RESTful webservices in PHP

Key concepts

RESTful webservices, JSON, GET, PUT, POST, PATCH, DELETE, stateless, safe, idempotent, .htaccess, mod_rewrite

Alternatieve bronnen

http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069?pgno=1
https://www.ibm.com/developerworks/library/ws-restful/
http://www.restapitutorial.com/
http://www.restapitutorial.com/lessons/httpmethods.html
http://www.restapitutorial.com/httpstatuscodes.html

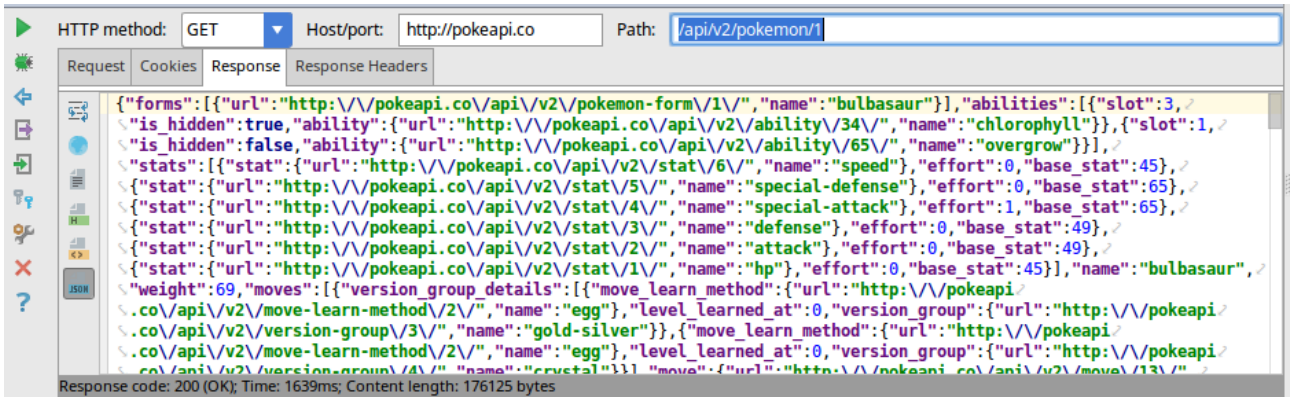
1. RESTFul webservices inleiding

Webservices zijn server-sided toepassingen die aangesproken worden door clients via het HTTP-protocol.

Bijvoorbeeld via het GET HTTP commondo kan de URI http://pokeapi.co/api/v2/pokemon/1/ bevraagd worden. Het resultaat is een JSON-string:

```
{
    "id": 1,
    "name": "bulbasaur",
    "base_experience": 64,
    "height": 7,
    "is_default": true,
    "order": 1,
    "weight": 69,
    "abilities": [
        {
            "is_hidden": true,
            "slot": 3,
            "ability": {
                "name": "chlorophyll",
                "url": "http://pokeapi.co/api/v2/ability/34/"
            }
        }
    ],
    ...
}
```

Je kan deze API uittesten via PHPStorm: Tools > Test RESTful webservice[1]



Restful webservices maken gebruik van het HTTP-protocol, zijn staless, maken gebruik van URI's die lijken op mappenstructuren en versturen data in XML- of JSON-formaat.

De volgende HTTP-commando's kunnen gebruikt worden in een webservice:

GET:	haal een resource op
POST:	maak een nieuwe resource (URI niet gekend)
PUT:	maak of wijzig een resource (URI gekend)
PATCH:	doe een gedeeltelijke update (URI gekend)
DELETE:	verwijder een resource

Elk request moet stateless zijn: het request mag nooit informatie bevatten of veronderstellen over een vorig request. Het is wel toegelaten om bijvoorbeeld de gebruiker met id 3 op te vragen, het is niet toegestaan om de volgende gebruiker (tov het vorige request) te vragen.

Er wordt gewerkt met URI's die lijken op mappenstructuren:

bijvoorbeeld	GET /users/	haalt alle gebruikers op
	GET /users/1	haalt de gebruiker met id 1 op
	GET /writers/1/books/	haal van writer 1 alle boeken op
	GET /writers/1/books/2	haal van writer 1 boek 2 op
	DELETE /writer/1/book/	verwijder alle boeken van writer 1

Enkel GET requests zijn safe: de toestand op de server verandert niet door een GET-request.

