Credits See original tutorial here: https://phppot.com/php/php-restful-web-service/
Download code from here: https://phppot.com/wp-content/uploads/2015/10/restexample.zip
Download CURL from here: https://curl.haxx.se/download.html
https://winampplugins.co.uk/download.php?file=curl/curl_7_53_1_openssl_nghttp2_x64.7z
https://winampplugins.co.uk/download.php?file=curl/curl_7_53_1_openssl_nghttp2_x86.7z
(NOTE: IF YOU ARE USING WINDOWS 10, THEN YOU DO NOT NEED TO DOWNLOAD CURL)

Example 8 – PHP RESTful Web Service

Representational State Transfer (REST) is an architecture style. Web services that follow the RESTful principles are RESTful services. In RESTful services, URIs are used to access the resources. Data and functions are called resources in the RESTful glossary. So eventually data and function are what we will access via services.

In this RESTful web services tutorial, I will show you how to create a RESTful web service. I will not be using any framework for this, just plain PHP. Most of the times I do prefer to write custom code **without depending on frameworks**, this approach has its own advantages.

On the Internet, I have seen web services tutorials and most of the times they all turn out to be error-prone or incomplete. I tested those RESTful services using a REST client and mostly they fail.

Objectives of this RESTful Web Service Example

- Build a RESTful Webservice.
- With plain PHP, without dependency on any framework.
- URI patterns should follow REST principles.
- RESTful service should be capable responding to requests for formats like JSON, XML.
- Should demonstrate the use of HTTP Status code based on different scenarios.
- Demonstrate the use of Request headers.
- Test the RESTful web service using a REST client.

RESTful Webservice Example

Following is a domain class used to demonstrate the RESTful services.

```
<?php
/* A domain Class to demonstrate RESTful web services */
Class Mobile {

      private $mobiles = array(
            1 => 'Apple iPhone 6S',
            2 => 'Samsung Galaxy S6',
            3 => 'Apple iPhone 6S Plus',
            4 => 'LG G4',
            5 => 'Samsung Galaxy S6 edge',
            6 => 'OnePlus 2');

      /*you should hookup the DAO here*/
      public function getAllMobile(){
            return $this->mobiles;
      }

      public function getMobile($id){
            $mobile = array($id => ($this->mobiles[$id]) ? $this->mobiles[$id] : $this-
>mobiles[1]);
            return $mobile;
      }
}
```

```
?>
```

RESTful Services URI Mapping

RESTful web services should have a uniform, neat URIs. HTACCESS file is used to do the URL mapping between the request and the real file. In this example, we are using two URIs.

To get the list of all mobiles.

http://localhost/restexample/mobile/list/

To get a particular mobile's detail using its id. In the below '2' is the id of a mobile.

http://localhost/restexample/mobile/list/2/

Following is the file that `.htaccess` maps the request URL and forwards the request to a PHP file.

```
# Turn rewrite engine on
Options +FollowSymlinks
RewriteEngine on

# map neat URL to internal URL
RewriteRule ^mobile/list/$   RestController.php?view=all [nc,qsa]
RewriteRule ^mobile/list/([0-9]+)/$   RestController.php?view=single&id=$1 [nc,qsa]
```

RESTful Web Service Controller

In file `.htaccess`, we are forwarding all the request to file RestController.php with a key named 'view' to identify the request. Following is the RestController.php file that receives the request dispatches to respective methods to handle the request. 'view' key is used to identify the URL request.

```
<?php
require_once("MobileRestHandler.php");

$view = "";
if(isset($_GET["view"]))
	$view = $_GET["view"];
/*
controls the RESTful services
URL mapping
*/
switch($view){

	case "all":
		// to handle REST Url /mobile/list/
		$mobileRestHandler = new MobileRestHandler();
		$mobileRestHandler->getAllMobiles();
		break;

	case "single":
		// to handle REST Url /mobile/show/<id>/
		$mobileRestHandler = new MobileRestHandler();
		$mobileRestHandler->getMobile($_GET["id"]);
		break;

	case "" :
		//404 - not found;
		break;
```

```
}
?>
```

