Course notes
Restful Web services

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

SOC

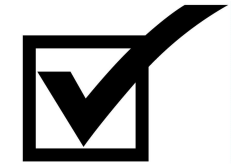
LECTURE NOTES

Service Oriented Computing

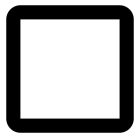
By, **Dr. Neila Ben Lakhal**
@ENICARTHAGE

Chapter 4. REST ARCHITECTURE

- *Web services types :*



SOAP Web services



REST Web services

What is REST?

Roy stated that REST "provides a set of architectural constraints that, when applied as a whole, emphasizes **scalability** of component interactions, **generality** of interfaces, **independent** deployment of components, and intermediary components to reduce interaction latency, enforce security, and **encapsulate** legacy systems."

Excerpt From: Haafiz Waheed-ud-din Ahmad. “Building RESTful Web Services with PHP 7.”

What is REST?

- **REST** is a term coined by Roy Fielding in his Ph.D [1].
- A dissertation to describe an **architecture style** of networked systems.
- An acronym standing for **RE**presentational **S**tate **T**ransfer.
- It Defines the architectural style for building large-scale distributed network-based systems.
- HTTP 1.1. was based on REST.
- Examples of REST web services you already know :
  - The modern web is an example of REST architecture.
  - Static web sites . Web applications, especially read-only ones like search engines.

[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

REST Goal

- REST Goal : establish a message exchange process between a **Resource Requester** and a **Resource provider**.

- A feeling of déjà-vu?

- Yes : We are already using a specific form of REST with WWW !!!
- With every page opened in your browser, you are making a **REST call** ..
- The **browser** acts as a **client** and **server** acts as a REST service **provider**.

REST vs Web

- A key difference between REST and Web Apps:

  - Web serves DATA to **humans**

  ⇢ must be in a human-readable format (HTML)

  - REST serves DATA to **machines**

  ⇢ can be in any machine interpretable format (JSON, XML, ...)

What kind of DATA ?

Web vs. programmable Web

- The Web serves DATA
  - **Web data examples :** book information, messages, 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 through it.

  DATA is served in HTML.
- The Programmable WEB serves SERVICES
  - **Web services examples** : search engines, online stores, currency converter, exchange rate calculator, etc.
  - Instead of arranging data in HTML pages, data is served in plain XML, JSON,...
  - It is not necessarily for human consumption.
  - It is is intended as input to a software program.

Types of service in the programmable Web

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

Web vs. programmable Web

- In the same way **Documents** are inhabitants of the Web, **Services** are inhabiting 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 common : All are based on **HTTP**.

SOAP vs REST

- Unlike SOAP, REST has no protocols or standards of its own
  - There will never be a W3C recommendation for REST.

- Like SOAP, It relies on **URIs** (Uniform Resource Identifier) and **HTTP** in order to **establish a message exchange process**
  - Instead of using complex mechanisms such as CORBA, or RPC to to make calls between machines.

RESTful web services ?

- RESTful web services are based on RESTful resources.
- A RESTful resource is an entity/resource that is mostly stored on a server and that client request using RESTful web services.
- Here are a few characteristics of a resource in terms of RESTful web services:
  - It is an entity normally referred as a noun in the URL
  - It is unique.
  - It has data associated with it.
  - It has at least one URI.
- With REST, we use **URIs** to connect clients and servers in order to exchange **resources** in the form of **representations**.

What can be a Resource ?

- Anything that can be stored and represented as a stream of bits: a document, *a* row in a DB, the result of running an algorithm.
- For example, in a shopping cart : a Product, Category, and an order can be a resource.
- **What makes a resource a resource in REST**?
  - It has to have at least one **URI.**
  - The **URI** is the name + 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.

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***.

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

How a resource can be manipulated ?

- There are 5 typical operations which can be performed on a resource:
  - List
  - Create
  - Read
  - Update
  - Delete
- For each operation, we need two things:
  - The URI and HTTP method or verb.
  - URI: HTTP method (verb) + resource name (noun)

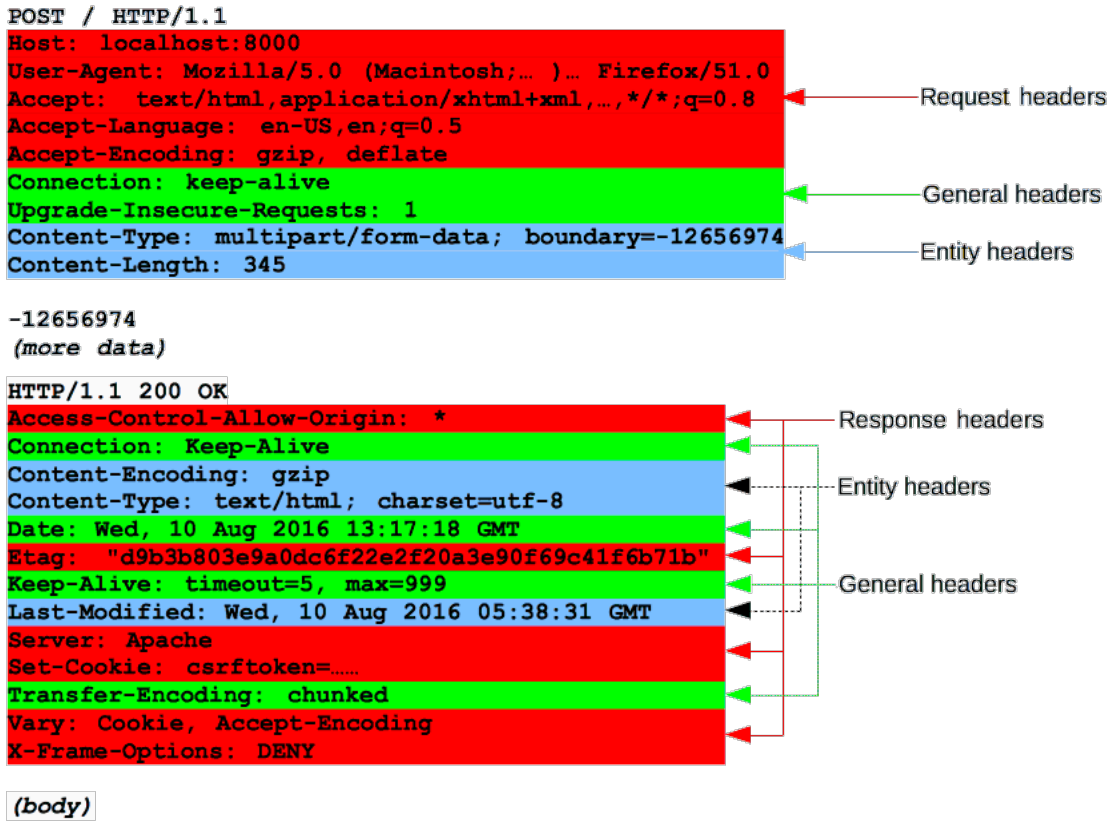
HTTP, the protocol that all web services have in common.

- Since we will be mainly dealing with URLs over HTTP and will be using HTTP methods, it is better to spend some time understanding the nature of HTTP methods.

- 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	



Source : https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

HTTP Request

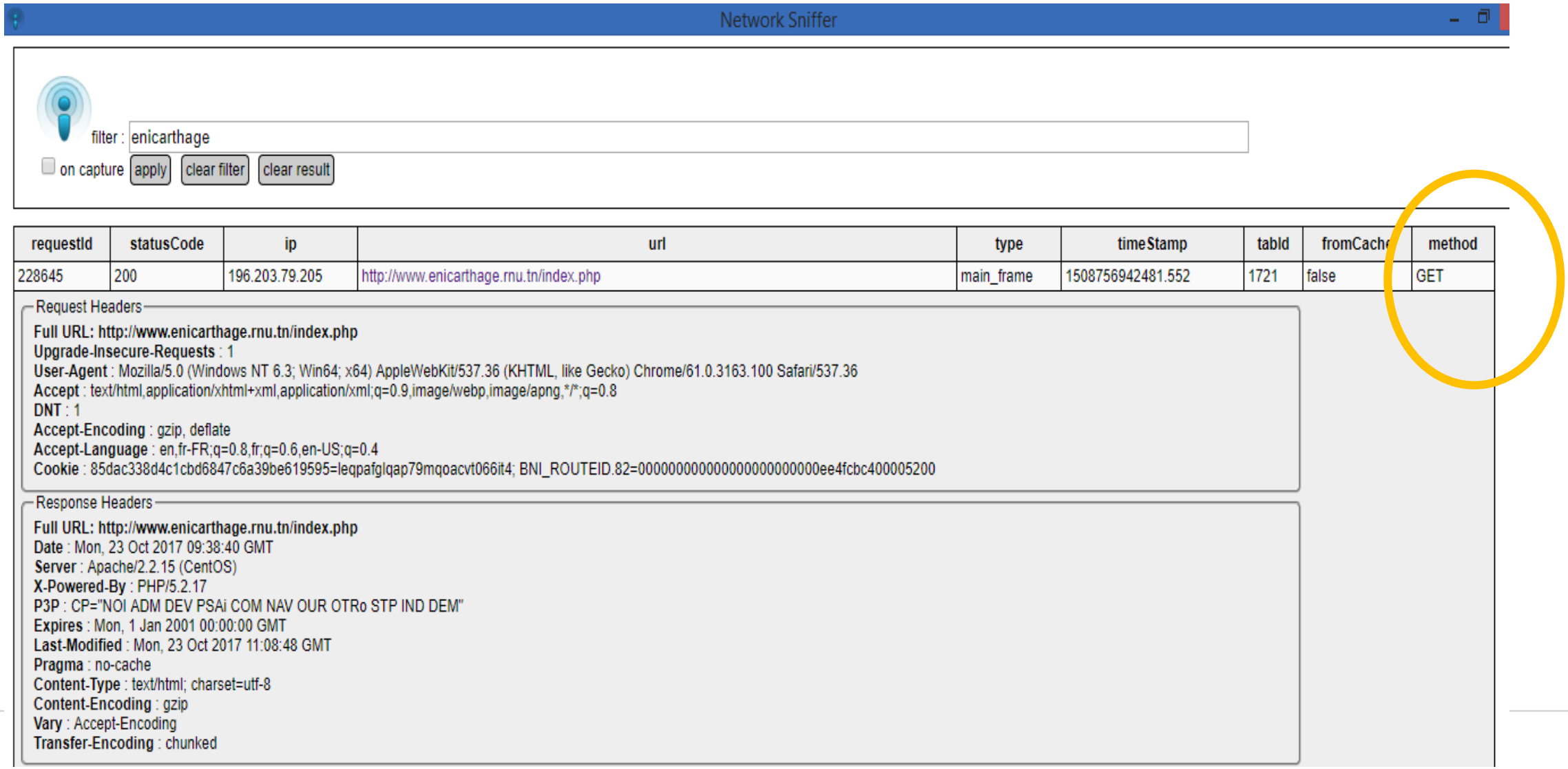
- How to tell the server what the client want to do ?
- The HTTP Request can be :
  - Call for a Web document.
  - Call for a Web service:
    - REST
    - SOAP
  - It depends on the **HTTP method (verb)**

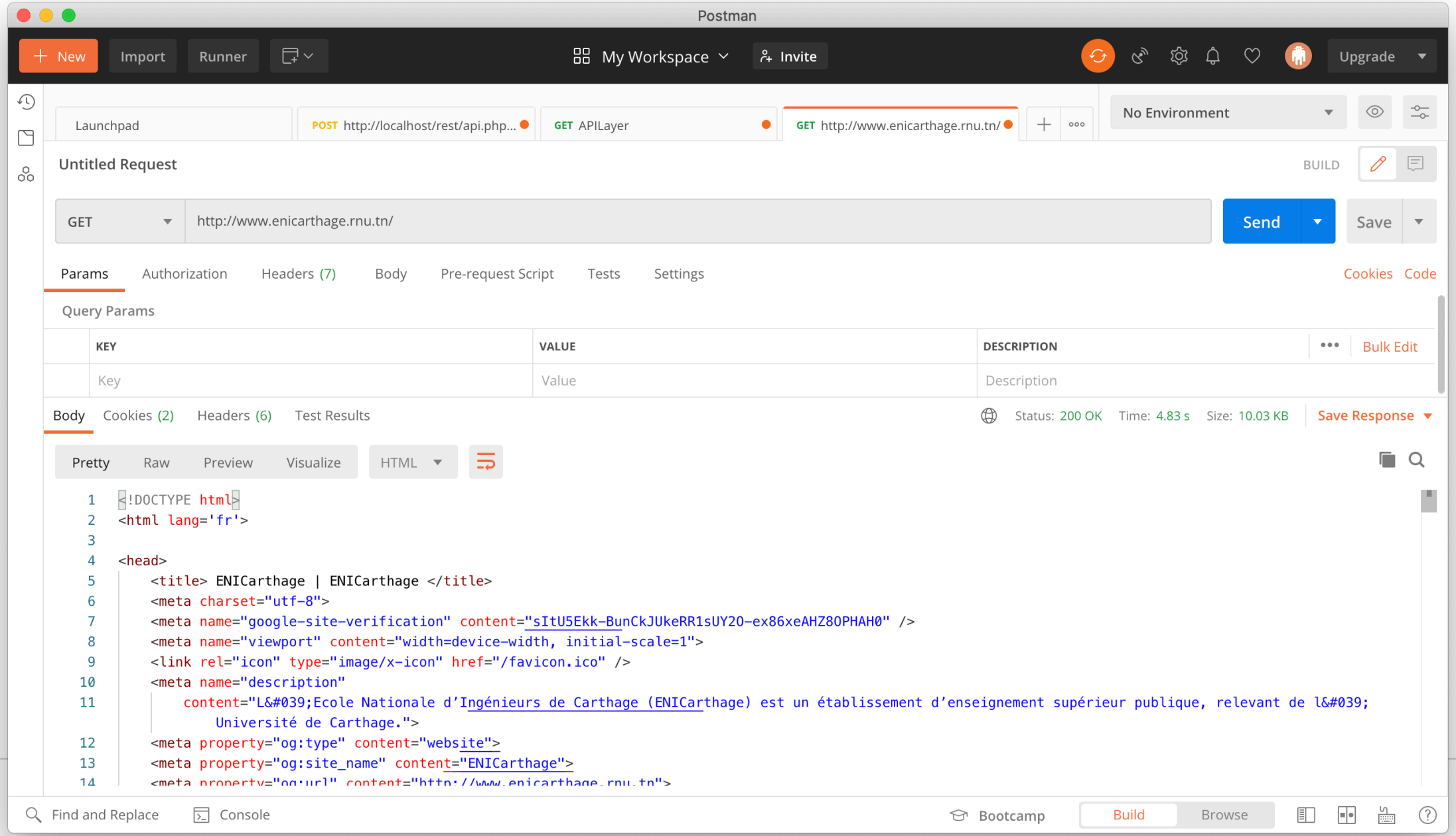
Nature of HTTP Methods

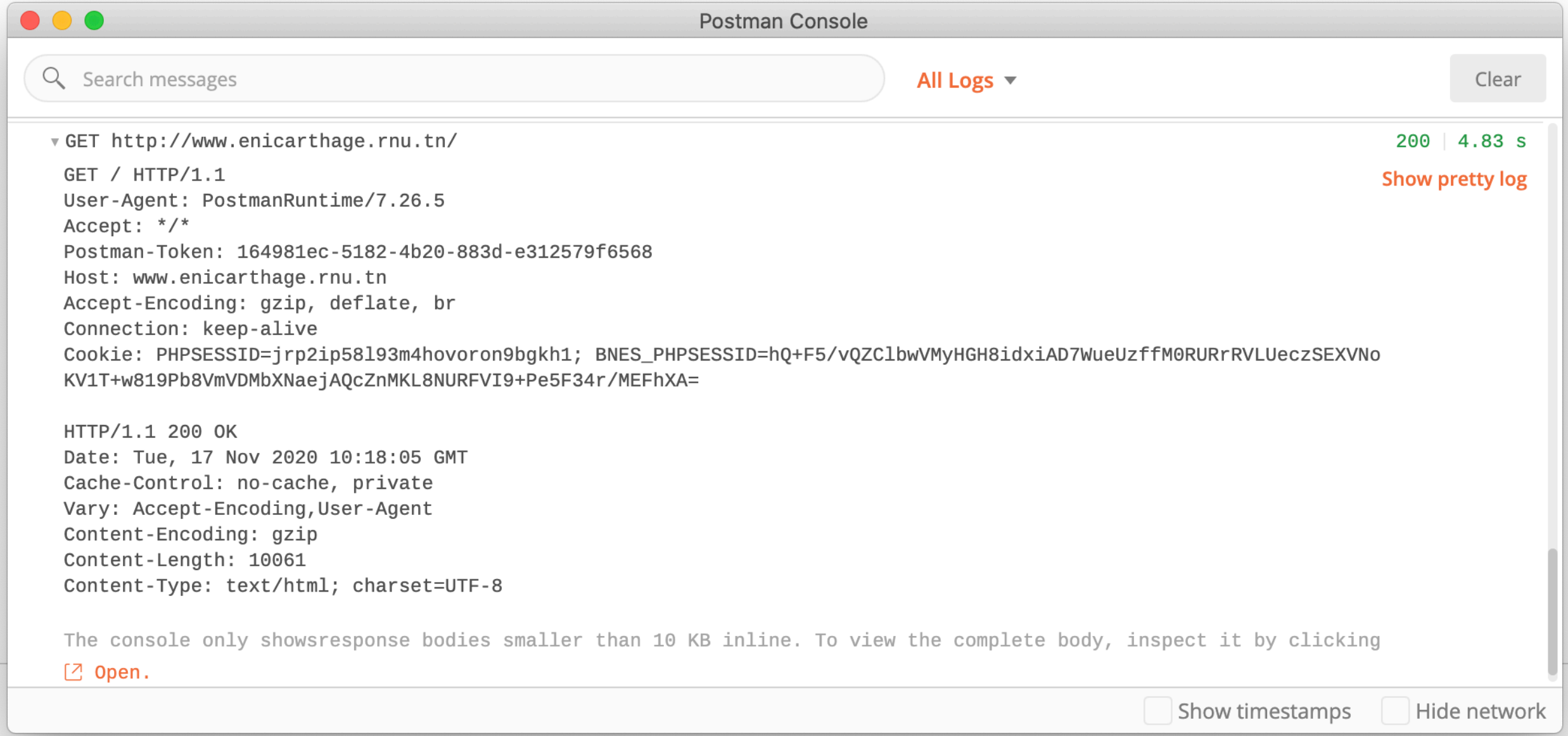
- Safe/unsafe methods
  - Safe methods : e.g., GET
    - Methods are not expected to change any resource on the server.
  - Unsafe methods : e.g., POST, PUT, DELETE, PATCH
    - Methods are expected to change some resource on the server.
- Idempotent/ non-idempotent methods
  - Idempotent methods : e.g., GET, PUT, PATCH, and DELETE
    - Methods achieve the same results no matter how many time we repeat the same operations.
  - Non-idempotent methods : e.g., POST
    - It creates a new resource on the server and has a response.

