Best Practices for developing REST API using PHP

aspire SYSTEMS attention. always.

Web services are a common way to enable distribution of data. They can be used to allow different software components interact with one another. It can be built using many ways and one among them is REST. REST language is independent and can be fit into Perl, Java and PHP. This whitepaper covers the best practices in developing REST API using PHP.

A Quick Overview of REST

Representational State Transfer (REST) provides a lighter weight alternative to mechanisms like RPC (Remote Procedure Calls) and SOAP, WSDL, etc. The most common use of REST is on the Web with HTTP being the only protocol used here. In REST each service is viewed as resource identified by a URL, and the most common operations allowed are the HTTP – GET (Read), PUT (Update), POST (Create) and DELETE (Delete). It uses HTTP for all four CRUD operations. It relies on a stateless, client-server and cacheable communications protocol. The possible server response will be in JSON, XML, HTML and CSV format. With REST, a simple network connection is all you need. You can even test the API directly, using your browser. REST can be carried over secure sockets, and content can be encrypted using any mechanism.

REST in PHP:

Authentication:

HTTP basic authentication

The simplest way to deal with authentication is to use HTTP basic authentication. It use a special HTTP header where we can add 'username: password' encoded in base64.
//Encode username & password and build Authorization $auth_token = base64_encode($username. '-' . $password);
$headers = array ('Authorization=Basic ' . $auth_token, 'User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.3) Gecko/20070309 Firefox/2.0.0.3');
Even though credentials are encoded, it is very easy to retrieve the username and password from a basic authentication. We should not use this authentication scheme on plain HTTP, but can be used only through SSL/TLS.

HMAC

Another way of Authentication is to use HMAC (hash based message authentication). Instead of having passwords that needs to be send over, we actually send a hashed version of the password. HMAC is a hash function applied to the body of a message along with a secret key. Instead of sending the username and password with a Web service request, we can send some identifier for the private key and an HMAC. When the server receives the request, it looks up the user’s private key and uses it to create an HMAC for the incoming request.

The two big advantages here are, it allows you to verify the password without requiring the user to embed it in the request, and the second is it verifies the basic integrity of the request. If an attacker manipulated the request in any way in transit, the signatures would not match and the request would not be authenticated.

PHP has a built-in HMAC function: hash_hmac which generate a keyed hash value using the HMAC method.

```
hash_hmac (string $algo, string $data, string $key)
```

First parameter is the name of selected hashing algorithm (i.e. "md5",”Shal”) the second one is message to be hashed and the last parameter is shared secret key used for generating the HMAC.

```
echo hash_hmac('sha1', "Message", "Secret Key");
Output will be: b8e7ae12510bdfb1812e463a7f086122cf37e4f7
```

OAuth

One of the downsides of HMAC is using passwords. OAuth allows applications to securely access data using tokens based authentication. First we need to register with provider to get the consumer key and consumer secret using that we can request for tokens and make an API request. In PHP we have in-build OAuth class like getAccessToken (fetch an access token), getRequestToken (fetch a request token), generateSignature …etc

//Create OAuth client Object & Fetch the request Tokens

```
$o = new oAuth(MY_CONSUMER_KEY,MY_CONSUMER_SECRET);
$response=$o->getRequestTokens(‘http://photos.example.net/oauth/request_token’);
```

We can save the request token and secret in session for later exchange. Next we need to get the access tokens.

// Sign requests with the request token & Exchange request for access token

```
$o->setToken($_SESSION['request_token'], $_SESSION['request_secret']);
$response=$o->getAccessToken('https://photos.example.net/oauth/access_token' );
```

//Simple API request to get the image

Note: Create object and get request/access tokens.

```
$oauth->fetch("http://photos.example.net/photo?file=vacation.jpg");
$response_info = $oauth->getLastResponseInfo();
header("Content-Type: {$response_info["content_type"]}");
echo $oauth->getLastResponse();
```

OAuth Security:

We can secure our communication specifically making non-HTTPS by generating signature, nonce and timestamp.