A simple RESTful base class

Following class has a couple of methods that can be commonly used in all REStful service handlers. One method is used to construct the response and another method is to hold the different HTTP status code and its respective messages. Such common methods can be added to this class and this can be made a base class for all RESTful handler classes.

```
<?php
/* A simple RESTful webservices base class
Use this as a template and build upon it */
class SimpleRest {

	private $httpVersion = "HTTP/1.1";

	public function setHttpHeaders($contentType, $statusCode){

		$statusMessage = $this -> getHttpStatusMessage($statusCode);

		header($this->httpVersion. " ". $statusCode ." ". $statusMessage);
		header("Content-Type:". $contentType);
	}

	public function getHttpStatusMessage($statusCode){
		$httpStatus = array(
			100 => 'Continue',
			101 => 'Switching Protocols',
			200 => 'OK',
			201 => 'Created',
			202 => 'Accepted',
			203 => 'Non-Authoritative Information',
			204 => 'No Content',
			205 => 'Reset Content',
			206 => 'Partial Content',
			300 => 'Multiple Choices',
			301 => 'Moved Permanently',
			302 => 'Found',
			303 => 'See Other',
			304 => 'Not Modified',
			305 => 'Use Proxy',
			306 => '(Unused)',
			307 => 'Temporary Redirect',
			400 => 'Bad Request',
			401 => 'Unauthorized',
			402 => 'Payment Required',
			403 => 'Forbidden',
			404 => 'Not Found',
			405 => 'Method Not Allowed',
			406 => 'Not Acceptable',
			407 => 'Proxy Authentication Required',
			408 => 'Request Timeout',
			409 => 'Conflict',
			410 => 'Gone',
			411 => 'Length Required',
```

```
                412 => 'Precondition Failed',
                413 => 'Request Entity Too Large',
                414 => 'Request-URI Too Long',
                415 => 'Unsupported Media Type',
                416 => 'Requested Range Not Satisfiable',
                417 => 'Expectation Failed',
                500 => 'Internal Server Error',
                501 => 'Not Implemented',
                502 => 'Bad Gateway',
                503 => 'Service Unavailable',
                504 => 'Gateway Timeout',
                505 => 'HTTP Version Not Supported');
        return ($httpStatus[$statusCode]) ? $httpStatus[$statusCode] : $status[500];
    }
}
?>
```

RESTful Web Service Handler

This is the class that handles the REST request. You need to note a couple of things carefully. First one is how the REST handler decides what kind of response format should be sent back. This is decided based on the Request Header parameter “Accept”. The protocol here is, when the request is sent, it should set the Request header parameter “Accept” and send it. The values can be like “application/json” or “application/xml” or “text/html”. The second thing you should note is the usage of Status codes. For success, status code 200 should be set in response and sent. Similarly, there are different status codes available and they should be used according to the situation.

```
<?php
require_once("SimpleRest.php");
require_once("Mobile.php");

class MobileRestHandler extends SimpleRest {
    function getAllMobiles() {
        $mobile = new Mobile();
        $rawData = $mobile->getAllMobile();
        if(empty($rawData)) {
            $statusCode = 404;
            $rawData = array('error' => 'No mobiles found!');
        } else {
            $statusCode = 200;
        }
        $requestContentType = $_SERVER['HTTP_ACCEPT'];
        $this ->setHttpHeaders($requestContentType, $statusCode);
        if(strpos($requestContentType,'application/json') !== false){
            $response = $this->encodeJson($rawData);
            echo $response;
        } else if(strpos($requestContentType,'text/html') !== false){
            $response = $this->encodeHtml($rawData);
            echo $response;
        } else if(strpos($requestContentType,'application/xml') !== false){
            $response = $this->encodeXml($rawData);
            echo $response;
        }
    }
    public function encodeHtml($responseData) {
```

```
            $htmlResponse = "<table border='1'>";
            foreach($responseData as $key=>$value) {
                  $htmlResponse .= "<tr><td>". $key. "</td><td>". $value. "</td></tr>";
            }
            $htmlResponse .= "</table>";
            return $htmlResponse;
      }
      public function encodeJson($responseData) {
            $jsonResponse = json_encode($responseData);
            return $jsonResponse;
      }
      public function encodeXml($responseData) {
            // creating object of SimpleXMLElement
            $xml = new SimpleXMLElement('<?xml version="1.0"?><mobile></mobile>');
            foreach($responseData as $key=>$value) {
                  $xml->addChild($key, $value);
            }
            return $xml->asXML();
      }
      public function getMobile($id) {
            $mobile = new Mobile();
            $rawData = $mobile->getMobile($id);
            if(empty($rawData)) {
                  $statusCode = 404;
                  $rawData = array('error' => 'No mobiles found!');
            } else {
                  $statusCode = 200;
            }
            $requestContentType = $_SERVER['HTTP_ACCEPT'];
            $this ->setHttpHeaders($requestContentType, $statusCode);
            if(strpos($requestContentType,'application/json') !== false){
                  $response = $this->encodeJson($rawData);
                  echo $response;
            } else if(strpos($requestContentType,'text/html') !== false){
                  $response = $this->encodeHtml($rawData);
                  echo $response;
            } else if(strpos($requestContentType,'application/xml') !== false){
                  $response = $this->encodeXml($rawData);
                  echo $response;
            }
      }
}
?>
```