Example : An HTTP GET request and response

http://www.enicarthage.rnu.tn/index.php

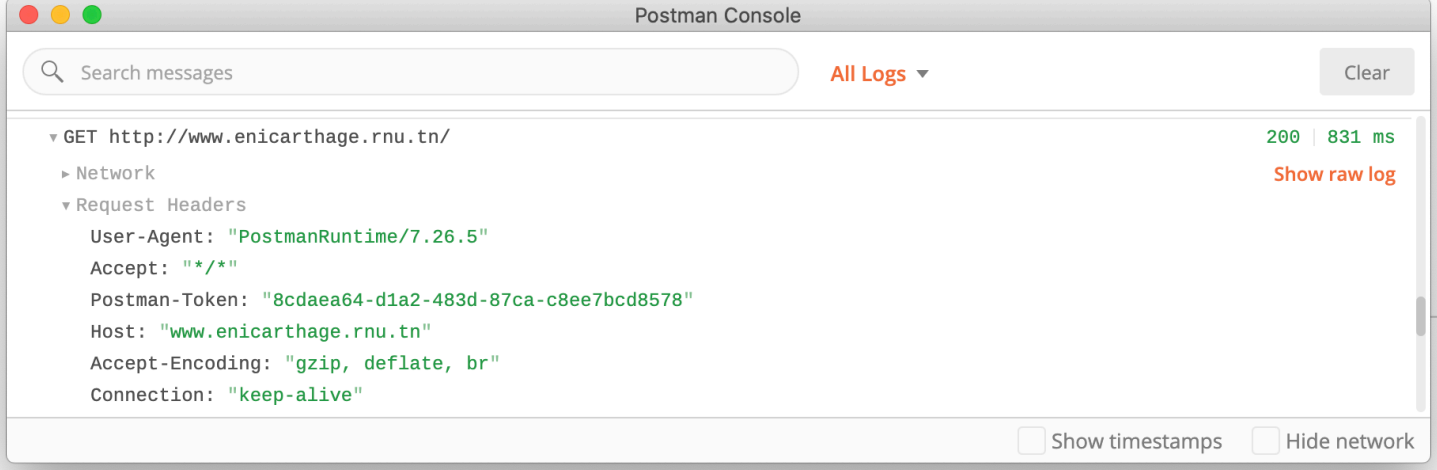



Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Launchpad
POST http://localhost/rest/api.php...
GET APILayer
GET http://www.enicarthage.rnu.tn/
No Environment
Untitled Request
BUILD
GET
http://www.enicarthage.rnu.tn/
Send
Save
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies (2)
Headers (6)
Test Results
Status: 200 OK
Time: 4.83 s
Size: 10.03 KB
Save Response
Pretty
Raw
Preview
Visualize
HTML
1 <!DOCTYPE html>
2 <html lang='fr'>
3
4 <head>
5 <title> ENICarthage | ENICarthage </title>
6 <meta charset="utf-8">
7 <meta name="google-site-verification" content="sItU5Ekk-BunCkJUkeRR1sUY2O-ex86xeAHZ8OPHAH0" />
8 <meta name="viewport" content="width=device-width, initial-scale=1">
9 <link rel="icon" type="image/x-icon" href="/favicon.ico" />
10 <meta name="description"
11 content="L'Ecole Nationale d'Ingénieurs de Carthage (ENICarthage) est un établissement d'enseignement supérieur publique, relevant de l'
Université de Carthage.">
12 <meta property="og:type" content="website">
13 <meta property="og:site_name" content="ENICarthage">
Find and Replace
Console
Bootcamp
Build
Browse


Postman Console
Search messages
All Logs
Clear
GET http://www.enicarthage.rnu.tn/
200 | 4.83 s
Show pretty log
GET / HTTP/1.1
User-Agent: PostmanRuntime/7.26.5
Accept: */*
Postman-Token: 164981ec-5182-4b20-883d-e312579f6568
Host: www.enicarthage.rnu.tn
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: PHPSESSID=jrp2ip58l93m4hovoron9bgkh1; BNES_PHPSESSID=hQ+F5/vQZClbwVMyHGH8idxiAD7WueUzffM0RURrRVLUeczSEXVNo
KV1T+w819Pb8VmVDMbXNaejAQcZnMKL8NURFVI9+Pe5F34r/MEFhXA=
HTTP/1.1 200 OK
Date: Tue, 17 Nov 2020 10:18:05 GMT
Cache-Control: no-cache, private
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 10061
Content-Type: text/html; charset=UTF-8
The console only showsresponse bodies smaller than 10 KB inline. To view the complete body, inspect it by clicking
Open.
Show timestamps
Hide network

neila.benlakhal@gmail.com

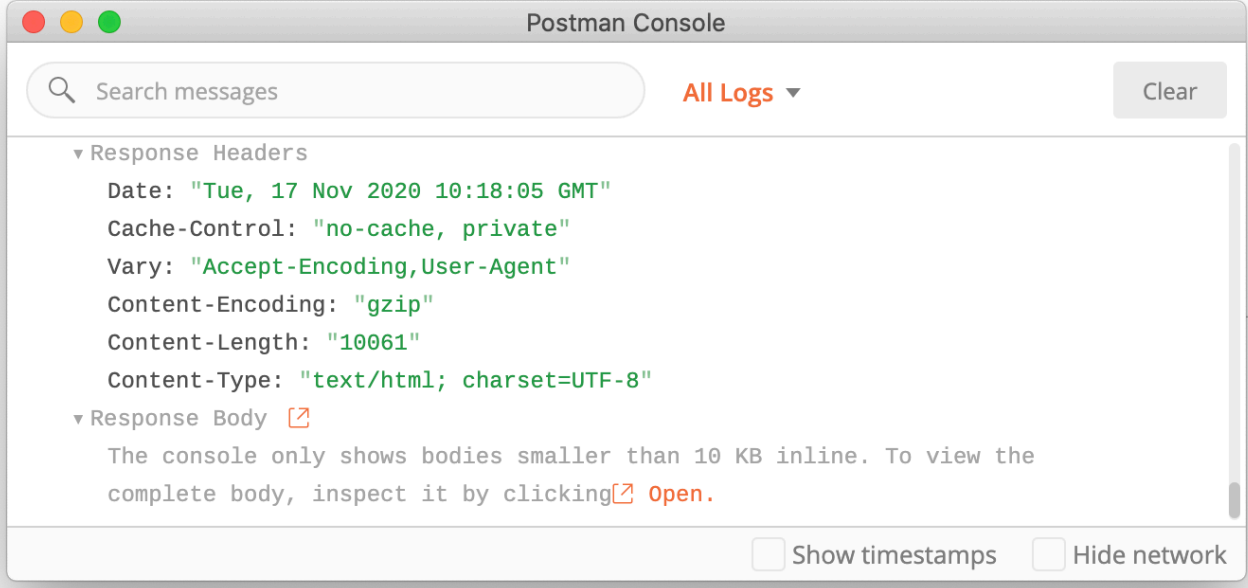
HTTP request (Web document)



- **HTTP method** : The name of the HTTP method (in this case GET) 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 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.
- **Request headers** : key-value pairs that act like informational stickers. E.g., Host, User-Agent, Accept,....
- **Entity-body** : also called the document or **representation**: This is the document inside the envelope.
  - *This 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)



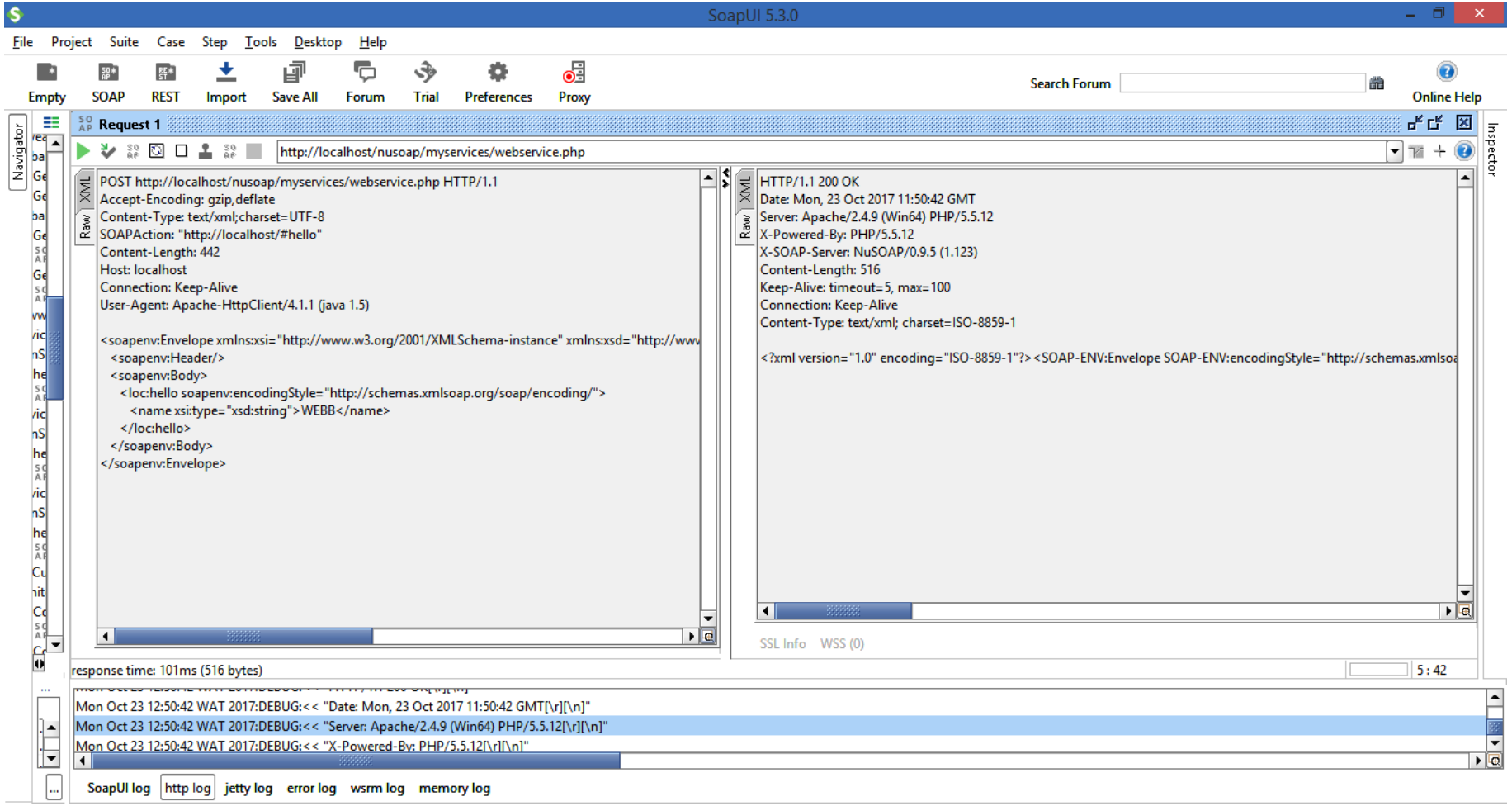
- The response can be divided into 3 parts
  - **HTTP response code** : This numeric code tells the client the request status; (200: "OK").
  - **Response headers** : These are informational stickers slapped onto the envelope.
  - **Entity-body or representation** :
    - This is the document inside 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.**

Method information : SOAP-style

- Typical SOAP service keeps its method information in the **entity-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 : SOAP envelope body.

➔ **SOAP services : RPC-style architecture.**

HTTP Request/response (Service)



Method information : REST-style

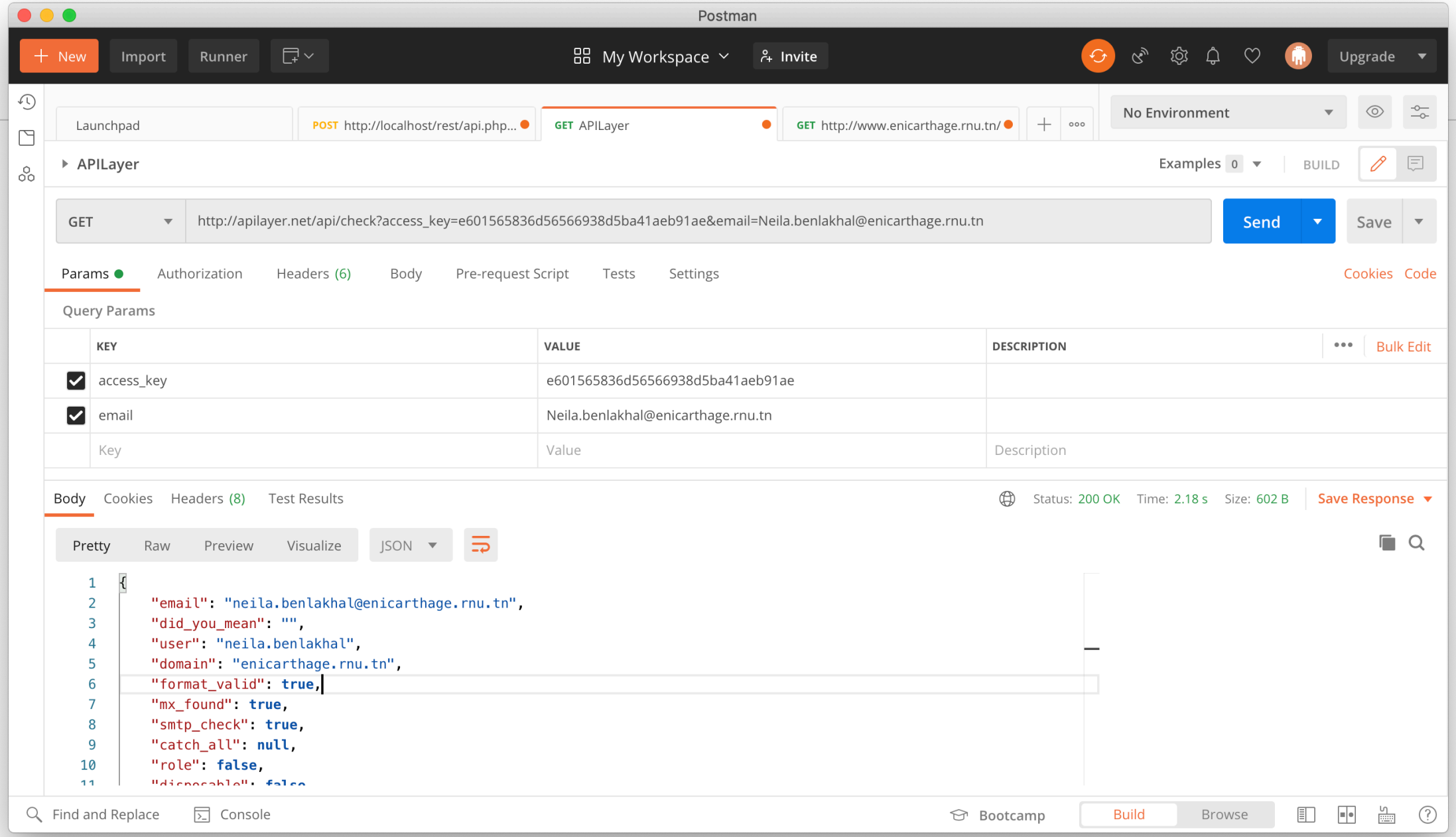
- REQUEST:

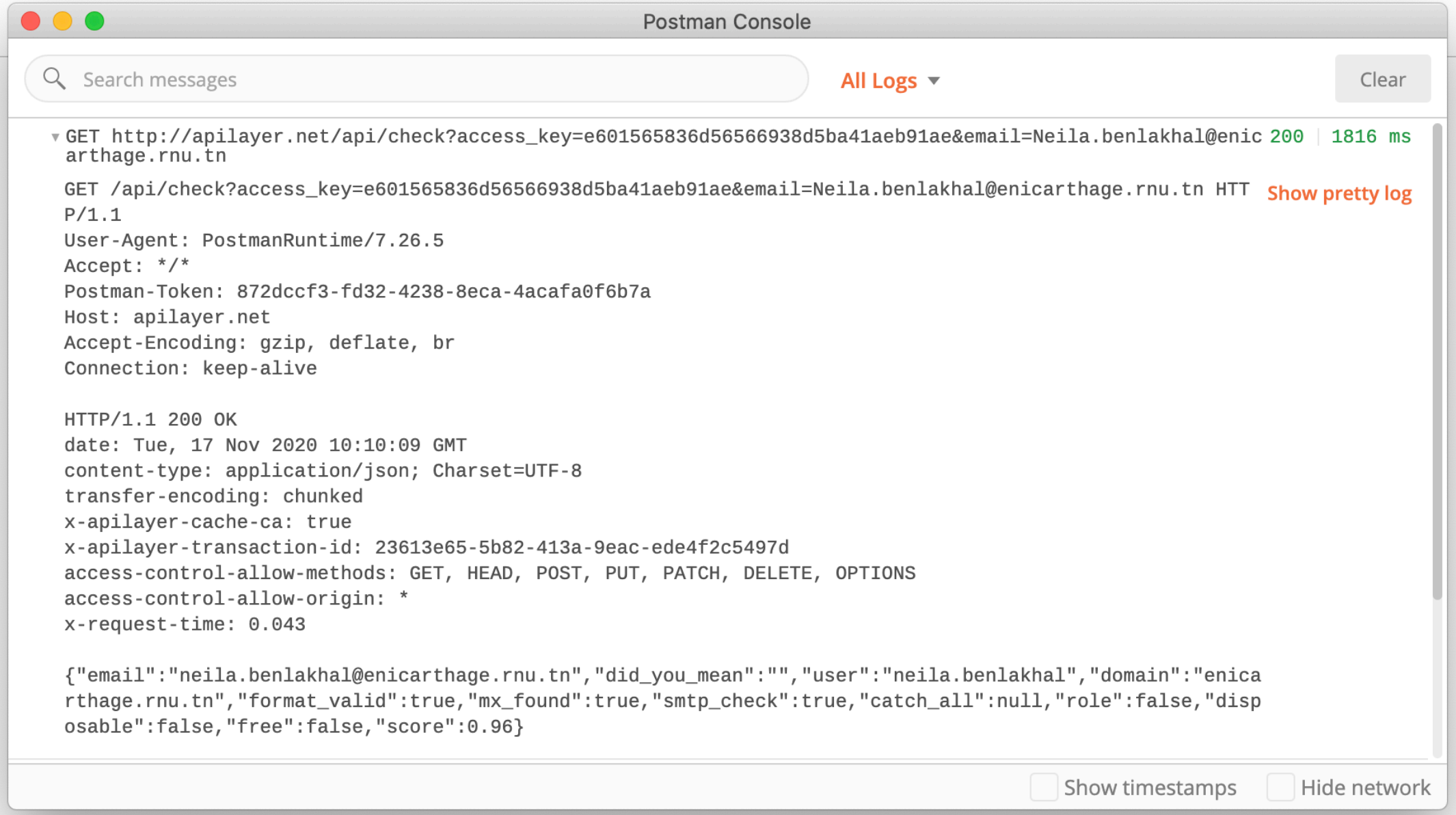
```
GET  http://apilayer.net/api/check?access_key=...&email=Neila.benlakhal@enicarthage.rnu.tn
```

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

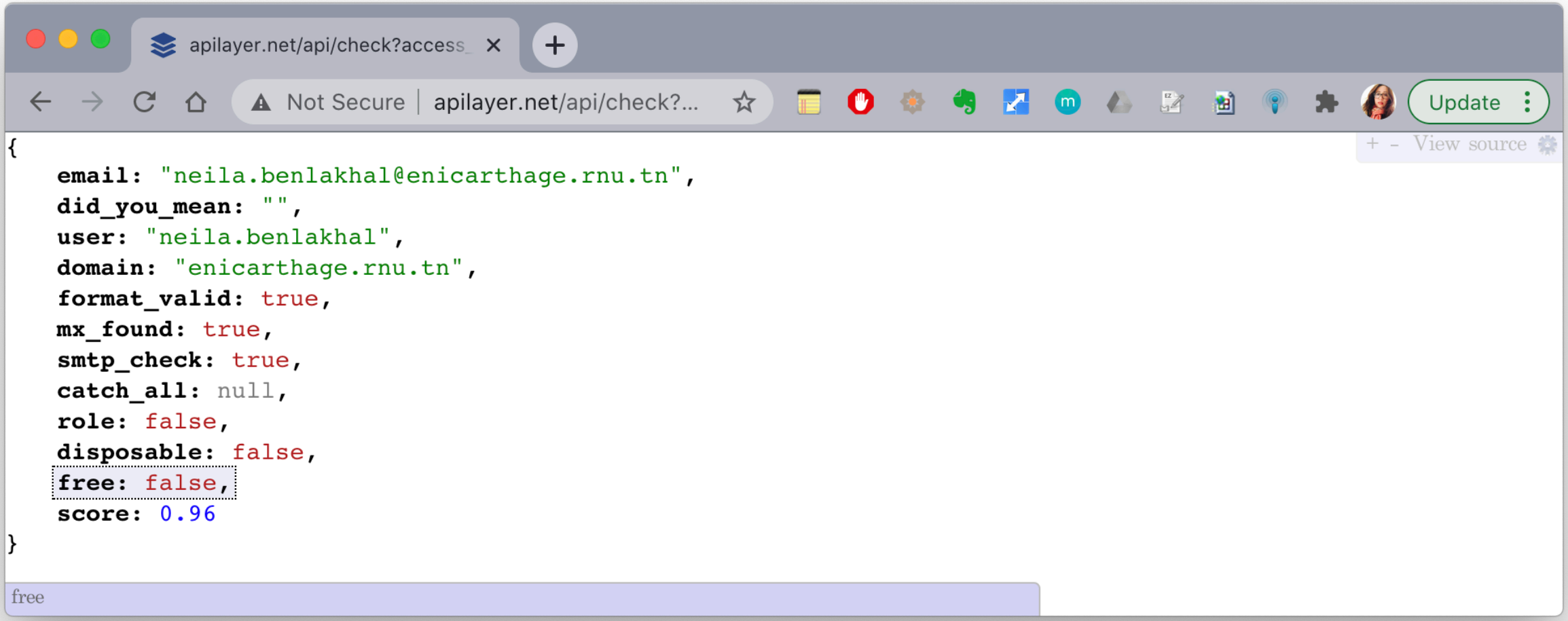
http://apilayer.net/api/check?access_key=e601565836d56566938d5ba41aeb91ae&email=Neila.benlakhal@enicarthage.rnu.tn

→ **RESTful service.**


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Launchpad
POST http://localhost/rest/api.php...
GET APILayer
GET http://www.enicarthage.rnu.tn/
No Environment
APILayer
Examples 0
BUILD
GET
http://apilayer.net/api/check?access_key=e601565836d56566938d5ba41aeb91ae&email=Neila.benlakhal@enicarthage.rnu.tn
Send
Save
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
access_key
e601565836d56566938d5ba41aeb91ae
email
Neila.benlakhal@enicarthage.rnu.tn
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 2.18 s
Size: 602 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"email": "neila.benlakhal@enicarthage.rnu.tn",
"did_you_mean": "",
"user": "neila.benlakhal",
"domain": "enicarthage.rnu.tn",
"format_valid": true,
"mx_found": true,
"smtp_check": true,
"catch_all": null,
"role": false,
Find and Replace
Console
Bootcamp
Build
Browse


Postman Console
Search messages
All Logs
Clear
GET http://apilayer.net/api/check?access_key=e601565836d56566938d5ba41aeb91ae&email=Neila.benlakhal@enicarthage.rnu.tn 200 | 1816 ms
GET /api/check?access_key=e601565836d56566938d5ba41aeb91ae&email=Neila.benlakhal@enicarthage.rnu.tn HTTP/1.1
Show pretty log
User-Agent: PostmanRuntime/7.26.5
Accept: */*
Postman-Token: 872dccf3-fd32-4238-8eca-4acafa0f6b7a
Host: apilayer.net
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
HTTP/1.1 200 OK
date: Tue, 17 Nov 2020 10:10:09 GMT
content-type: application/json; Charset=UTF-8
transfer-encoding: chunked
x-apilayer-cache-ca: true
x-apilayer-transaction-id: 23613e65-5b82-413a-9eac-ede4f2c5497d
access-control-allow-methods: GET, HEAD, POST, PUT, PATCH, DELETE, OPTIONS
access-control-allow-origin: *
x-request-time: 0.043
{"email":"neila.benlakhal@enicarthage.rnu.tn","did_you_mean":"","user":"neila.benlakhal","domain":"enicarthage.rnu.tn","format_valid":true,"mx_found":true,"smtp_check":true,"catch_all":null,"role":false,"disposable":false,"free":false,"score":0.96}
Show timestamps
Hide network

REST service response



HTTP verbs and URL structure

- Here is how (CRUD) operations can be performed on a resource using a combination of URIs and HTTP verbs:

Data action	HTTP equivalent
CREATE	POST or PUT
READ	GET
UPDATE	PUT or PATCH
DELETE	DELETE

Task	Method	Path
Create a new task	POST	/tasks
Delete an existing task	DELETE	/tasks/{id}
Get a specific task	GET	/tasks/{id}
Search for tasks	GET	/tasks
Update an existing task	PUT	/tasks/{id}

Others HTTP verbs (Less used)

- OPTIONS: This method is used by the client to determine the options or actions associated with the target resource, without causing any action on the resource or retrieval of the resource.

- HEAD: This method can be used for retrieving information about the entity without having the entity itself in the response.

CREATE operation

HTTP method : POST

URI: /{resource}

Parameters: can be multiple parameters in POST body

Result : This should create a new resource with parameters in the body. If a request parameter is missed by mistake, the response code is 400, represents a bad request.

Creating blog post

- **Request:** `POST /posts HTTP/1.1`
- **Body parameters:**

  Content: This is an awesome post
  Title: Awesome Post

- **Response:**

```
{id:1, title:"Awesome Post", content:"This is an awesome
post", link: "/posts/1" }
```

- **Response code:** 201 Created

neila.benlakhal@gmail.com

Search/List operation

HTTP method : GET

URI: /{resource}

Result : It returns the list of the type of resource whose name is mentioned.

Listing all blog posts

- **Request:** `GET /posts HTTP/1.1`
- **Response:**

```
{
data:[
  {
   id:1, title:"Awesome Post", content:"This is an
awesome post", link: "/posts/1"
  },
  {
   id:2, title:"Amazing one", content:"This is an
amazing post", link: "/posts/2"
  }
 ],
total_count: 2,
limit:10,
pagination: {
    first_page: "/posts?page=1",
    last_page: "/posts?page=1",
    page=1
 }
}
```

- **Response code:** `200 OK`

Here, data is an array of objects as there are multiple records returning. Other than `total_count`, there is a pagination object as well, and right now it shows the first and last pages because `total_count` for records is only 2. So, there is no next or previous page. Otherwise, we should also have to show the next and previous in pagination.

READ operation

HTTP method : GET

URI:
/{resource}/{resource_id}

Result : This should return the record based on the resource's ID.

Examples :

Reading blog post

- **Request:** `GET /posts/1 HTTP/1.1`
- **Response:**

```
{id:1, title:"Awesome Post", content:"This is an
awesome post", link: "/posts/1" }
```

- **Response code**: 200 OK

Note, if a blog post with an ID provided (in the current case, 1) does not exist, it should return **404**, which means resource Not Found.

UPDATE operation

- Can be done with PATCH(not always supported) or PUT verbs :

- HTTP method : PATCH
- URI: /{resource}/{resource_id}
- Result : To update some attributes of the resource.
- Parameters: There can be multiple parameters passed using a query string or in the body.

- HTTP method : PUT
- URI: /{resource}/{resource_id}
- Result : To replace the whole resource.
- Parameters: There can be multiple parameters passed using a query string or in the body.

Update operation

HTTP method : PATCH /PUT

URI: /{resource}/{resource_id}

Result : To update some attributes /Whole resource.

Parameters: There can be multiple parameters passed using a query string or in the body.

Updating blog post

- **Request:** `PATCH /posts/1?title=Modified%20Post HTTP/1.1`
- **Response:**

```
{id:1, title:"Modified Post", content:"This is an
awesome post", link:"posts/1" }
```

- **Response code**: 200 OK

Note, if a blog post with the ID provided (that is 1 in this case) does not exist, it should return the response code **404** that means resource not found.

Delete operation

HTTP method : DELETE

URI:
/{resource}/{resource_id}

Result : Delete the resource based on the resource ID in the URI.

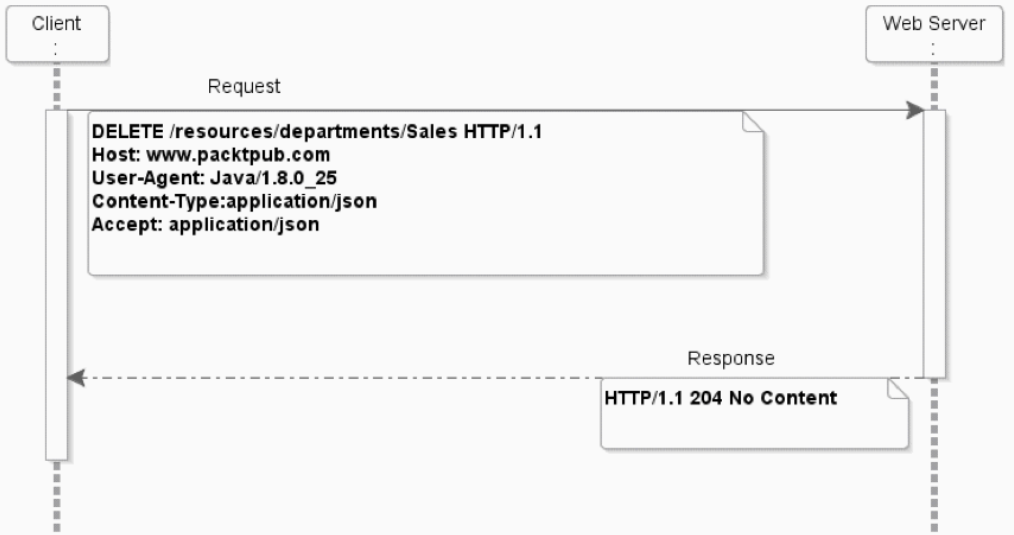
Delete blog post

- **Request:** `DELETE /posts/1 HTTP/1.1`
- **Response:**

```
{success:"True", deleted_id:1 }
```

- **Response code:** `200 OK`

*Note, if a blog post with an ID provided (in the current case, 1) does not exist, it should return **404**, which means resource `Not Found`.*



Methods summary (with HTTP status)

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

◉Source : https://www.restapitutorial.com/lessons/httpmethods.html

Example of resources | URI

- The latest version of a software release
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.

What makes a web service RESTFUL ?

◉A RESTful system or RESTful web service must abide by the following 6 principals :

1. *Uniform interface :* all clients use the same interface
2. *Separation of concern :* client and server do not interfere with each other task
3. *Stateless*: all client server communications are stateless
4. *Cacheable* : caching happens on client side
5. *Layered system* : layers can exist between client and server
6. *Code on demand* : ability to download and execute code on client side

1. Uniform interface 1/3

◉A RESTful Web service exposes a set of ***Resources*** which identify the targets of the interaction with its clients.

◉What is a Resources ?

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

◉Resources are identified by ***URI***s, which provide a global addressing space for resource and service discovery.

1. Uniform interface 2/3

- REST makes the interface separate from the implementation (like in OOP, interface is separated from implementation).

- Uniform interface means that all client will use the same interface (URI) for the same resource.

- How REST concretizes Uniform Interface ? By :
  - Using Unique Resource identification (URI)
  - Using HTTP verbs instead of service defined methods
    - Resources are manipulated using a fixed set of operations : the **HTTP verbs**.

1. Uniform interface 3/3

How unique resource identification is done ?

- Every resource is identified by a URI. The URI is the same for all clients.
- The same resource is identified by the same URI from all the clients.
- No matter how it is stored on the server, it will remain separate from what is returned in the client response.
- A resource is stored on the server side on different format. Client need not to be aware of this format.
  - For example: a resource can be a line in a database table, but the client will request it with a URI, and will receive in always in XML, JSON...
- The client doesn't have to know the storage format, it only needs to know what is requesting and what is getting in response.

2. Separation of concern

- It means that the server and client have separate responsibilities, so one is not responsible for the other's duties.
- Example:
  - The client is not responsible for data storage on the server as it is the server's responsibility.
  - The server doesn't need to know about the user interface.
- The server can be more **scalable** and the user interface on the client can be independent and more interactive.

