Web Services and PHP

Contents

- Create WS by PHP
- Calling RESTful WS using PHP curl
- Map nice URL to internal file

Key REST principles

- Give every “thing” an ID
- Link things together
- Use standard methods
- Resources with multiple representations
- Communicate statelessly

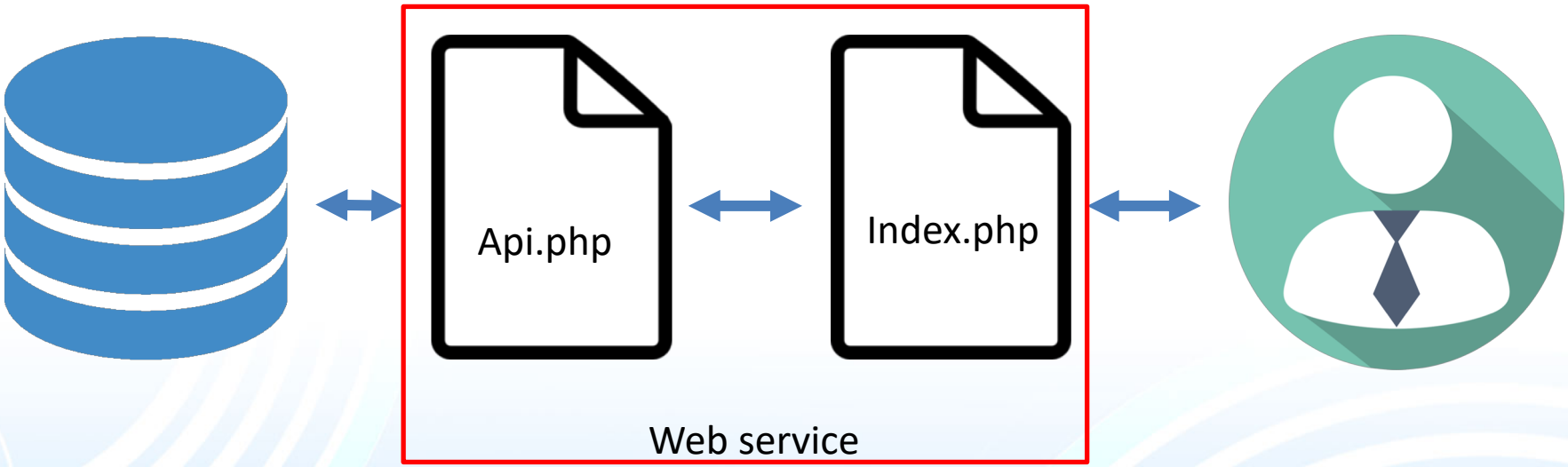
REST: Technologies

- HTTP as the basis,
- PHP to add dynamic behaviour to HTML pages,
- XML and JSON for data exchange, and
- AJAX and jQuery for client-side programming (e.g. browser).

Creating REST web service using PHP and MySQL

- Create a database
- Create a table
- Create a class *api* to interact with the database
- Create a php file to interact with the class *api* and return JSON
- Create HTML to interact with the service

Overview Architecture



Configuration

```
{
 "host":"localhost",
 "user":"root",
 "password":"root",
 "database":"testrest"
}
```

Api.php

- There are three function in this file.
  - Construct
  - Destruct
  - Get

Api.php: Construct

```
function __construct()
{
$conf = json_decode(file_get_contents('configuration.json'), TRUE);
$this->db = new mysqli($conf["host"], $conf["user"], $conf["password"],
$conf["database"]);
}
```

Api.php: Destruct

```
function __destruct()
{
$this->db->close();
}
```

Api.php: Get

```
function get($params)
{
$query = 'SELECT u.id AS id'
. ', u.username AS username'
. ', u.realname AS realname'
. ', u.password AS password'
. ', u.email AS email'
. ' FROM user AS u'
. ' WHERE u.id = ' . $params['id']
. ' ORDER BY u.username'
;
$list = array();
$result = $this->db->query($query);
while ($row = $result->fetch_assoc())
{
$list[] = $row;
}
return $list;
}
```

Api.php

- For the api.php, it is used to connect with the database in order to get the data.

Index.php

```php
require_once 'api.php';

$api = new api();
```

```php
$message = array();

switch($_GET["action"])
{
case 'get':
$params = array();
$params['id'] = isset($_GET["id"]) ? $_GET["id"] : '';
if (is_array($data = $api->get($params))) {
$message["data"] = $data;
} else {
$message["message"] = "Error on get method";
}
break;

default:
$message["message"] = "Unknown method " .
$_GET["action"];
break;
}
```

Index.php: Sending JSON

```
header('Content-type: application/json; charset=utf-8');
echo $_GET["callback"]."(".json_encode($message).")";
```

Cross-domain issues

- **Cross-origin resource sharing (CORS)** is a mechanism that allows restricted resources (e.g. fonts) on a web page to be requested from another domain outside the domain from which the first resource was served.
- However, some resources are not allowed for exchanging cross domains such as Ajax requests
- HTTP POST is allowed.
- (HTTP GET) It is possible to make cross-origin requests either using **JSONP** (if you trust the server!) or using a **CORS** request (Cross-Origin Resource Sharing), which both client and server must agree to (I can supply more details if you need it on this).