GET, PUT en DELETE requests zijn idempotent. Dit wil zeggen meerdere keren hetzelfde request uitvoeren geeft hetzelfde resultaat als één keer het request uitvoeren. POST en PATCH requests zijn niet idempotent.

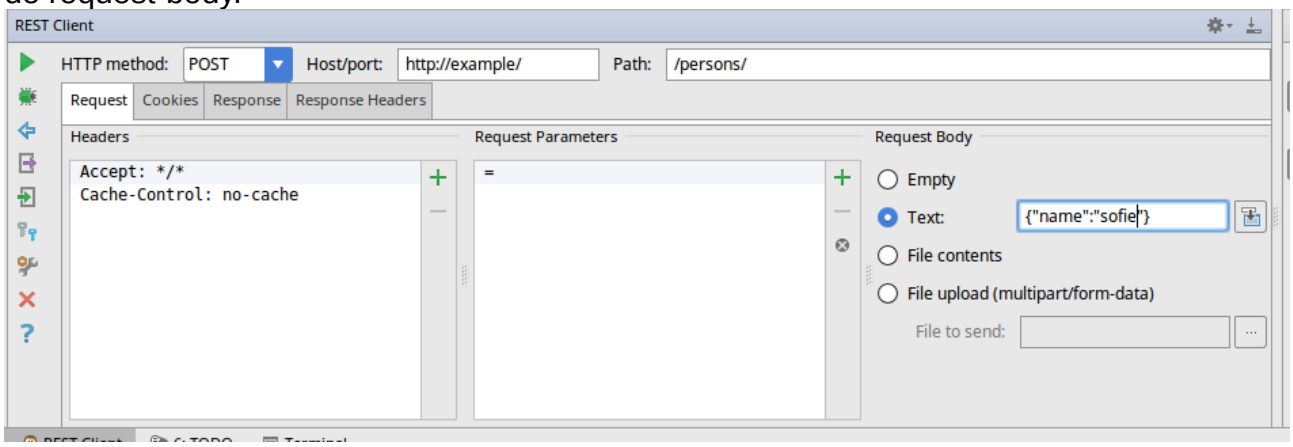
Bijvoorbeeld het request PUT /persons/123 met in de request-body de tekst {"name":"sofie"} een eerste keer uitvoeren maak de resource met id 123 aan als deze niet bestaat. De waarde id = 123 en name = sofie wordt een eerste keer weggeschreven in een databank. De volgende keren dat dit request uitgevoerd wordt verandert de toestand

1Als alternatief kan je ook POSTMAN gebruiken https://www.getpostman.com/ .

niet: er blijft een waarde met id 123 en name sofie in de databank staan.

Het request POST /persons/ met in de request-body de tekst {"name":"sofie"} maakt telkens een nieuwe resource op de server aan. Telkens dit request uitgevoerd wordt, wordt name jan toegevoegd in de databank op de eerstvolgende niet-gebruikte id.

In onderstaande screenshot wordt getoond hoe je de tekst {“name”:”sofie”} kan plaatsen in de request-body.



Afhankelijk van het al dan niet slagen van de request wordt een HTTP-response code meegegeven met de response. In onderstaande tabel worden enkele voorbeelden van requests en responses gegeven

	URI	request-body	response-code	response-body
GET	/api/persons/		200 (OK)	[{“id”:1,”name”:”sofie”}, {“id”:2,”name”:”tim”}]
GET	/api/persons/1		200 (OK)	{“id”:1,”name”:”sofie”}
GET	/api/persons/7		404 (not found)	
GET	/api/persons/a		400 (bad request)	
PUT	/api/persons/1	{”name”:”guy”}	200 (OK)	{“id”:1,”name”:”guy”}
PUT	/api/persons/3	{”name”:”kurt”}	201 (resource created)	{“id”:3,”name”:”kurt”}
PUT	/api/persons/1	{niet valid	400 (bad request)	
POST	/api/persons/	{”name”:”gert”}	201 (resource created)	{“id”:4,”name”:”gert”}

POST	/api/persons/	{niet valid	400 (bad request)	
DELETE	/api/persons/1		200 (OK)	
DELETE	/api/persons/7		404 (not found)	
DELETE	/api/persons/a		400 (bad request)	