3. Layered system

- A REST system can have multiple layers
- If a client is requesting for a response and getting a response, it can't be differentiated if it was returned from the server or another **middleware** server (a layer).
- If one server layer is replaced by an other, it doesn't affect the client unless it provides what it is expected to provide.
- One layer doesn't have knowledge beyond the next layer with which it is interacting directly.

4. Code on demand

- Is an optional constraint.
- The server can add more functionality to the REST client, by sending code that can be executable by that client.
- Example:
  - if we want to add some logic which will work in the browser then server-side developers will have to send some JavaScript code to the client side.
  - JavaScript code is code on demand which the server sends to the client that extends the functionality of the REST client.

5. 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.
- If the session needs to be maintained:
- A session identifier will be created (server side)
- With every request, client will send a session identifier
- Server will use the received session identifier to know that two requests are coming from the same client.
- Stateless advantage : simplicity

Every request with same parameters will provide the same response.

6. Caching 1/2

- RESTful web service response must define itself as cache-able or not, so that the client can know whether it should be cached or not

- Advantage : less overhead and better performance because the client will not go to the server if it is able to use the cached version.
- Caching is possible for idempotent requests : requests that do no perform actions on the server side (e.g., storing data,..).
- Example of cacheable requests: GET requests

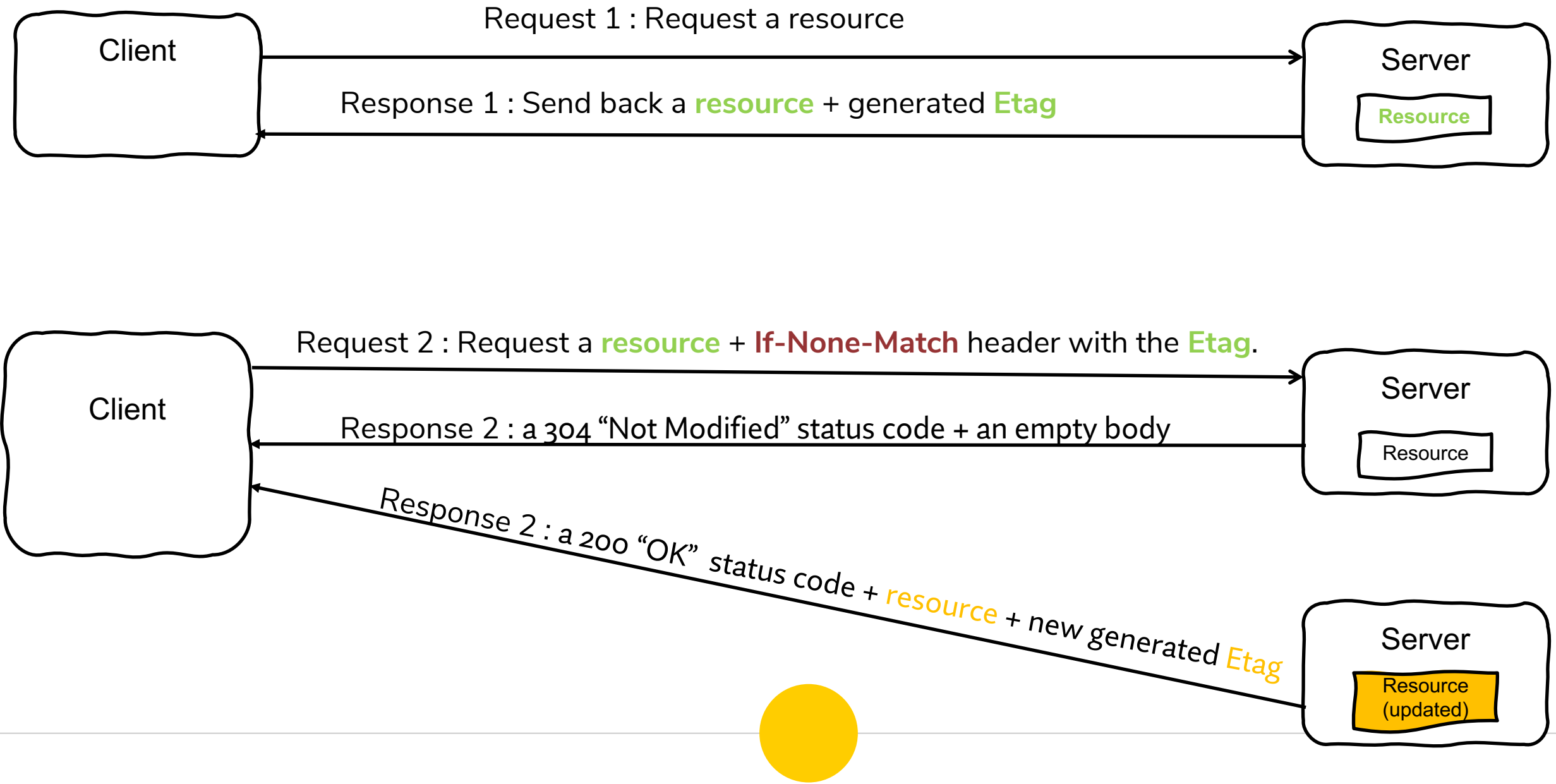
6. Caching 2/2

- Caching can either be done by :
  - The server, which makes a decision about whether to serve a previous version of a resource.
  - The client storing the result of previous requests.
- How caching is done ? Caching is done using HTTP **Caching Headers:**
  - When returning a resource, either an **ETag** Header or a **Last-Modified** Header (the date this record last changed) is included with the response.
  - Etag(Entity Tag) Header examples : a hash of the resource or a combination of file size and timestamp.

ETag header ?

- An ETag (entity tag) is an HTTP response header returned by an HTTP/1.1 compliant web server used to determine change in content at a given URL.

- The **ETag value** can be thought of as a hash computed out of the bytes of the Response body. Because the service likely uses a cryptographic hash function, even the smallest modification of the body will drastically change the output and thus the value of the ETag.


Client
Request 1 : Request a resource
Response 1 : Send back a resource + generated Etag
Server
Resource
Client
Request 2 : Request a resource + If-None-Match header with the Etag.
Response 2 : a 304 “Not Modified” status code + an empty body
Response 2 : a 200 “OK” status code + resource + new generated Etag
Server
Resource
Server
Resource (updated)

Request/response samples :

//First, the Client makes a REST API call - The Response includes the ETag header :

Request 1

curl -H "Accept: application/json" -i http://localhost/rest/api.php?order_id=15478959

Response 1

HTTP/1.1 **200** OK
ETag: "f88dd058fe004909615a64f01be66a7"
Content-Type: application/json;charset=UTF-8
Content-Length: 52

//The Resource hasn't changed on the Server,
//The Response will contain no body and a status code of 304 – Not Modified:

Request 2

curl -H "Accept: application/json" -H
'If-None-Match: "f88dd058fe004909615a64f01be66a7"'
-i http://localhost/rest/api.php?order_id=15478959

Response 2

HTTP/1.1 **304** Not Modified
ETag: "f88dd058fe004909615a64f01be66a7"

//The Resource has changed on the Server,
//The Response will contain body and a status code of 200
// Data: new data and a new ETag which, again, can be stored for further use:

Request 3

curl -H "Accept: application/json" -H
'If-None-Match: "f88dd058fe004909615a64f01be66a7"'
-i http://localhost/rest/api.php?order_id=15478959

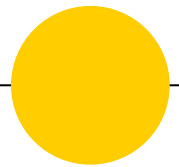
Response 3

HTTP/1.1 **200** OK
ETag: "03cb37ca667706c68c0aad4cb04c3a211"
Content-Type: application/json;charset=UTF-8
Content-Length: 56

ETag Header : Steps explanation

- **Request 1.** Client sends a request to retrieve a resource.
- **Response 1.** Server sends back retrieved resource + generated Etag.
- **Request 2.** Client sends a request to retrieve the same resource + If-None-Match header with the value of the Etag.
- **Response 2.**
  - *If resource was modified :*
  - The resource on server-side will have a different Etag from the received Etag (request 2), a new resource will be returned with its ETag header and 200 OK.
  - *If resource was not modified:*
  - if the ETag values do match, the server can simply respond with a 304 “Not Modified” status code and an empty body, indicating to the client that it can use the version it has.

Description and Discovery of RESTful Web Services



Describing RESTFUL Web services

- Unlike SOAP web services, which used WSDL, REST service providers did not really pay enough attention to documenting their RESTful web APIs.
- Soon they realized the need for a good API documentation solution in order to accelerate the adoption of their REST API services

- Many REST API metadata standards have emerged :
  - **Web Application Description Language (WADL)**
  - RESTful API Modeling Language
  - Swagger

WADL (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.

- WADL is not a standard.

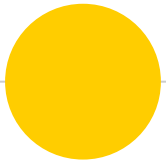
- The WADL schema is based on the WADL specification (https://www.w3.org/Submission/wadl/).

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

WADL structure

- The ***<application>*** element is the root for the WADL metadata.
- It contains :
  - o..N ***<doc>*** elements (documentations) (optional)
  - o..1 ***<grammars>*** elements : It acts as the container for schemas that describe the service's resource. The ***<grammars>*** element typically refers to XML schema.
  - o..N of the following :
    - ***<resources>*** element that contains o..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>
```

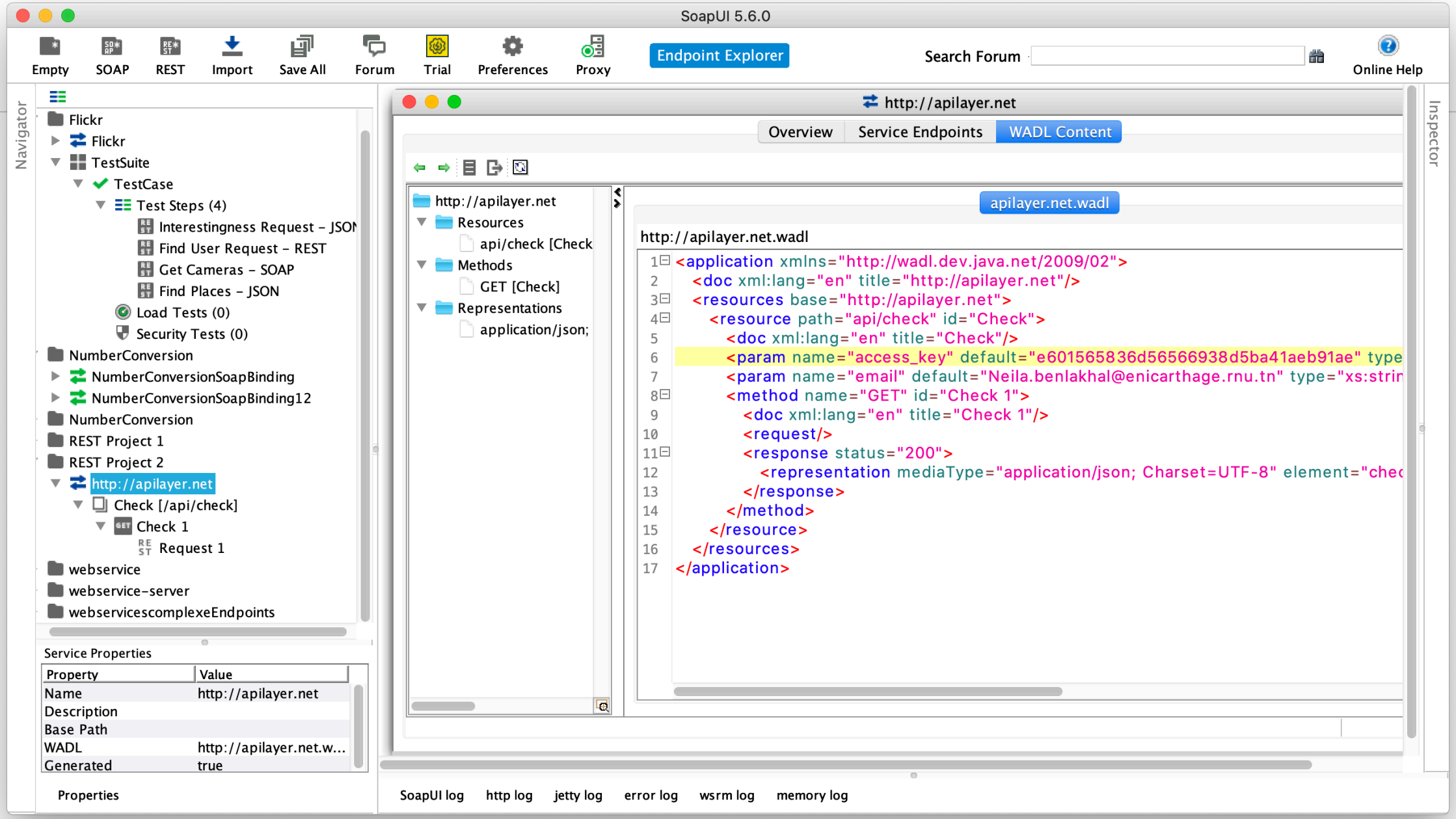


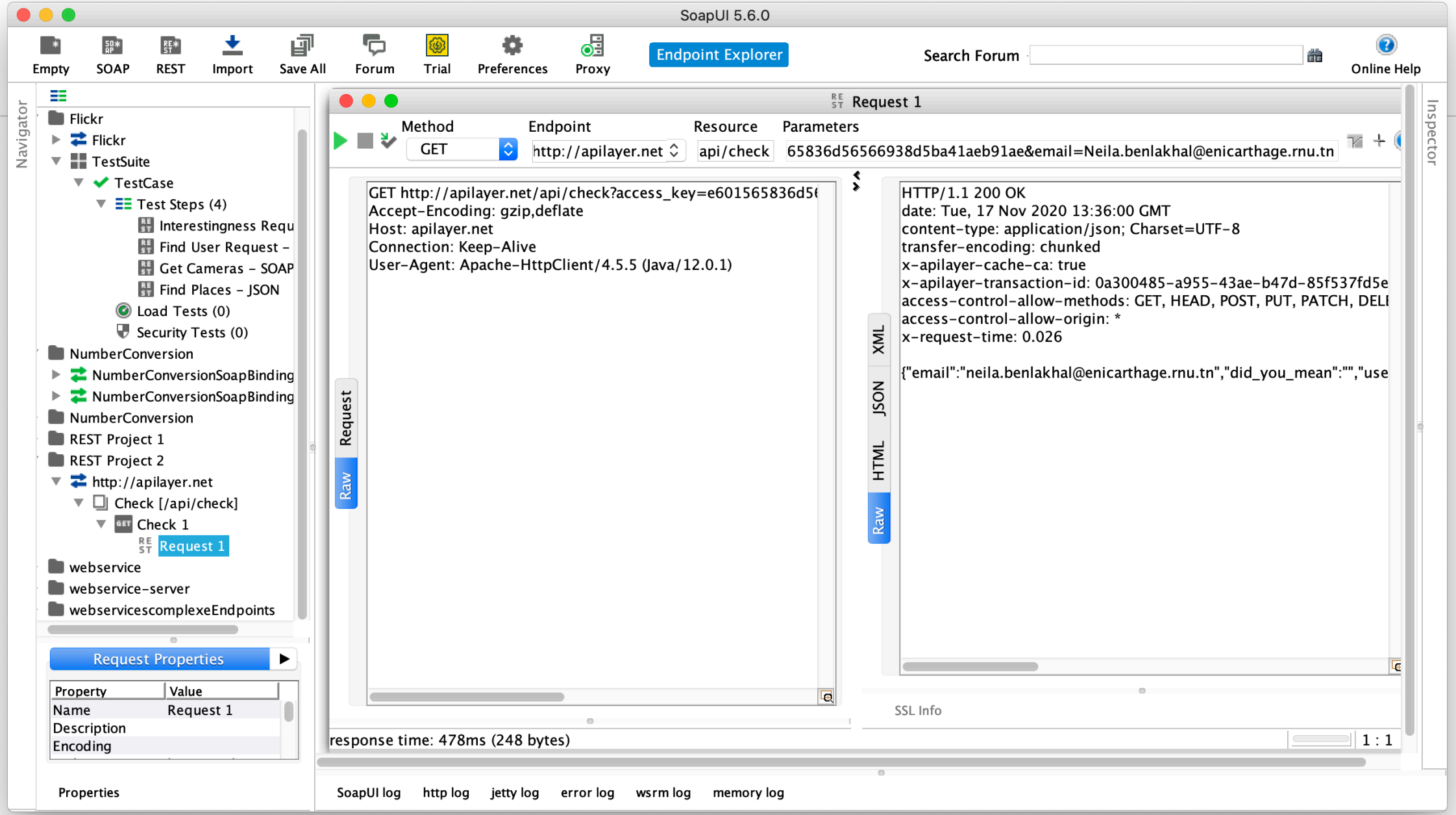
```
GET http://www.example.org/services/getStockQuote/symbol?
```

How to generate WADL

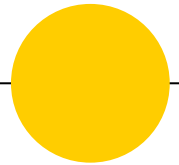
- Many environments and tools generate a WADL for a restful web services like :
  - SOAPUI
  - JAX-RS: The Jersey framework has a built-in support for generating the WADL file for your JAX-RS applications.
  - Etc.
- It is also possible to generate a code from a WADL file, for example :
  - Generating a Java client from WADL using the wadl2java tool. (https://github.com/javaee/wadl)

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




SoapUI 5.6.0
Empty
SOAP
REST
Import
Save All
Forum
Trial
Preferences
Proxy
Endpoint Explorer
Search Forum
Online Help
Navigator
Flickr
Flickr
TestSuite
TestCase
Test Steps (4)
Interestingness Requ
Find User Request -
Get Cameras - SOAP
Find Places - JSON
Load Tests (0)
Security Tests (0)
NumberConversion
NumberConversionSoapBinding
NumberConversionSoapBinding
NumberConversion
REST Project 1
REST Project 2
http://apilayer.net
Check [/api/check]
Check 1
Request 1
webservice
webservice-server
webservicescomplexeEndpoints
Request Properties
Property
Value
Name
Request 1
Description
Encoding
Properties
Request 1
Method
Endpoint
Resource
Parameters
GET
http://apilayer.net
api/check
65836d56566938d5ba41aeb91ae&email=Neila.benlakhal@enicarthage.rnu.tn
Request
Raw
GET http://apilayer.net/api/check?access_key=e601565836d5
Accept-Encoding: gzip,deflate
Host: apilayer.net
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.5 (Java/12.0.1)
XML
JSON
HTML
Raw
HTTP/1.1 200 OK
date: Tue, 17 Nov 2020 13:36:00 GMT
content-type: application/json; Charset=UTF-8
transfer-encoding: chunked
x-apilayer-cache-ca: true
x-apilayer-transaction-id: 0a300485-a955-43ae-b47d-85f537fd5e
access-control-allow-methods: GET, HEAD, POST, PUT, PATCH, DELE
access-control-allow-origin: *
x-request-time: 0.026
{"email":"neila.benlakhal@enicarthage.rnu.tn","did_you_mean":"","use
SSL Info
response time: 478ms (248 bytes)
1 : 1
Inspector
SoapUI log
http log
jetty log
error log
wsrm log
memory log

Implementing RESTFUL web services client



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***
  - ***JSON Decoder***

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 1+ libraries for making HTTP requests.
- To build a RESTful 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: **Curl (**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,…), jersey framework (JAX-RS),…
- Python: httplib2
- Etc.

Tools for client implementation

- We will use :
- cURL for HTTP request handling
- json_encode() :
  - Php Array/Objects → JSON
- json_decode() :
  - JSON→ Php Array/Objects
- The PHP language provides the json_encode() and json_decode() functions. Using these functions, we can easily encode PHP arrays and objects as well as decode various JSON structures.

Tools for client implementation

The following example demonstrates the simplicity of using the json_encode() function:

jsoon-encodeexample.php

