  - More robust error handling
  - Your developer tools may not support REST yet