RESTful Web Service Client

(a) use curl.exe

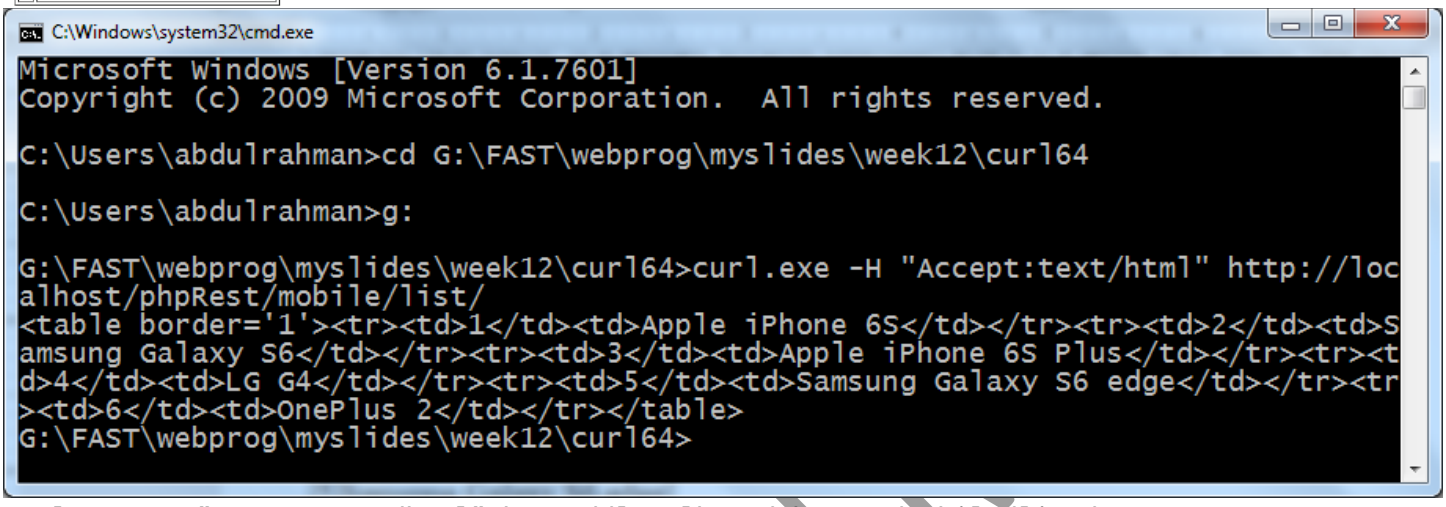
(b) Google Chrome extension

To test a RESTful web service, you have to write REST client programmatically and consume the service. Another way is to use a REST client. There are many standalone REST clients available in the market. I generally use a Google Chrome plugin REST client. “Advance Rest Client” is a Google Chrome extension and it is a good REST client. You need to add the extension and use it as below. Following are the screen shots of testing the above example RESTful web service.