```php
<?php
class User {
public $name;
public $age;
public $salary;
}
$user = new User();
$user->name = 'John';
$user->age = 34;
$user->salary = 4200.50;
echo json_encode($user); // {"name":"John","age":34,"salary":4200.5}
$employees = ['John', 'Mariya', 'Sarah', 'Marc'];
echo json_encode($employees); // ["John","Mariya","Sarah","Marc"]
?>
```

Tools for client implementation

- This example demonstrates of using the json_decode() and json_encode() functions usage :

jsoon-encodeexample.php

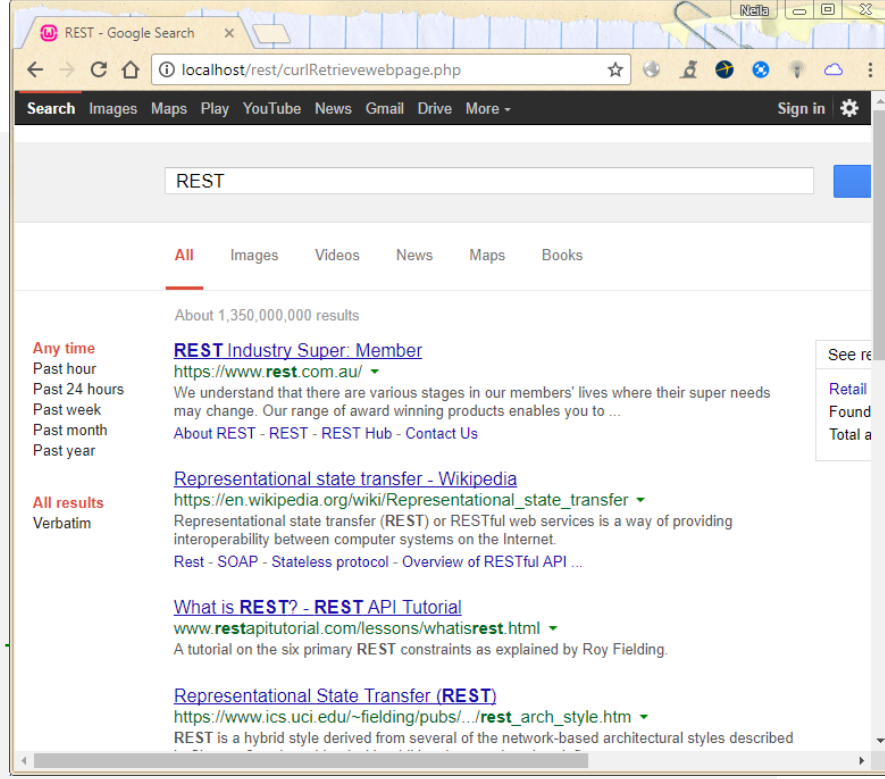
```php
<?php
class User {
public $name;
public $age;
public $salary;
}
$user = new User();
$user->name = 'John';
$user->age = 34;
$user->salary = 4200.50;
echo json_encode($user);
// {"name":"John","age":34,"salary":4200.5}
$employees = ['John', 'Mariya', 'Sarah', 'Marc'];
echo json_encode($employees);
// ["John","Mariya","Sarah","Marc"]


$user = json_decode('{"name":"John","age":34,"salary":4200.5}');
print_r($user);
//    stdClass Object
  //    (
// [name] => John
// [age] => 34
// [salary] => 4200.5
// )

?>
```

cURL/PHP

- **PHP/cURL**: The module for PHP that makes it possible for PHP programs to use libcurl.
- Libcurl is a C library that allows to connect and communicate to many different types of servers with many different types of protocols.
- It 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 authentication.
- Curl (read client URL) is a command-line tool for transferring data specified with URL syntax and libcurl is the library curl is using to do its job.

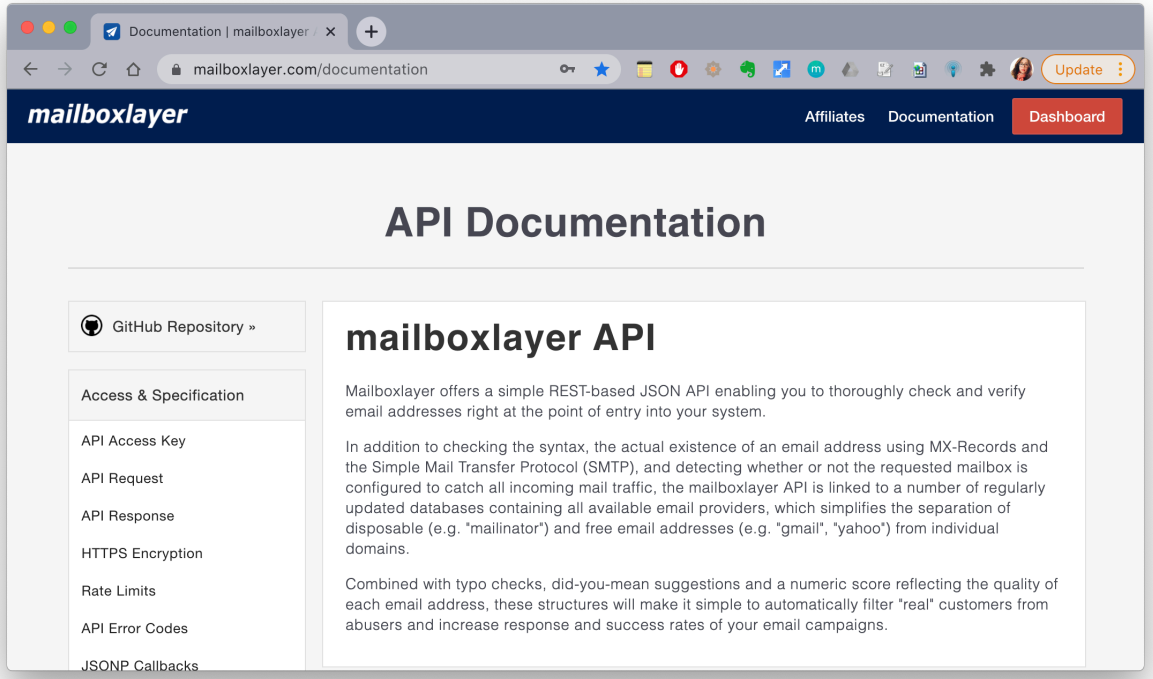
Retrieving Web page in PHP Using Curl

```php
<?php
//step1 : initialize a new session
$cSession = curl_init();
// From URL to get webpage contents.
//step2 set option for CURLOPT_URL.
//This value is the URL which we are sending the request to.
$url = "http://www.google.com/search?q=REST";
//Set option for CURLOPT_RETURNTRANSFER:
//true will tell curl to return the string instead of printing i
//Set option for CURLOPT_HEADER,
//false will tell curl to ignore the header in the return value.
curl_setopt($cSession,CURLOPT_URL,$url);
curl_setopt($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;
?>
```



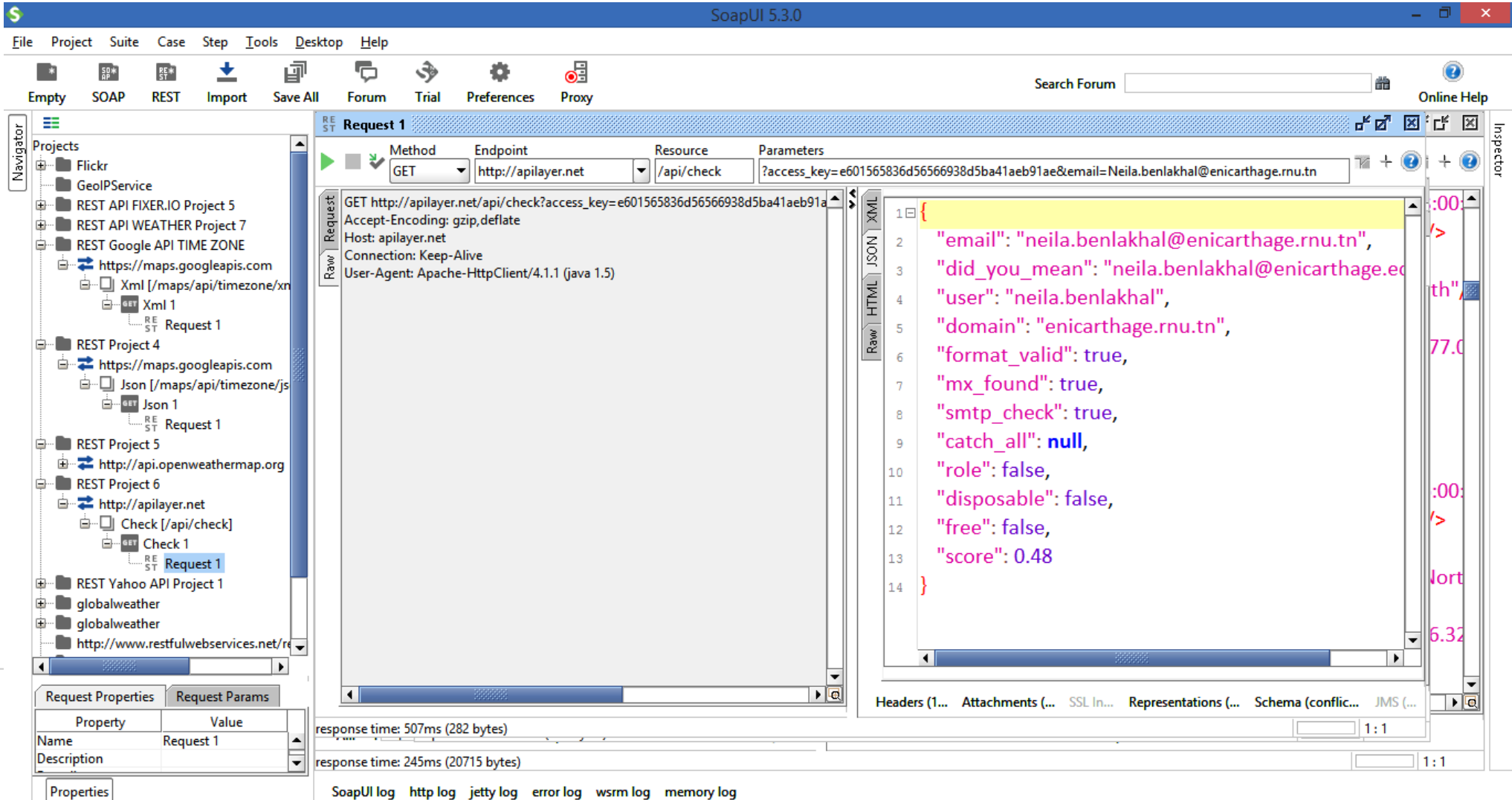
Curl client/restful web services (Json)

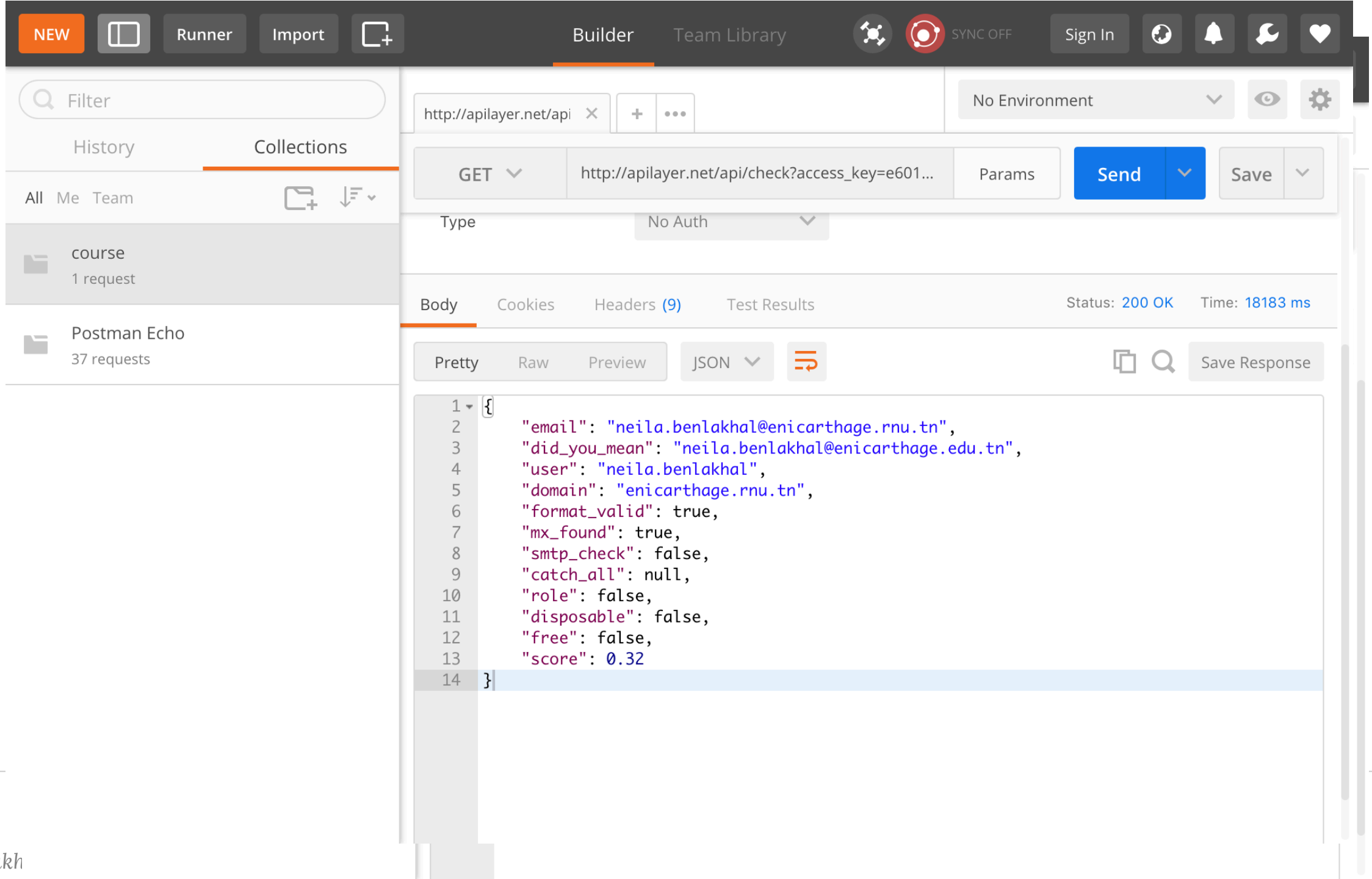
- https://mailboxlayer.com/documentation
- Mailboxlayer offers a simple REST-based JSON API enabling you to thoroughly check and verify email addresses right at the point of entry into your system.



Test via SOAPUI

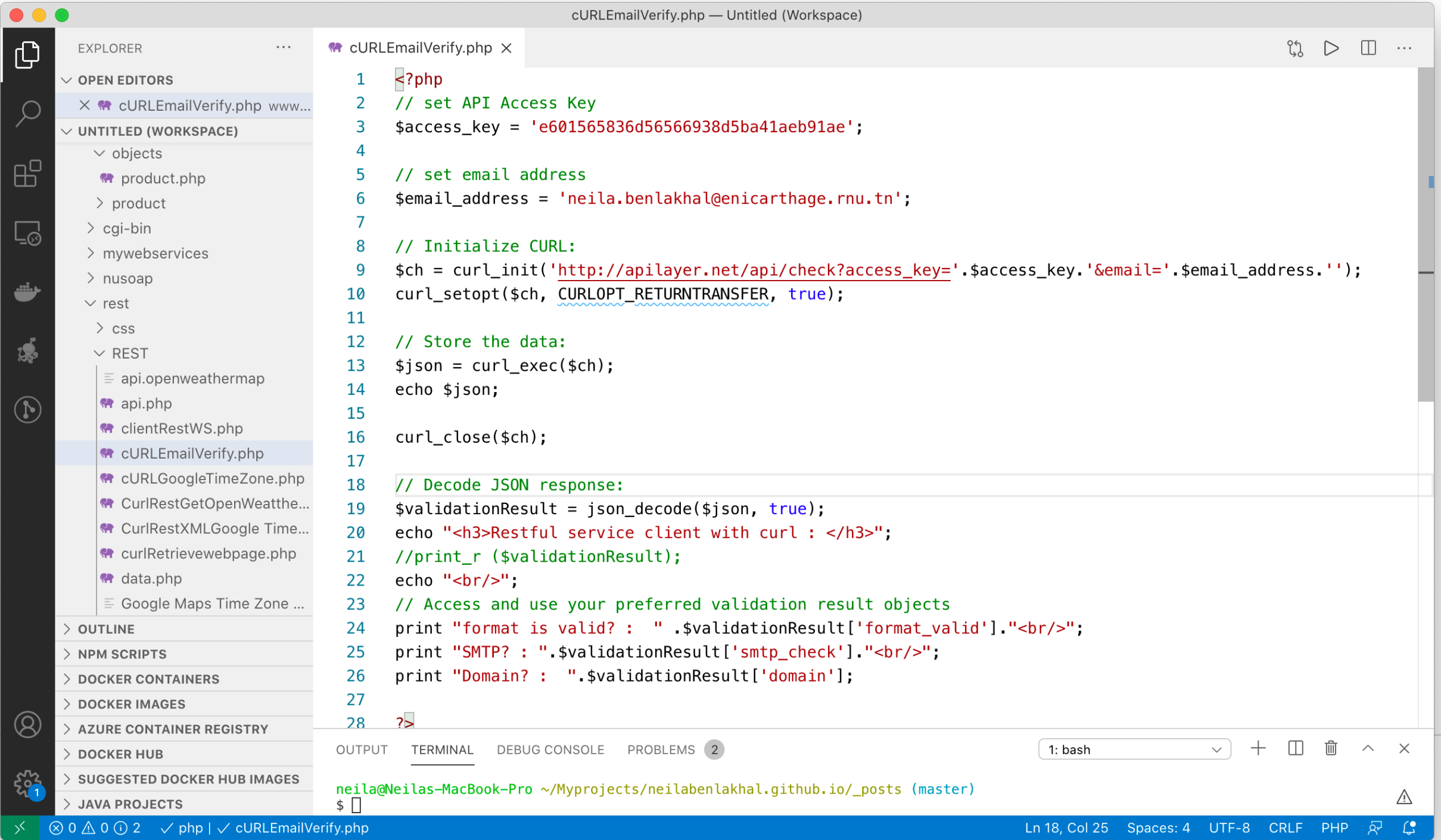
GET
http://apilayer.net/api/check?access_key=e601565836d56566938d5ba41aeb91ae&email=Neila.benlakhal@enicarthage.rnu.tn



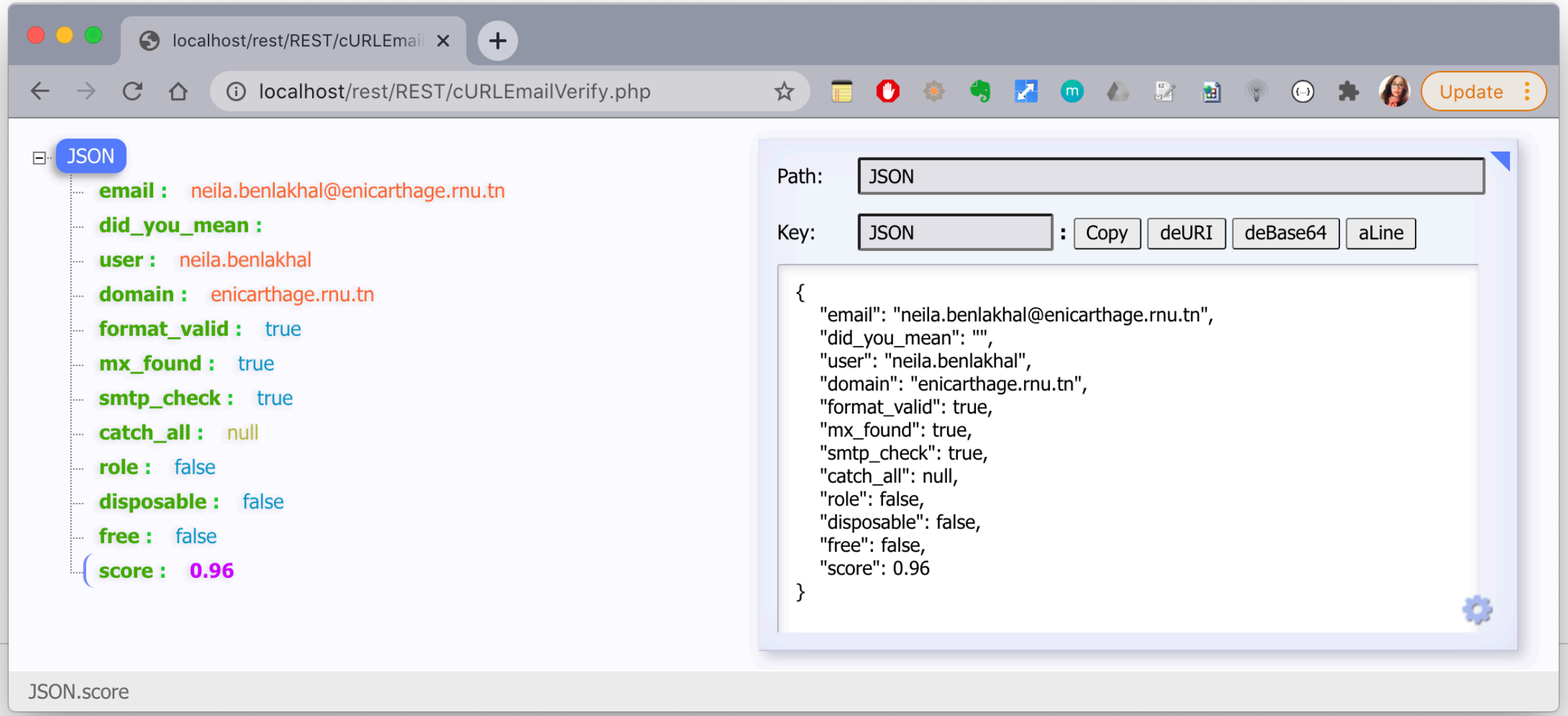

NEW
Runner
Import
Builder
Team Library
SYNC OFF
Sign In
Filter
History
Collections
All Me Team
course
1 request
Postman Echo
37 requests
http://apilayer.net/api
No Environment
GET
http://apilayer.net/api/check?access_key=e601...
Params
Send
Save
Type
No Auth
Body
Cookies
Headers (9)
Test Results
Status: 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.'&email='.$email_address.'');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
// Store the data:
$json = curl_exec($ch);
curl_close($ch); echo $json;
// Decode JSON response:
// $validationResult = json_decode($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']; ?>
```