Generating Signatures: It protects the content of the request from changing and ensure that requests can only be made by an authorized Consumer.

```
OAuth::generateSignature ( string $http_method , string $url , $extra_parameters )
```

Nonce (number used once): It identifies each unique signed request, and prevent requests from being used more than once. This nonce value is included in the signature, so it cannot be changed by an attacker.

```
OAuth::setNonce ( string $nonce )
```

Timestamp: We can add a timestamp value to each request .When a request comes with an old timestamp, the request will be rejected.

```
OAuth::setTimestamp ( string $timestamp )
```

From a security standpoint, the combination of the timestamp value and nonce string provide a perpetual unique value that cannot be used by an attacker.

HTTP Methods

The most-commonly-used HTTP methods are POST, GET, PUT and DELETE. The other methods OPTIONS, HEAD, PATCHES and TRACE which are utilized less frequently.

GET: This method is used for getting data from the server. The data may be anything a HTML document, an image, XML file etc.

// List documents in a collection

```
$responce=example_rest("GET"," www.example.com/api/document","collection=test");
```

POST: This method is used to create a new resource or to update an existing resource in the server.

// Create a document

```
$responce = example_rest("POST","www.example.com/api/document ","collection=test",'{ "hello":"World" }');
```

DELETE: This is pretty easy to understand. It is used to delete a resource identified by a URL.

// Delete collection

```
$response=example_rest("DELETE"," www.example.com/api/collection/test");
```

PATCHES: This method can be used to update partial resources. For instance, when you only need to update one field of the resource, we can use this utilizing less bandwidth.

// Change document

```
$document_handle= $responce['documents'][0];
$responce=example_rest("PATCH",$document_handle,"",'{ "hello" : "World of mine" }');
```

HEAD: This method is identical to GET except that the server does not return a message-body in the response. This method can be used for obtaining meta information about the entity implied by the request without transferring the entity-body itself.

TRACE: This method simply echoes back to the client whatever string has been sent to the server, and is used mainly for debugging purposes

OPTIONS: This method allows determining the options and requirements associated with a resource. Responses to this method are not cacheable. If the OPTIONS request includes an entity-body (as indicated by the presence of Content-Length), then the media type MUST be indicated by a Content-Type field.

PUT: PUT is often utilized for update capabilities. Use PUT whenever you need to update a resource completely through a specific resource. For instance, if you know that an article resides at http://example.org/article/1234, you can PUT a new resource representation of this article directly through a PUT on this URL. PHP doesn't have $_PUT like it does in $_POST, Here's how to access the content of a PUT request in PHP:

```
$_PUT = array();
if($_SERVER['REQUEST_METHOD'] == 'PUT') {   parse_str(file_get_contents('php://input'), $_PUT);  }
```

Safe and idempotent methods

Safe methods that do not modify resources. For instance, using GET or HEAD on a resource url, should never change the resource.
Idempotent methods that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same; The methods GET, HEAD, PUT and DELETE are idempotent methods.

Making REST Requests with PHP

PHP, being an extremely flexible and feature-rich language, provides a number of different mechanisms you can choose to make your REST requests. The most commonly used mechanism is file_get_contents and Curl.

Requests using file_get_contents:

This mechanism is the simplest method and is especially good for very short scripts .It fetch a URL and returns the contents as a string. For HTTP GETs, REST requests take the form of a URL, so any parameters that might contain spaces, or other characters that are illegal in a URL, must be encoded using PHP's urlencode function.

```
//GET Request
$request =  'http://search.yahooapis.com/ImageSearchService/V1/imageSearch?appid=YahooDemo&query='.urlencode('Al Gore').'&results=8';
$response = file_get_contents($request);
```

The limitation with this mechanism is, it's only available in PHP 4.3 and later versions. And in PHP installation it must have fopen wrappers enabled in order to access URLs using file_get_contents and its main drawbacks is does not let you do HTTP POST calls.In this case we can use CURL which supports for POST, GET, PUT, DELETE methods .The limitation with this mechanism is, it's only available in PHP 4.3 and later versions. And in PHP