RESTful Web Service HTML Output

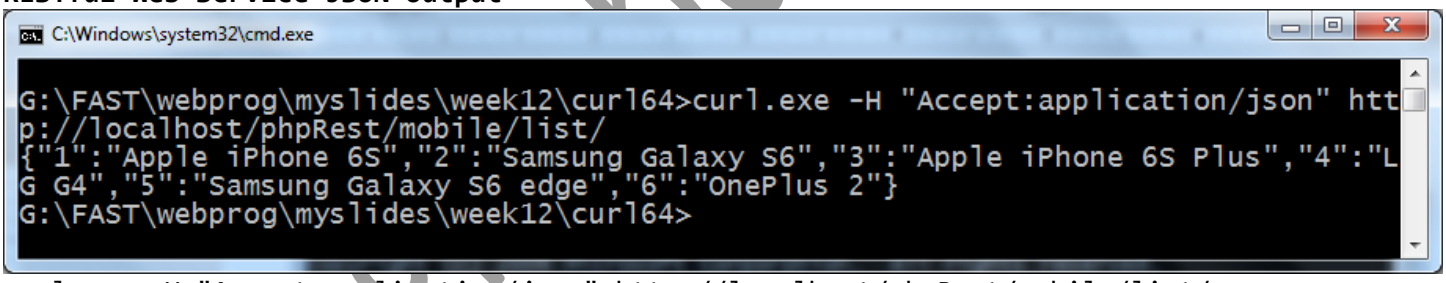
localhost/phpRest/mobile/list/

1	Apple iPhone 6S
2	Samsung Galaxy S6
3	Apple iPhone 6S Plus
4	LG G4
5	Samsung Galaxy S6 edge
6	OnePlus 2



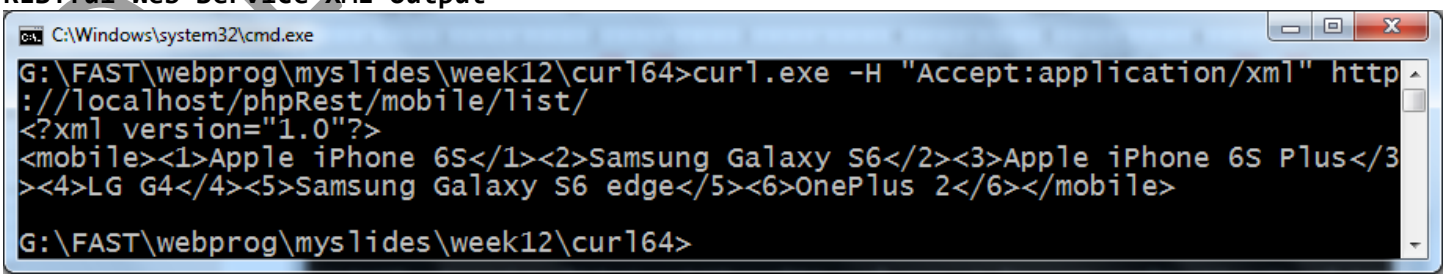

```
curl.exe -H "Accept:text/html" http://localhost/phpRest/mobile/list/
<table border='1'><tr><td>1</td><td>Apple iPhone 6S</td></tr><tr><td>2</td><td>S
amsung Galaxy S6</td></tr><tr><td>3</td><td>Apple iPhone 6S Plus</td></tr><tr><t
d>4</td><td>LG G4</td></tr><tr><td>5</td><td>Samsung Galaxy S6 edge</td></tr><tr
><td>6</td><td>OnePlus 2</td></tr></table>
G:\curl64>
```

RESTful Web Service JSON Output




```
curl.exe -H "Accept:application/json" http://localhost/phpRest/mobile/list/
{"1":"Apple iPhone 6S","2":"Samsung Galaxy S6","3":"Apple iPhone 6S Plus","4":"L
G G4","5":"Samsung Galaxy S6 edge","6":"OnePlus 2"}
G:\curl64>
```

RESTful Web Service XML Output




```
curl.exe -H "Accept:application/xml" http://localhost/phpRest/mobile/list/
<?xml version="1.0"?>
<mobile><1>Apple iPhone 6S</1><2>Samsung Galaxy S6</2><3>Apple iPhone 6S Plus</3
><4>LG G4</4><5>Samsung Galaxy S6 edge</5><6>OnePlus 2</6></mobile>
G:\FAST\webprog\myslides\week12\curl64>
```