cURLEmailVerify.php — Untitled (Workspace)
EXPLORER
OPEN EDITORS
cURLEmailVerify.php www...
UNTITLED (WORKSPACE)
objects
product.php
product
cgi-bin
mywebservices
nusoap
rest
css
REST
api.openweathermap
api.php
clientRestWS.php
cURLEmailVerify.php
cURLGoogleTimeZone.php
CurlRestGetOpenWeatthe...
CurlRestXMLGoogle Time...
curlRetrievewebpage.php
data.php
Google Maps Time Zone ...
OUTLINE
NPM SCRIPTS
DOCKER CONTAINERS
DOCKER IMAGES
AZURE CONTAINER REGISTRY
DOCKER HUB
SUGGESTED DOCKER HUB IMAGES
JAVA PROJECTS
cURLEmailVerify.php
1 <?php
2 // set API Access Key
3 $access_key = 'e601565836d56566938d5ba41aeb91ae';
4
5 // set email address
6 $email_address = 'neila.benlakhal@enicarthage.rnu.tn';
7
8 // Initialize CURL:
9 $ch = curl_init('http://apilayer.net/api/check?access_key='.$access_key.'&email='.$email_address.'');
10 curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
11
12 // Store the data:
13 $json = curl_exec($ch);
14 echo $json;
15
16 curl_close($ch);
17
18 // Decode JSON response:
19 $validationResult = json_decode($json, true);
20 echo "<h3>Restful service client with curl : </h3>";
21 //print_r ($validationResult);
22 echo "
";
23 // Access and use your preferred validation result objects
24 print "format is valid? : " .$validationResult['format_valid']."
";
25 print "SMTP? : ".$validationResult['smtp_check']."
";
26 print "Domain? : ".$validationResult['domain'];
27
28 ?>
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
1: bash
neila@Neilas-MacBook-Pro ~/Myprojects/neilabenlakhal.github.io/_posts (master)
$
Ln 18, Col 25
Spaces: 4
UTF-8
CRLF
PHP

(with json-handle chrome extension)



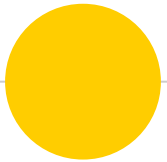
To call others REStful web services with others HTTP verbs :

```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/#!

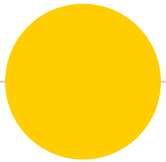
cURL GET request

```php
<?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];
?>
```

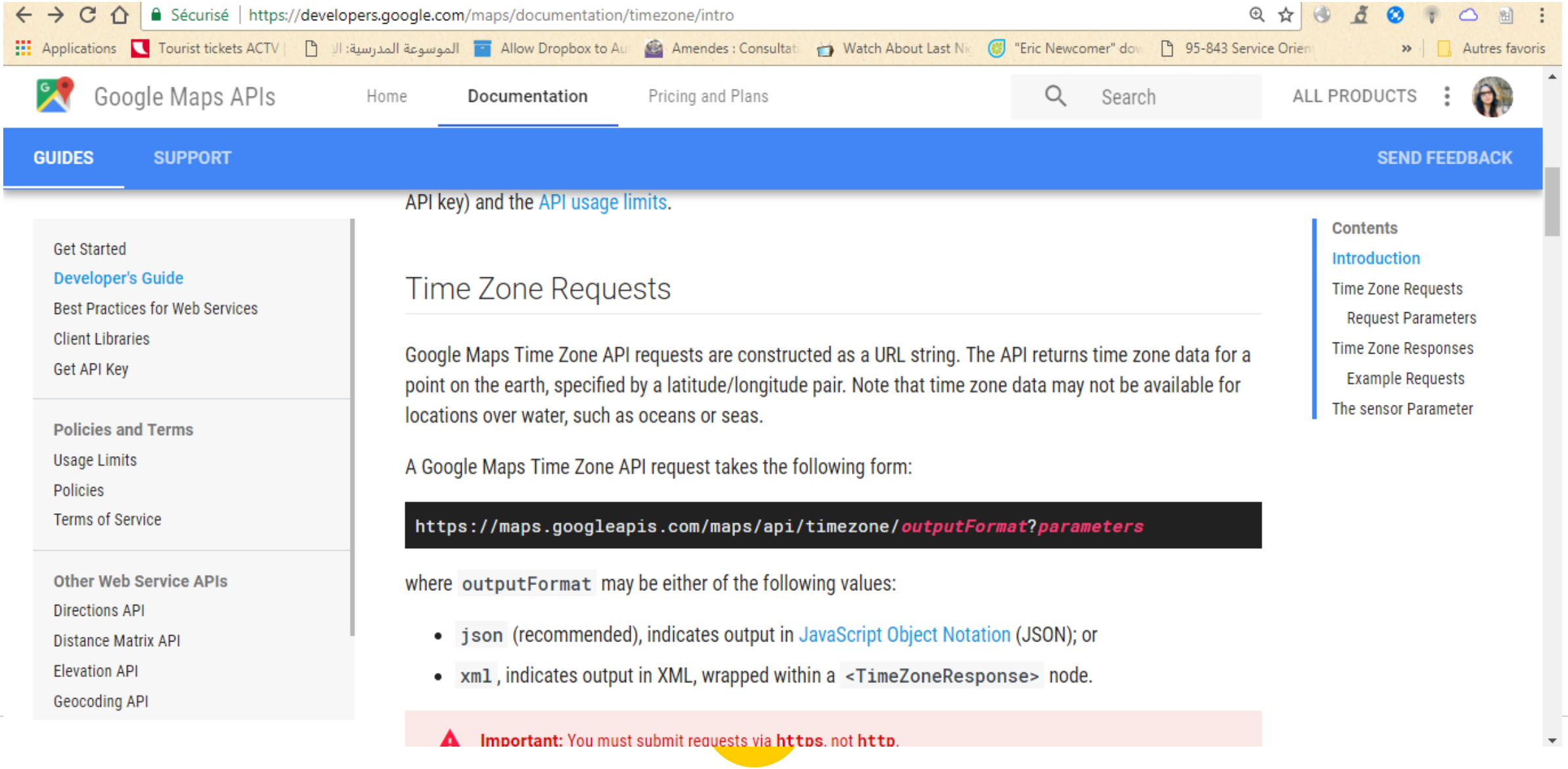


cURL POST request

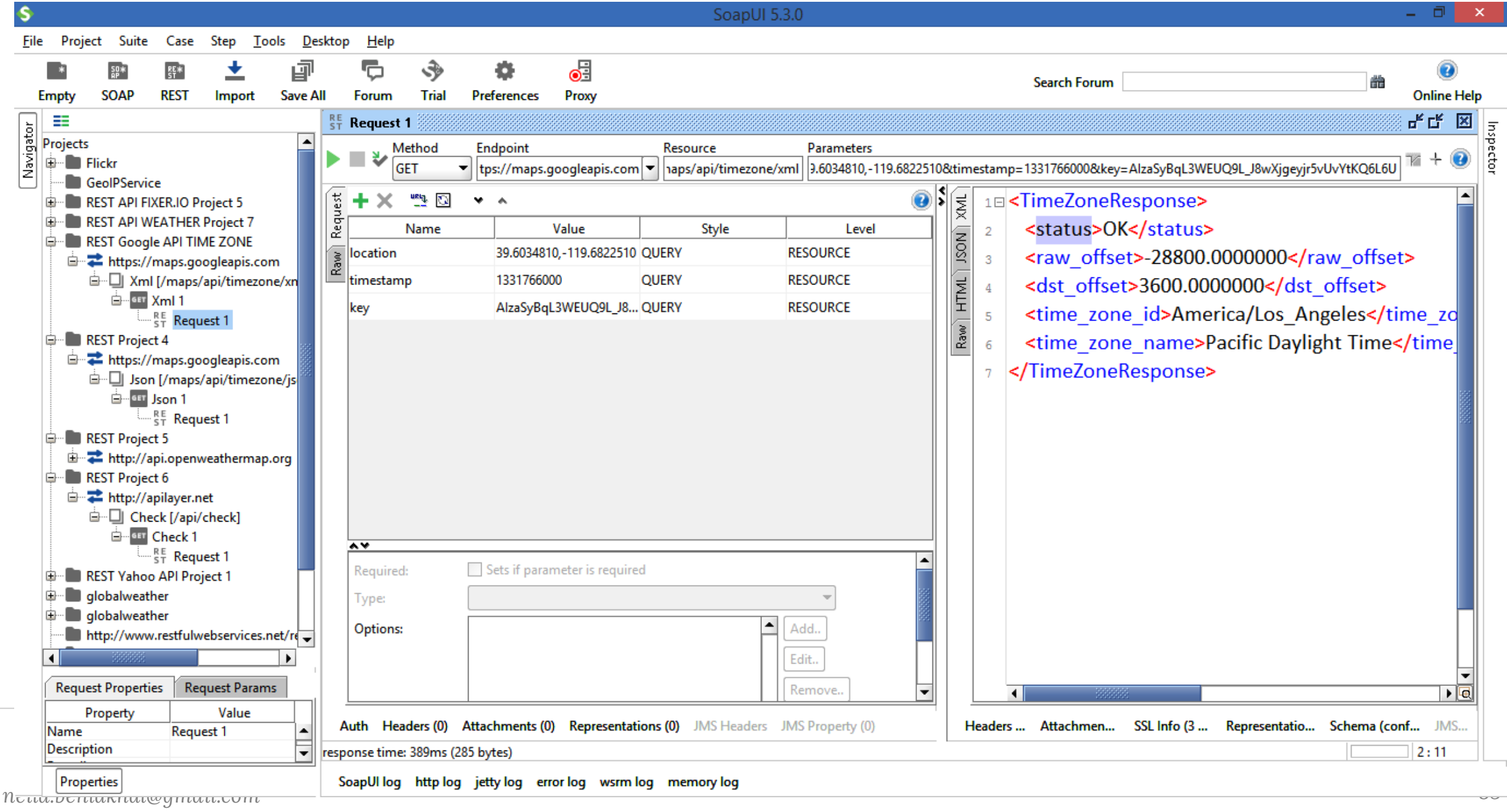
```php
<?php
$data_array =  array(
      "customer"          => $user['User']['customer_id'],
      "payment"           => array(
            "number"            => $this->request->data['account'],
            "routing"           => $this->request->data['routing'],
            "method"            => $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];
?>
```



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



Test via SOAPUI

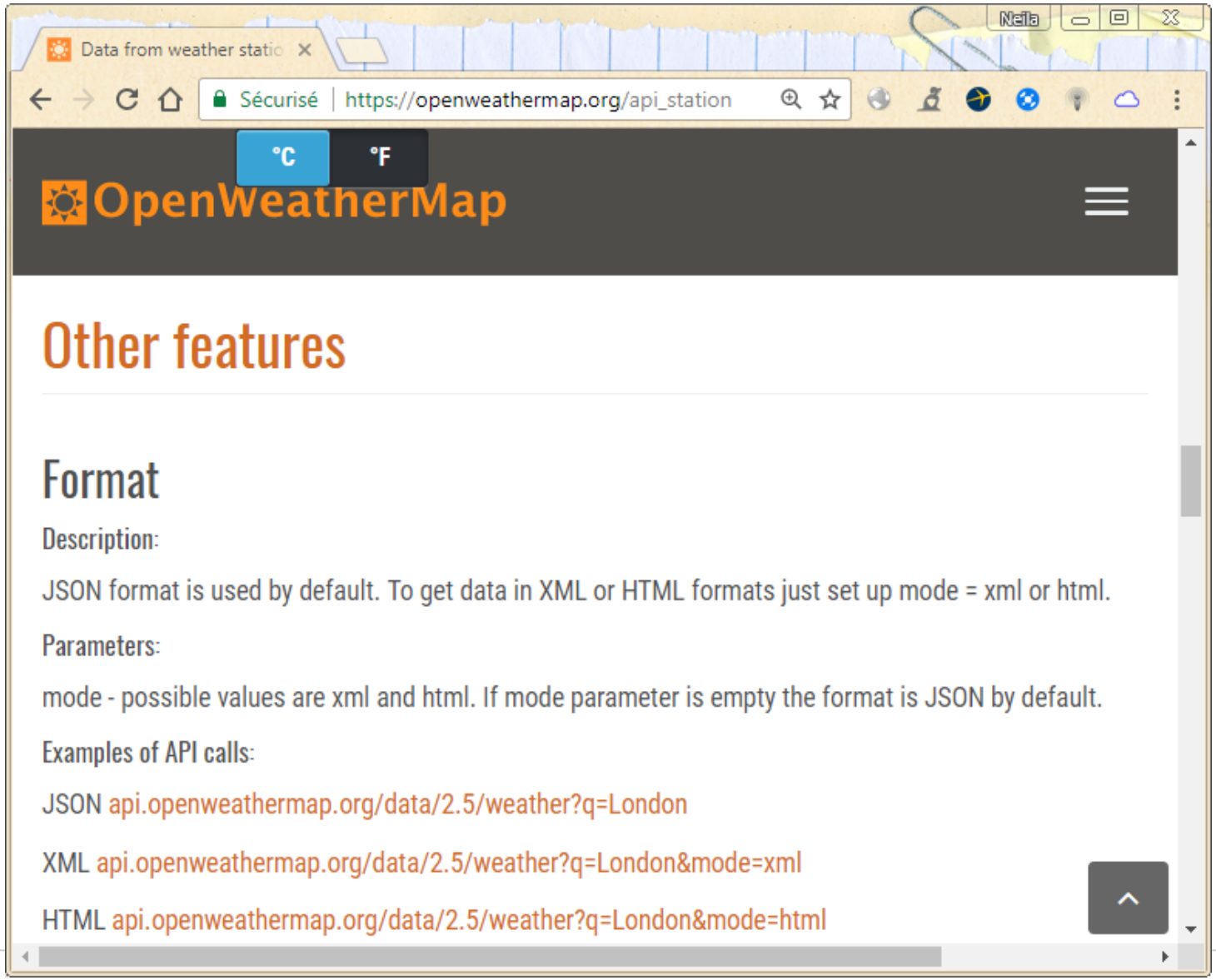


Client PhP (sans curl/ XML Parser)