Requests using CURL:

CURL is a PHP extension that must be loaded in your PHP configuration and it offers a lot of control over network transport options and is especially convenient if you are making secure HTTPS requests.

```
//GET Request
$request = http://example.com/api/conversations
$session = curl_init($request);
```

Then we can set any CURL options that we might needed for our application.

```
// Tell curl not to return headers, but do return the response
curl_setopt($session, CURLOPT_HEADER, false);
curl_setopt($session, CURLOPT_RETURNTRANSFER, true);
```

The response is returned by running curl_exec on the session handle. After executing the session, we close it with curl_close:

```
$response = curl_exec($session);
curl_close($session);
```

For POST calls slightly different from GET. We separate the request URL and parameters into separate variables and we need to specify various curl options .

```
// Tell curl to use HTTP POST
curl_setopt ($session, CURLOPT_POST, true);

// Tell curl that this is the body of the POST
curl_setopt ($session, CURLOPT_POSTFIELDS, $postargs);
```

Error Handling in REST with PHP:

There are many ways in which we can tackle error handling. Most REST services will send some kind of error condition structure which embeds an error message describing the error and some kind of code. The most common way to handle errors in REST is to use HTTP status codes. There are over 70 HTTP status codes available; the most common codes to use for any HTTP request are 200 – OK, 404 – Not found and 500 – Internal Server Error. In PHP HTTP status code are returned in the HTTP response header.

In file_get _contents mechanism the HTTP status code can be retrieved from the PHP global array $http_response_header. The array contains one HTTP header per array element and is set automatically after execute file_get_contents.

```
list($version, $status_code, $msg) = explode(' ', $http_response_header[0], 3);
```

In CURL HTTP status code can be accessed via the HTTP header, we can set the curl option for the header to true and we can extract the status code from the header.

```
curl_setopt($session, CURLOPT_HEADER, true);
```

As a best practice we need to set the HTTP status code based on the request result. For example, if a request has no content the appropriate “204 – No Content” HTTP status is sent. Or if a new record is being created the appropriate “201- Created” HTTP status is sent.

Caching

Caching is essential to decrease network loads and improve overall reliability of your Web applications. Let’s see different way of caching REST request .

Cache request in a file:

We can cache the web service requests in local files.

```
$response = request_cache($request, $cache_fullpath, $cache_timeout);
```

Here the three parameters are the actual request URL, the path to the cached version of the request, and the number of seconds before the cache becomes stale. The function request_cache checks to see if the request has already been cached or if it has become stale. If the request has not been cached or it is stale, then we make the request and cache it.

Caching Requests with APC:

The Alternative PHP Cache (APC) is a fast caching mechanism available as a PHP extension.

```
$response = request_cache($request,  $cache_timeout);
```

The two parameters are the actual request URL and the number of seconds before the cache becomes stale. The URL is used as the key to retrieve the cached request which makes APC a very simple cache. In PHP we have in-build APC functions like apc_add (cache new variable in the data store), apc_cache_info (Retrieves cached information from APC's data store) , apc_fetch (Fetch a stored variable from the cache) ..etc

HTTP Caching:

HTTP Caching done by setting HTTP cache headers on the response. In PHP we can set HTTP headers by using the Header() function. The most important and versatile header is the Cache-Control header, which is actually a collection of various cache information.

// Cache-Control header, and an Expires header three days in the future

```
Header("Cache-Control: must-revalidate");
$offset = 60 * 60 * 24 * 3;
ExpStr = "Expires: “. gmdate("D, d M Y H:i:s", time() + $offset) . “GMT";
Header($ExpStr);
```

Setting ETags:

The Etag header is often used to check if the api response itself has changed.When you make a rest API call, the response header includes an ETag with a value which is the hash of the data returned in the API call.