2. Consumptie van Restful webservices in Javascript (JQuery)

Restful webservices worden dikwijls geconsummeerd vanuit Javascript. Aan de hand van de JSON-string die de webservice teruggeeft wordt de html-pagina aangepast. In onderstaand voorbeeld wordt de webservice van jsonplaceholder gebruikt om de eerste post op te halen (http://jsonplaceholder.typicode.com/posts/1)

De JSON-string is van de vorm

```
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere ...  ",
  "body": "quia et suscipit\nsuscipit ..."
}
```

In het voorbeeld wordt de post met id 1 uitgelezen en wordt de body van deze post in een div geplaatst.

example1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Example</title>
    <meta charset="utf-8">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"><
/script>
</head>
<body>
<div id="result">ok</div>
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax('http://jsonplaceholder.typicode.com/posts/1', {
            method: 'GET'
        }).then(function(data) {
            $("#result").text(data.body);
        });
    });
</script>
</body>
</html>
```

Alle posts uitlezen kan via onderstaande code.

```
    $(document).ready(function () {
        $.ajax('http://jsonplaceholder.typicode.com/posts/', {
            method: 'GET'
        }).then(function(data) {
            var tableNode=$("#result");
            for (var i =0; i<data.length;i++){
                var trNode=$("<tr>");
                var thNodeId=$("<th>");
                thNodeId.text(data[i].id);
                trNode.append(thNodeId);
                var tdNodeBody=$("<td>");
                tdNodeBody.text(data[i].body);
                trNode.append(tdNodeBody);
                tableNode.append(trNode);
            }
        });
    });
```

Een post aanmaken kan via method ‘POST’: het id van de post is nog niet gekend en er wordt een nieuwe id voor de post voorzien.

```
$.ajax('http://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  data: {
    title: 'foo',
    body: 'bar',
    userId: 1
  }
}).then(function(data) {
  console.log(data);
});

/* will return
{
  id: 101,
  title: 'foo',
  body: 'bar',
  userId: 1
}
*/
```

Deze en bijkomende voorbeelden kan je terugvinden op de onderstaande url:
https://github.com/typicode/jsonplaceholder#how-to

3. Een voorbeeld in PHP gebruik makend van AltoRouter

Een router ontleedt een request naar de server en laat het request overeenkomen met een stuk code dat uitgevoerd moet worden.

Via composer kan je altorouter installeren:

composer require altorouter/altorouter 1.1

app.php

```
...
try {
    ...
    $router = new AltoRouter();
    $router->setBasePath('/');
    $router->map(
        'GET',
        'persons/[i:id]',
        function ($id) use ($personController) {
            $personController->handleFindPersonById($id);
        }
    );
    $router->map(
        'GET',
        'persons/',
        function () use ($personController) {
            $personController->handleFindPersons();
        }
    );
    $router->map(
        'POST',
        'persons/',
        function () {
            $requestBody = file_get_contents('php://input');
            $jsonObject=json_decode($requestBody);
        // ...
        }
    );
    $match = $router->match();
    if ($match && is_callable($match['target'])) {
        call_user_func_array($match['target'], $match['params']);
    } else {
        http_response_code(500);
    }
} catch (Exception $exception) {
    http_response_code(500);
}
```

De code

```
$router->map(
    'GET',
    'persons/[i:id]',
    function ($id) use ($personController) {
        $personController->handleFindPersonById($id);
    }
);
```

Zorgt ervoor dat GET requests naar een /persons/ gevolgd door de integer id gemapt worden naar een anonieme functie[2]. Deze functie heeft ook toegang tot de variabele $personController. De functie wordt ook de target van de route genoemd, id is de parameter (params) van de route.

Wanneer de route gematched wordt, wordt de target functie uitgevoerd[3]:

```
    $match = $router->match();

    if( $match && is_callable( $match['target'] ) ){
        call_user_func_array( $match['target'], $match['params'] );
    }
```

Het bestand .htaccess zorgt ervoor dat elke request naar een url op de server herschreven wordt naar het bestand api.php. Het . op regel 3 in dit bestand is deel van een reguliere expressie, een url die eender welk symbool bevat wordt herschreven naar api.php.

.htaccess

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule . app.php [L]
```

Probeer de onderstaande url's uit:

http://192.168.33.22/persons/1
http://192.168.33.22/persons/

2http://php.net/manual/en/functions.anonymous.php
3http://php.net/manual/en/function.call-user-func-array.php