```php
<?php $url =
"https://maps.googleapis.com/maps/api/timezone/xml?location=38.908133%2C-
77.047119&timestamp=1458000000&key=AIzaSyBqL3WEUQ9L_J8wXjgeyjr5vUvYtKQ6L6U";
$result = file_get_contents($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: GET
Endpoint: http://api.openweathermap.org
Resource: /data/2.5/forecast
Parameters: ?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">
      <symbol number="500" name="light rain" var="10n"/>
```

Headers (10) Attachments (0) SSL Info Representations (1) Schema (conflicts) JMS (0)

response time: 1420ms (20673 bytes) 414 : 31

SoapUI log http log jetty log error log wsrm log memory log

Inspector

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 :

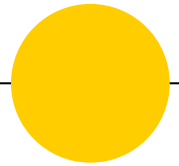
```
<?php
$url =
"http://api.openweathermap.org/data/2.5/forecast?id=2464
461&mode=xml&APPID=d57ea8ae4a87d8fb71a7cb680e74c029";
$result = file_get_contents($url);
echo $result; ?>
```

RESTFul service client in PhP: use file_get_contents 2/2

- Using the PHP file_get_contents() function, we can read files or API data but can not perform write, update, delete operations.

- Use cURL methods to perform all those operations using API data.

Implementing RESTful service (php)



Implement REST web service

- We will **create and consume simple REST API in PHP**

- To create a REST API, follow these steps:
  1. Create a Database and Table (MySQL)
  2. Create a Database Connection
  3. Create a REST API File

Database Creation 1/2

- Create a Database and Table (use phpMyAdmin):

```
CREATE DATABASE restDB;
```

```
CREATE TABLE IF NOT EXISTS `transactions` (
`id` int(20) NOT NULL AUTO_INCREMENT,
`order_id` int(50) NOT NULL,
`amount` decimal(9,2) NOT NULL,
`response_code` int(10) NOT NULL,
`response_desc` varchar(50) NOT NULL,
PRIMARY KEY (`id`),
UNIQUE KEY `order_id` (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;
```

Database creation 2/2

- Dumping Data in DATABASE :

```
INSERT INTO `transactions` (`id`, `order_id`, `amount`, `response_code`, `response_desc`) VALUES
(1, 15478952, 100.00, 0, 'PAID'),
(2, 15478955, 10.00, 0, 'PAID'),
(3, 15478958, 50.00, 1, 'FAILED'),
(4, 15478959, 60.00, 0, 'PAID');
```

- 2. Create a Database Connection

```
$con = mysqli_connect("localhost","root","",”restDB");
   if (mysqli_connect_errno()){
echo "Failed to connect to MySQL: " . mysqli_connect_error();
die();
}

```

Create a RESTful web service :

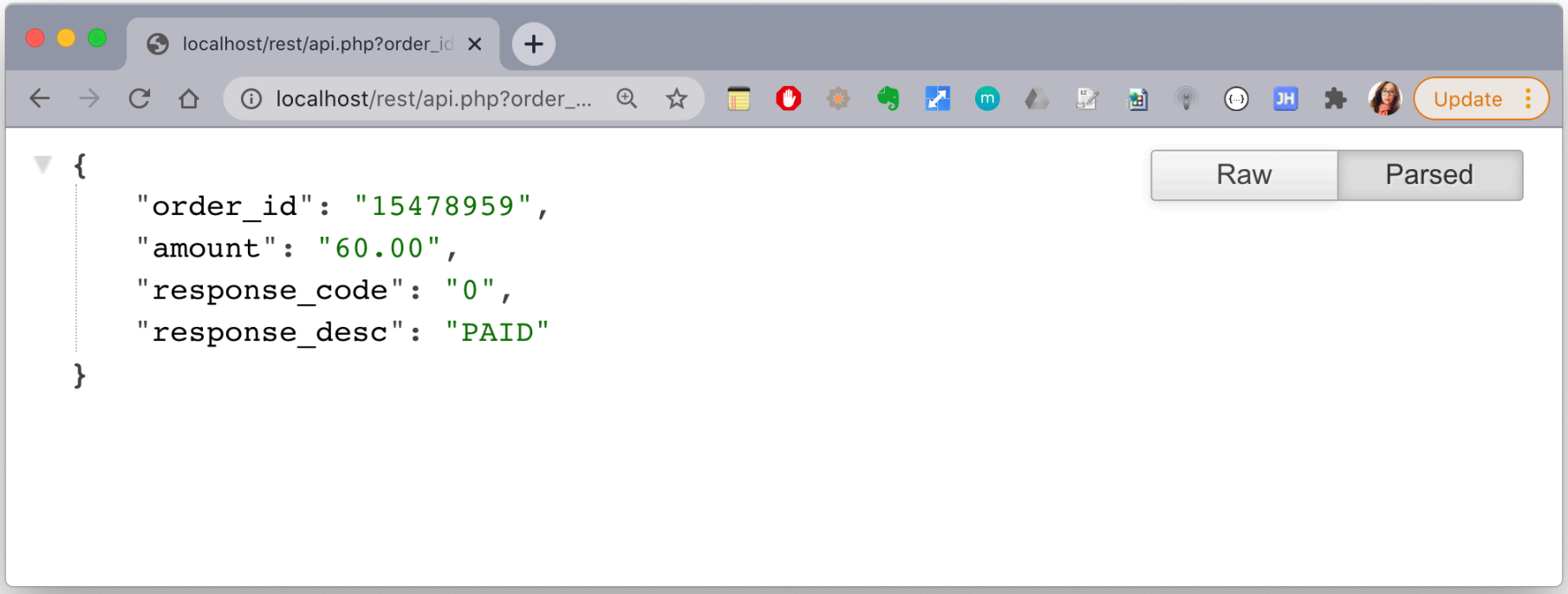
This script will accept the GET request and return output in the JSON format.

```php
<?php
header("Content-Type:application/json");
if (isset($_GET['order_id']) && $_GET['order_id']!="") {
include('db.php');
$order_id = $_GET['order_id'];
$result = mysqli_query(
$con,
"SELECT * FROM `transactions` WHERE order_id=$order_id");
if(mysqli_num_rows($result)>0){
$row = mysqli_fetch_array($result);
$amount = $row['amount'];
$response_code = $row['response_code'];
$response_desc = $row['response_desc'];
response($order_id, $amount, $response_code,$response_desc);
mysqli_close($con);
}else{response(NULL, NULL, 200,"No Record Found");}
}else{
response(NULL, NULL, 400,"Invalid Request");}
function response($order_id,$amount,$response_code,$response_desc){
$response['order_id'] = $order_id;
$response['amount'] = $amount;
$response['response_code'] = $response_code;
$response['response_desc'] = $response_desc;
$json_response = json_encode($response);
echo $json_response;}?>
```

Testing API

- Test the API (postman then navigator) with the following request :

```
1 http://localhost/rest/api.php?order_id=15478959
```



Changing URI to be conform to REST specification