JSONP

- **JSONP (JSON with padding)** is used to request data from a server residing in a different domain than the client. It was proposed by Bob Ippolito in 2005.
- JSONP enables sharing of data bypassing same-origin policy.

Index.html

- This file is used to interact with the service.

```
<table>
   <tr>
      <td>Search Person: </td>
      <td><input type="text" id="q" size="10"/></td>
   </tr>
</table>
<input type="button" value="Search" id="getResult"/><br/>
<label id="urlString"></label>
<p id="output"></p>
```

Ws.js

```
var searchURL = "http://localhost/webservice-php/index.php?callback=?"

$(document).ready(function() {


        $("#getResult").click(function(){

            var query = $("#q").val();
            $.getJSON(searchURL, {id: query, action: "get"}, function(jd){
                $("#output").html('<p>' + JSON.stringify(jd) + '</p>');
                console.log(jd.data);
            });
        });
    });
```

Allow Cross-Origin Resource Sharing

- Add `header('access-control-allow-origin: *');`

Modify Index.php: Sending JSON

```
header('Content-type: application/json; charset=utf-8');
header('access-control-allow-origin: *');
echo json_encode($message);
```

Ws.js

```
var searchURL = "http://localhost/webservice-php/index.php"

$(document).ready(function() {


        $("#getResult").click(function(){

            var query = $("#q").val();
            $.getJSON(searchURL, {id: query, action: "get"}, function(jd){
                $("#output").html('<p>' + JSON.stringify(jd) + '</p>');
                console.log(jd.data);
            });
        });
    });
```

Consuming Rest From PHP With Curl

- libcurl is a free and easy-to-use client-side URL transfer library, supporting FTP, FTPS, TFTP, HTTP, HTTPS, TELNET, DICT, FILE and LDAP.
- Installing the PHP/CURL binding
  - Activation of PHP/CURL: removing a semicolon from the following line in **php.ini:**
    ;extension=php_curl.dll
  - For xampp, php.ini is located in xamppfiles/etc

How to use the CURL functions ?

- 1. initialize a CURL session using the
  - **curl_init()**
- 2. set all options for the transfer via the
  - **curl_setopt()**
  - **curl_setopt_array()**
- 3. execute the session with the
  - **curl_exec()**
- 4. finish off your session using the
  - **curl_close()**

curl_setopt()

- **curl_setopt (resource $ch , int $option , mixed $value)**
  - ch: A cURL handle returned by curl_init().
  - option: The CURLOPT_XXX option to set.
  - value: The value to be set on option.
- For example,

```
curl_setopt ($curl, CURLOPT_URL, ”http://www.example.com”)
```

Curl_setopt_array()

- **`curl_setopt_array (resource $ch, array $options)`**
  - ch: A cURL handle returned by curl_init().
  - options: An array specifying which options to set and their values.
- For example,

```
curl_setopt_array($ch, array(
    CURLOPT_URL =>
'http://www.example.com/',
    CURLOPT_HEADER => false
    )
);
```

Important curl options to call web service

- CURLOPT_URL
  - Set an URL to work on
- CURLOPT_HTTPAUTH
  - Set a HTTP server authentication methods.
- CURLOPT_SSL_VERIFYPEER
  - Verify the peer's SSL certificate
- CURLOPT_RETURNTRANSFER
  - Return the response as a string of the return value of curl_exec()

Example of calling YouTube API

- See example
- Needed variables
  - $APi_key: API key from google
  - $query: keyword that is used to search
  - $params
  - $searchURL

Example of calling YouTube API: $params

```
$params = array(
    'part' => 'snippet',
    'maxResults' => '25',
    'q' => $query,
    'key' => $APi_key
);
```

Example of calling YouTube API: $searchURL

```
$searchURL =
"https://www.googleapis.com/youtube/v3/search?".http_build_q
uery($params);
```

- http_build_query()
  - Generate URL-encoded query string

Example of calling YouTube API: Calling Service

```
// Get cURL resource
$curl = curl_init();

curl_setopt_array($curl, array(
    CURLOPT_URL => $searchURL,
    CURLOPT_HTTPAUTH => CURLAUTH_ANY,
    CURLOPT_SSL_VERIFYPEER => true,
    CURLOPT_RETURNTRANSFER => true
)
);


// Send the request & save response to $resp
$resp = curl_exec($curl);

// Close request to clear up some resources
curl_close($curl);

header('Content-type: application/json; charset=utf-8');
echo $resp;
```