//Sample GET response

```
HTTP/1.1 200 Ok
Date: Wed, 21 Mar 2007 15:06:15 GMT
Server: Apache
ETag: "7776cdb01f44354af8bfa4db0c56eebcb1378975"
Content-Length: 23081
Vary: Accept-Encoding,User-Agent
Connection: close
Content-Type: application/xhtml+xml; charset=utf-8
```

Again make the same API call, include the If-None-Match request header with the ETag value .If the response data has not changed, the response status code will be 304 – Not Modified and no data is returned. If the data has changed since the last query, the data is returned as usual with a new ETag. Save the new ETag value and use it for subsequent calls.
Since REST is an HTTP thing, the best way of caching requests is to use HTTP caching.

Avoid Common REST Mistakes

Ignoring response codes: HTTP protocol has an extensive set of application-level response codes that you can use to describe various results of your API calls. If your API only returns 200 (OK) or 500 (Internal Server Error), or even 200 with an error message encoded in the response body, your application's loose coupling, reusability and interoperation will be difficult to achieve in changing environments.

URL Structure: In REST design the URL endpoints should be well formed and should be easily understandable. Every URL for a resource should be uniquely identified. If your API needs an API key to access, the api key should be kept in HTTP headers instead of including it in URL.

GET http://abc.com/v1/tasks/11 – Will give the details of a task whose id is 11

Ignoring MIME types: If resources returned by API calls only have a single representation, you are probably only able to serve a limited number of clients that can understand the representation. If you want to increase a number of clients that can potentially use your API, you should use HTTP's content negotiation. It allows you to specify standard media types for representations of your resource such as XML, JSON or YAML which in turn increase chances that even unforeseen clients will be able to use your API.

API Key: If you are building a private API where you want to restrict the access or limit to a private access, the best approach is to secure your API using an API key. The user is identified by the api key and all the actions can be performed only on the resources belongs to him. The API key should be kept in request header Authorization filed instead of passing via url.

Ignoring hypermedia: If your API calls send representations that do not contain any links, you are most likely breaking the REST principle called Hypermedia as the Engine of Application State (HATEOAS). Hypermedia is the concept of linking resources together allowing applications to move from one state to another by following links. If you ignore hypermedia, it is likely that URIs must be created at the client-side by using some hard-coded knowledge.

API Versioning: There is always a discussion happens whether API versioning to be maintained in the URL or in the HTTP request headers. It is recommended that version should be included in the request headers.

About Aspire Systems

At Aspire we follow similar best practices in developing REST API using PHP in our development projects. Aspire follows an approach called "ProducteeringTM" to build products better, faster and innovate continuously. This approach has a set of principles and practices, driven by the right people and supported by the right platforms. Aspire's proven engineering practices acts as the base for REST API development. Thus, through the best engineering practices and technology innovation which are key elements of Producteering, the elements of REST API are developed using PHP based on the requirements.

Aspire Systems is a global technology services firm serving as a trusted technology partner for our customers. We work with some of the world's most innovative enterprises and independent software vendors, helping them leverage technology and outsourcing in our specific areas of expertise. Our services include Product Engineering, Enterprise Transformation, Independent Testing Services and IT Infrastructure Support services.

Find Us

Slideshare LinkedIn Twitter

USA
Aspire Systems, Inc.
1735 Technology Drive
Suite 260, San Jose, CA - 95110, USA
Tel: +1-408-260-2076, +1-408-260-2090
Fax: +1-408-904-4591
E-mail: info@aspiresys.com

INDIA
Aspire Systems (India) Pvt. Ltd.
1/D-1, SIPCOT IT PARK
Siruseri, Chennai - 603 103
Tamil Nadu, India
Tel: +91-44-6740 4000
Fax: +91-44-6740 4234
E-mail: info-india@aspiresys.com

UK
Aspire Systems
1, Lyric Square, Hammersmith
London - W6 0NB, UK
Tel: +44 203 170 6115
E-mail: info@aspiresys.com

UAE
Aspire Systems (FZE)
Executive Suite X2-20
P.O.Box: 120725
Sharjah, UAE
Fax: +91-44-6740 4234
E-mail: info@aspiresys.com