```
1 http://localhost/rest/api.php?order_id=15478959
```

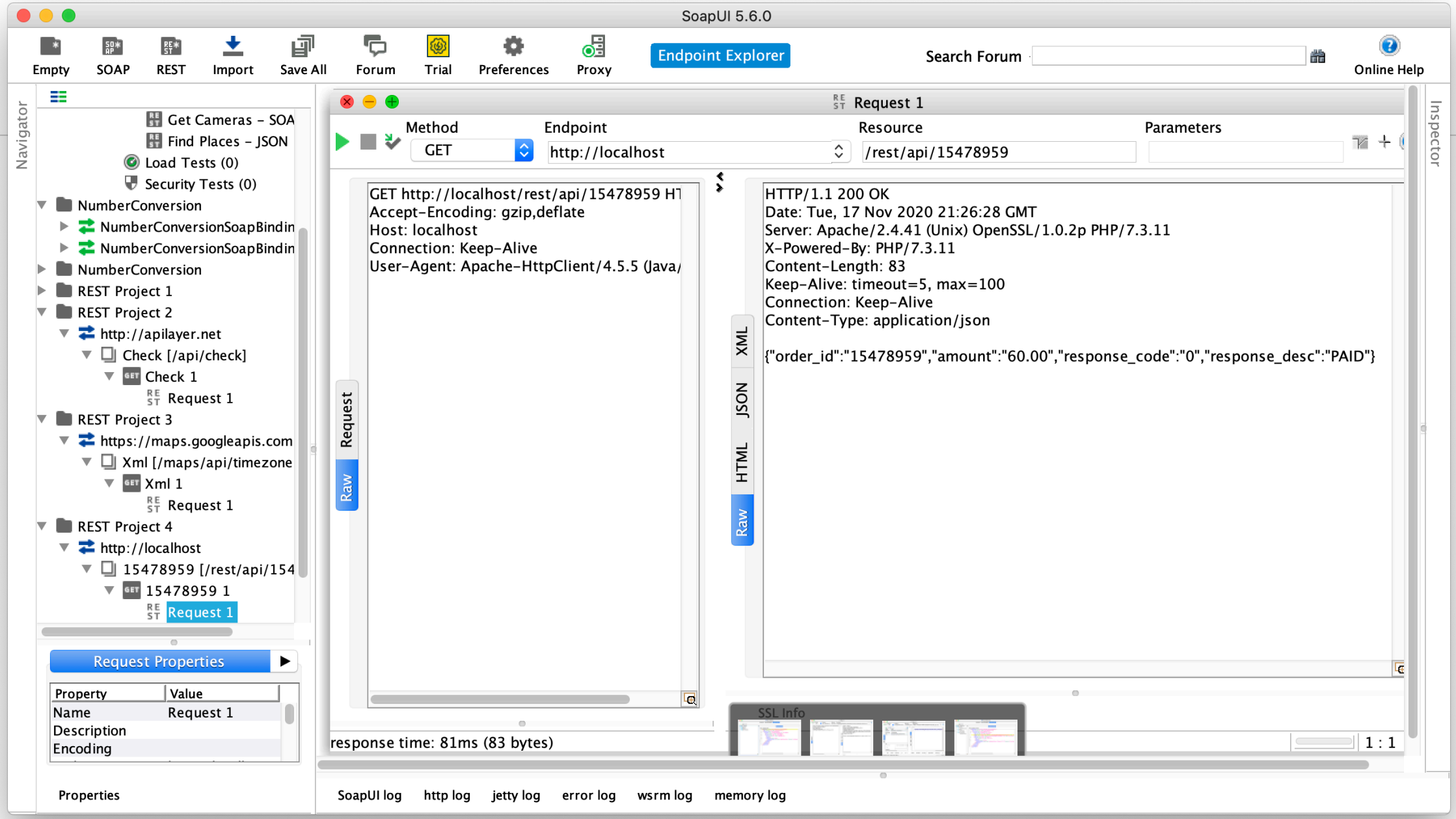
- Above URI does not respect RESTful architecture URI structure, therefore we will rewrite URL through the .htaccess file, copy paste the following rule in .htaccess file ; add file to the same folder as the API :

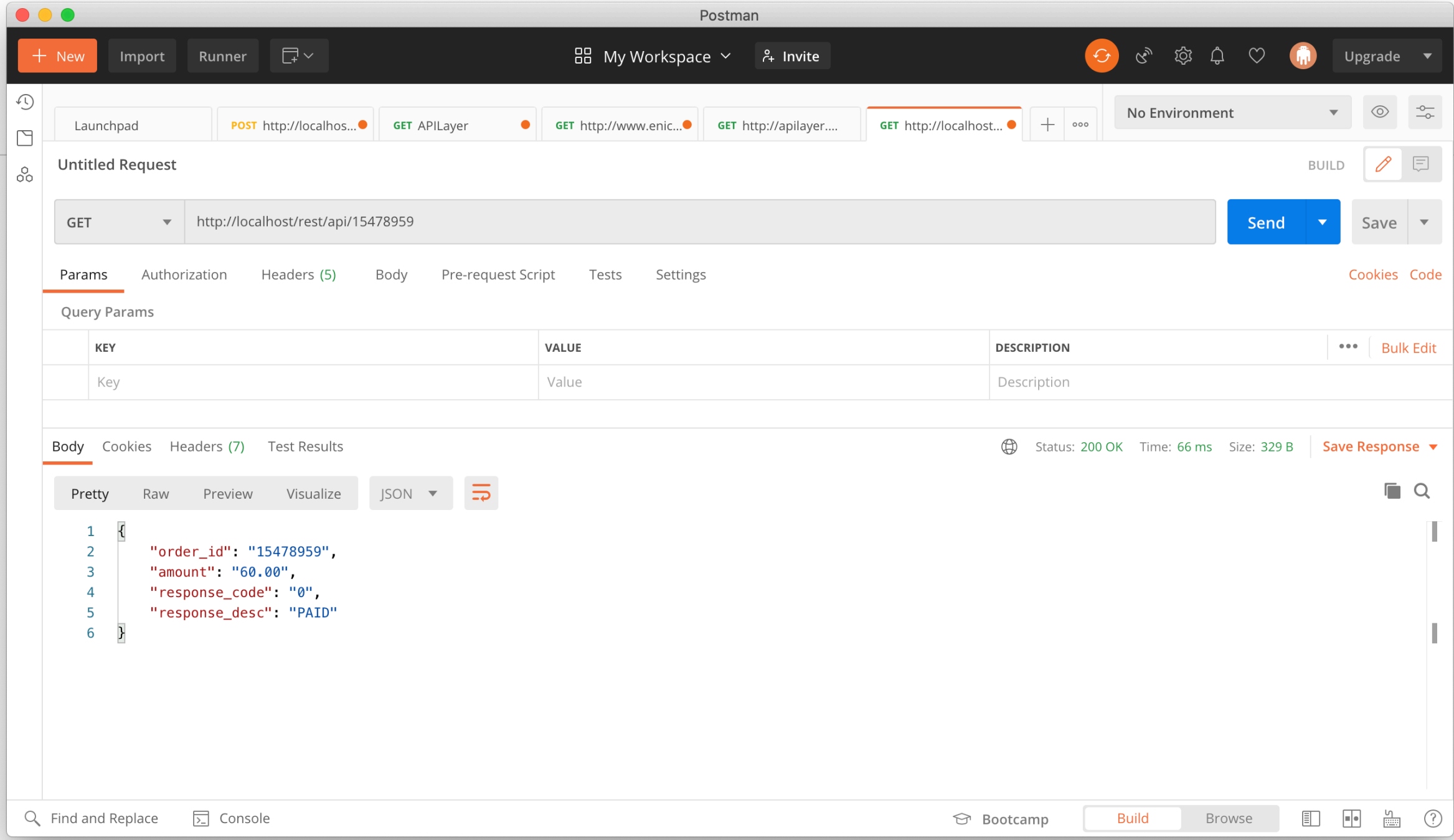
client.php | db.php | api.php | .htaccess

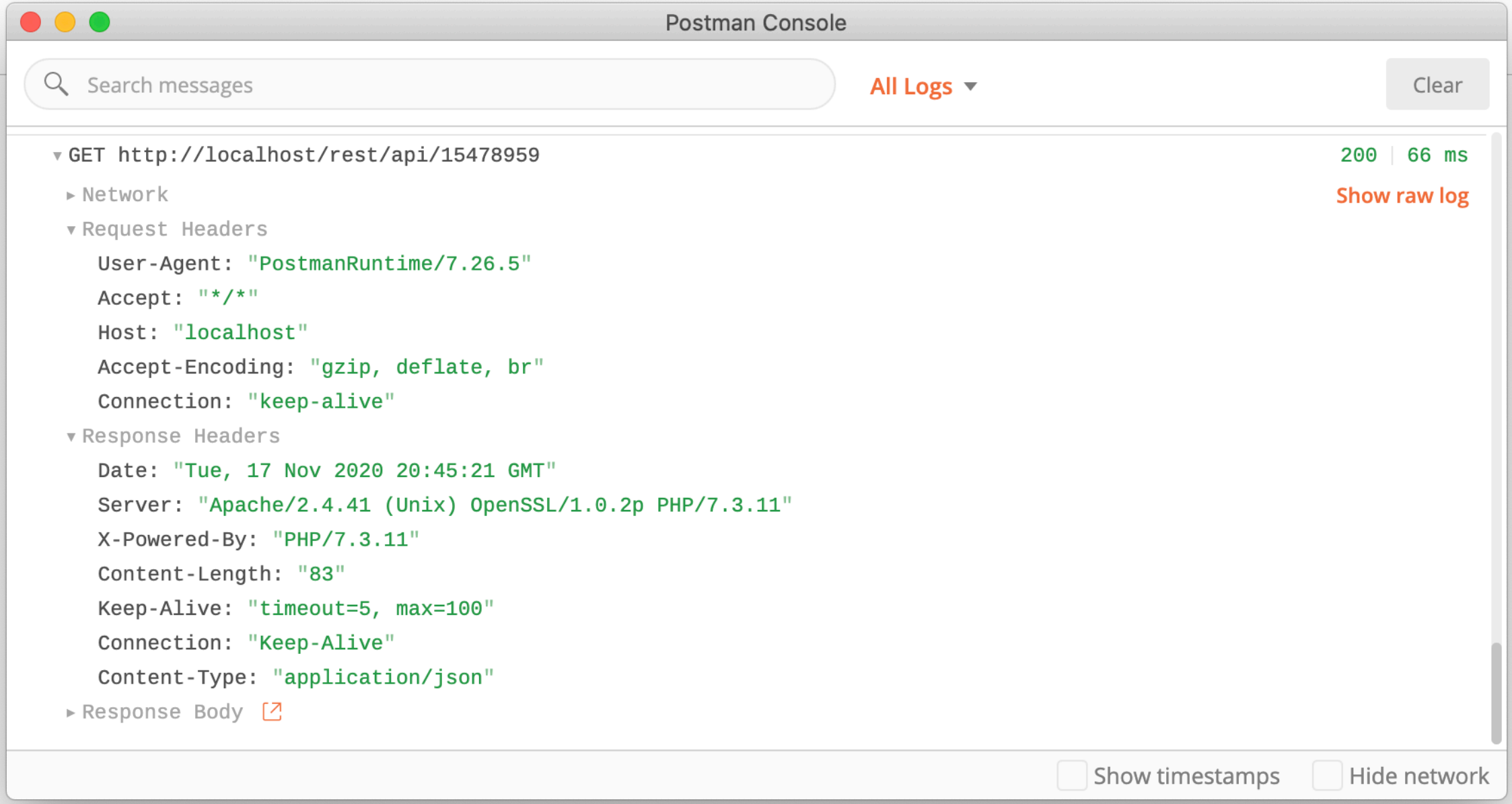
```
1   RewriteEngine On    # Turn on the rewriting engine
2
3   RewriteRule ^api/([0-9a-zA-Z_-]*)$ api.php?order_id=$1 [NC,L]
```

- Now you can get the transaction information by browsing the following URL:

```
1 http://localhost/rest/api/15478959
```


SoapUI 5.6.0
Empty
SOAP
REST
Import
Save All
Forum
Trial
Preferences
Proxy
Endpoint Explorer
Search Forum
Online Help
Navigator
Get Cameras - SOA
Find Places - JSON
Load Tests (0)
Security Tests (0)
NumberConversion
NumberConversionSoapBindin
NumberConversionSoapBindin
NumberConversion
REST Project 1
REST Project 2
http://apilayer.net
Check [/api/check]
Check 1
Request 1
REST Project 3
https://maps.googleapis.com
Xml [/maps/api/timezone
Xml 1
Request 1
REST Project 4
http://localhost
15478959 [/rest/api/154
15478959 1
Request 1
Request Properties
Property
Value
Name
Request 1
Description
Encoding
Properties
Request 1
Method
Endpoint
Resource
Parameters
GET
http://localhost
/rest/api/15478959
GET http://localhost/rest/api/15478959 HT
Accept-Encoding: gzip,deflate
Host: localhost
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.5 (Java/
Request
Raw
HTTP/1.1 200 OK
Date: Tue, 17 Nov 2020 21:26:28 GMT
Server: Apache/2.4.41 (Unix) OpenSSL/1.0.2p PHP/7.3.11
X-Powered-By: PHP/7.3.11
Content-Length: 83
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
{"order_id":"15478959","amount":"60.00","response_code":"0","response_desc":"PAID"}
XML
JSON
HTML
Raw
SSL Info
response time: 81ms (83 bytes)
1 : 1
Inspector
SoapUI log
http log
jetty log
error log
wsrm log
memory log


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Launchpad
POST http://localhos...
GET APILayer
GET http://www.enic...
GET http://apilayer....
GET http://localhost...
No Environment
Untitled Request
BUILD
GET
http://localhost/rest/api/15478959
Send
Save
Params
Authorization
Headers (5)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 66 ms
Size: 329 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"order_id": "15478959",
"amount": "60.00",
"response_code": "0",
"response_desc": "PAID"
}
Find and Replace
Console
Bootcamp
Build
Browse


Postman Console
Search messages
All Logs
Clear
GET http://localhost/rest/api/15478959
200 | 66 ms
Network
Show raw log
Request Headers
User-Agent: "PostmanRuntime/7.26.5"
Accept: "*/*"
Host: "localhost"
Accept-Encoding: "gzip, deflate, br"
Connection: "keep-alive"
Response Headers
Date: "Tue, 17 Nov 2020 20:45:21 GMT"
Server: "Apache/2.4.41 (Unix) OpenSSL/1.0.2p PHP/7.3.11"
X-Powered-By: "PHP/7.3.11"
Content-Length: "83"
Keep-Alive: "timeout=5, max=100"
Connection: "Keep-Alive"
Content-Type: "application/json"
Response Body
Show timestamps
Hide network

RESTful web service request handling steps 1/4

```
1 http://localhost/rest/api/15478959
```

◉Let's see what happens when a client requests for the restDB with the aforementioned URI.

1. A php client makes an HTTP request with the GET method type and 15478959 (order_id) the identifier for the order.

2. The client sets the representation type that it can handle through the Accept request header field.

RESTful web service request handling steps 2/4

3. This request message is self-descriptive:

- It uses a standard method (the GET method in this example) with known semantics for retrieving the content)
- The content-type is a set to a well-known media type.
- This request also declares the acceptable response format

Request Headers

Full URL: http://localhost/rest/client.php
Upgrade-Insecure-Requests : 1
DNT : 1
User-Agent : Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.193 Safari/537.36
Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Response Headers

Full URL: http://localhost/rest/client.php
Date : Tue, 17 Nov 2020 21:44:06 GMT
Server : Apache/2.4.41 (Unix) OpenSSL/1.0.2p PHP/7.3.11
X-Powered-By : PHP/7.3.11
Content-Length : 627
Keep-Alive : timeout=5, max=100
Connection : Keep-Alive
Content-Type : text/html; charset=UTF-8

RESTful web service request handling steps 3/4

4. The web server receives and interprets the GET request to be a retrieve action. At this point, the web server passes control to the underlying RESTful framework to handle the request.
5. Note that RESTful frameworks do not automatically retrieve resources, as that is not their job. The job of a framework is to ease the implementation of the REST constraints. Business logic and storage implementation is the role of the domain-specific PhP code.
6. The server-side program looks for the resource. Finding the resource could mean looking for it in some data store such as a database (restDB), a filesystem, or even a call to a different web service.

RESTful web service request handling steps 4/4

7. Once the program finds the resource details, it converts the binary data of the resource to the client's requested representation. In this example, we use the JSON representation for the resource.
8. With the representation converted to JSON, the server sends back an HTTP response with a numeric code of 200 together with the JSON representation as the payload.
9. Note that if there are any errors, the HTTP server reports back the proper numeric code, but it is up to the client to correctly deal with the failure. Similar to the request message, the response is also self-descriptive.

Network Sniffer

filter : localhost/rest/client.php

on capture apply clear filter clear result

requestId	statusCode	ip	url	type	timeStamp	tabId	fromCache	method
127262	200	::1	http://localhost/rest/client.php	main_frame	1605649446926.1729	2435	false	GET

Request Headers

Full URL: http://localhost/rest/client.php
Upgrade-Insecure-Requests : 1
DNT : 1
User-Agent : Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.193 Safari/537.36
Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Response Headers

Full URL: http://localhost/rest/client.php
Date : Tue, 17 Nov 2020 21:44:06 GMT
Server : Apache/2.4.41 (Unix) OpenSSL/1.0.2p PHP/7.3.11
X-Powered-By : PHP/7.3.11
Content-Length : 627
Keep-Alive : timeout=5, max=100
Connection : Keep-Alive
Content-Type : text/html; charset=UTF-8

Consuming RESTFUL web service 1/2

To consume a REST API, follow these steps:

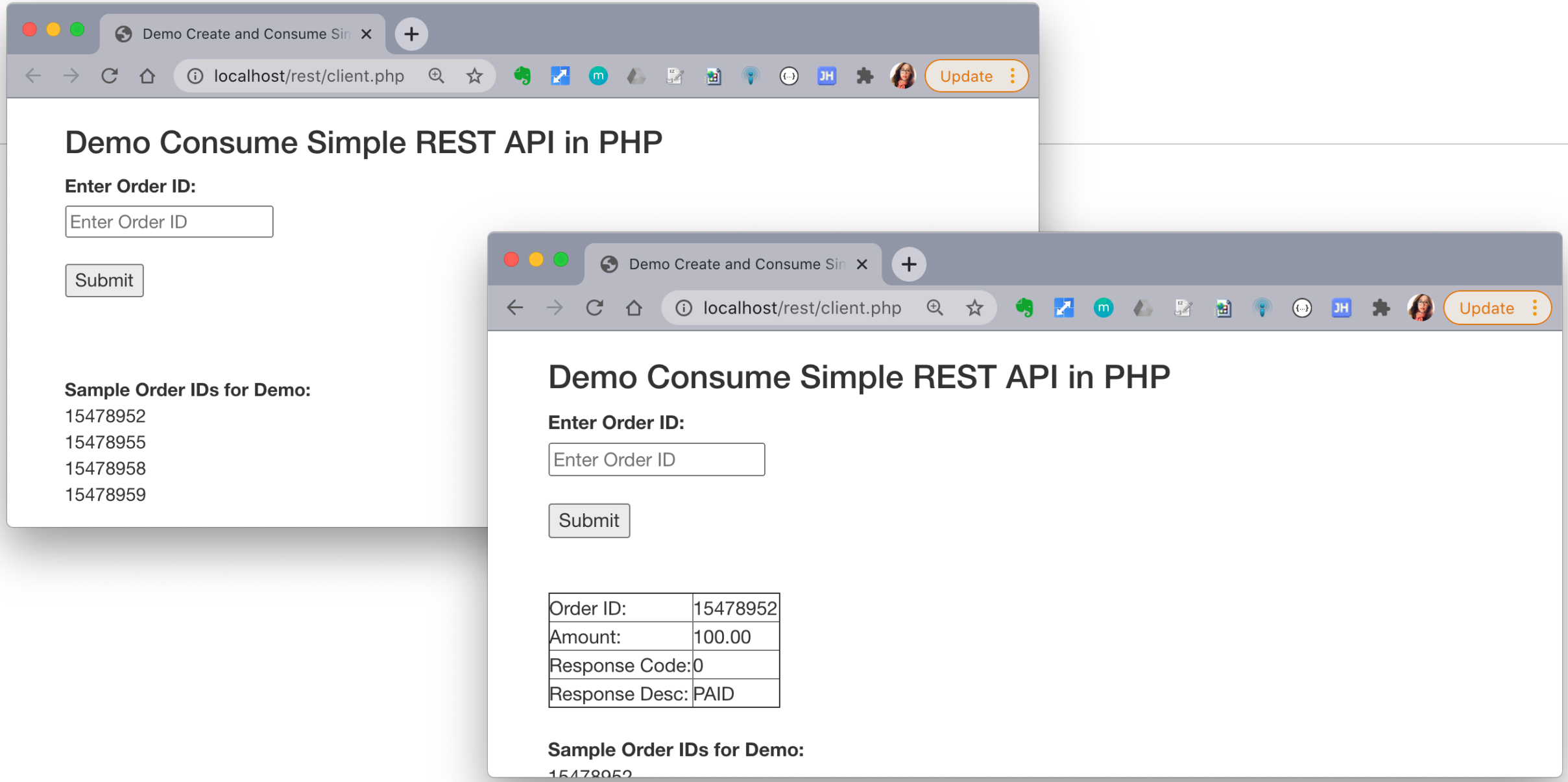
- 1. Create an Index File with HTML Form

```
<form action="" method="POST">
<label>Enter Order ID:</label><br />
<input type="text" name="order_id" placeholder="Enter Order ID" required/>
<br /><br />
<input type="submit" name="submit"/>
</form>
```

Consuming RESTFUL web service 2/2

- 2. Fetch Records through CURL

```php
<?php
if (isset($_POST['order_id']) && $_POST['order_id']!="") {
$order_id = $_POST['order_id'];
$url = "http://localhost/rest/api/".$order_id;
$client = curl_init($url);
curl_setopt($client,CURLOPT_RETURNTRANSFER,true);
$response = curl_exec($client);
$result = json_decode($response);
echo "<table>";
echo "<tr><td>Order ID:</td><td>$result->order_id</td></tr>";
echo "<tr><td>Amount:</td><td>$result->amount</td></tr>";
echo "<tr><td>Response Code:</td><td>$result-
>response_code</td></tr>";
echo "<tr><td>Response Desc:</td><td>$result-
>response_desc</td></tr>";
echo "</table>";
}
  ?>
```



https://www.allphptricks.com/create-and-consume-simple-rest-api-in-php/

Where to find REST WS ?

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

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.

Hybrid REST-RPC oriented 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.

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: GET | Endpoint: https://api.flickr.com | Resource: /services/rest | Parameters: ?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)

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)

response time: 1024ms (3734 bytes) 1 : 1

SoapUI log http log jetty log error log wsrm log memory log

Inspector

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