Example of calling your own web service

```
$params = array(
    'id' => '1',
    'action' => 'get'
);

$searchURL = "http://localhost/webservice-
php/index.php?".http_build_query($params);

$curl = curl_init();

curl_setopt_array($curl, array(
    CURLOPT_URL => $searchURL,
    CURLOPT_HTTPAUTH => CURLAUTH_ANY,
    CURLOPT_SSL_VERIFYPEER => true,
    CURLOPT_RETURNTRANSFER => true
));

$resp = curl_exec($curl);
curl_close($curl);

header('Content-type: application/json; charset=utf-8');
echo $resp;
```

Convert JSON response to PHP object

- `json_decode($json)`
  - Takes a JSON encoded string and converts it into a PHP variable.
- `json_encode($value)`
  - Returns the JSON representation of a value
- See example

```
//Convert JSON to object in PHP
$obj = json_decode($resp);
print_r($obj->{'items'}[0]->{'id'}->{'videoId'});
```

Map nice URL to internal file

- Advantages
  - Search-Engine Friendly
  - user-friendly

Normal URL: http://localhost/products/productsearch.php?id=123

Nice URL: http://localhost/products/list/123

Steps to Map nice URL to internal file

- Create a file named ".htaccess" in the file of your PHP files
  - Note that it is a hidden file. Therefore, you need to allow showing hidden file on your system
- In the file, you need to add the following in order to turn on rewriting function

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

Steps to Map nice URL to internal file (cont.)

- RewriteRule

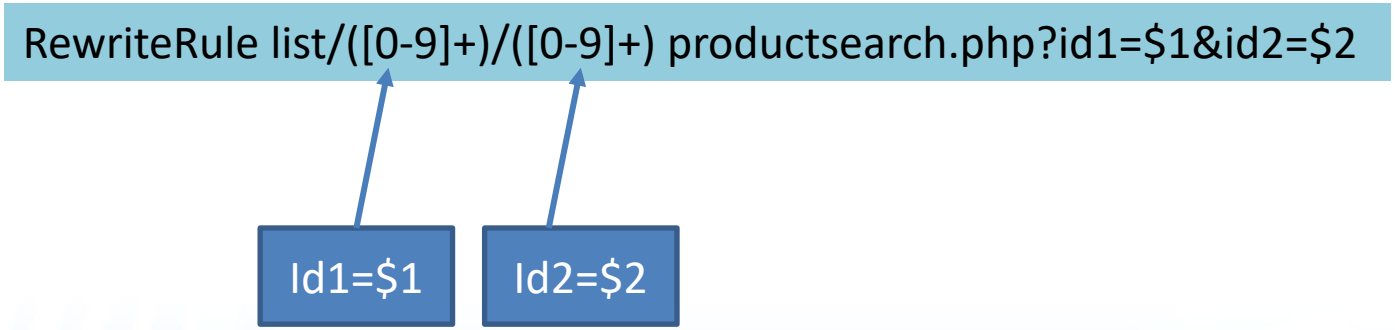
```
RewriteRule Pattern Substitution [Flag(s)]
```

- Pattern is a regular expression. E.g., [0-9]+ and [ad]*
- Substitution is a path to internal file
- For example,

```
RewriteRule list/([0-9]+) productsearch.php?id=$1
```

Steps to Map nice URL to internal file (cont.)

- Multiple parameters



Steps to Map nice URL to internal file (cont.)

- Flags are used to modify a behaviour of RewriteRule

```
RewriteRule Pattern Substitution [Flag(s)]
```

- For example,
  - NC (nocase): Use of the [NC] flag causes the RewriteRule to be matched in a case-insensitive manner.
  - QSA (qsappend): When the replacement URI contains a query string, the default behavior of RewriteRule is to discard the existing query string, and replace it with the newly generated one. Using the [QSA] flag causes the query strings to be combined. E.g.

QSA flag

- For example,

```
RewriteRule "/products/(.+)" "/products.php?id=$1" [QSA]
```

- With the [QSA] flag, a request for **/products/123?one=two** will be mapped to **/products.php?id=123&one=two**.
- Without the [QSA] flag, that same request will be mapped to**/products.php?id=123**, the existing query string will be **discarded**.

More flags

- https://httpd.apache.org/docs/2.4/rewrite/flags.html

Steps to Map nice URL to internal file (cont.)

- Note that you need to restart the web server when you modify the file “.htaccess”.

References

- https://httpd.apache.org/docs/2.4/rewrite/
- http://php.net/manual/en/book.curl.php