Développement d'une API REST (service web) en PHP sans framework

Michaël Mathieu

7 décembre 2023

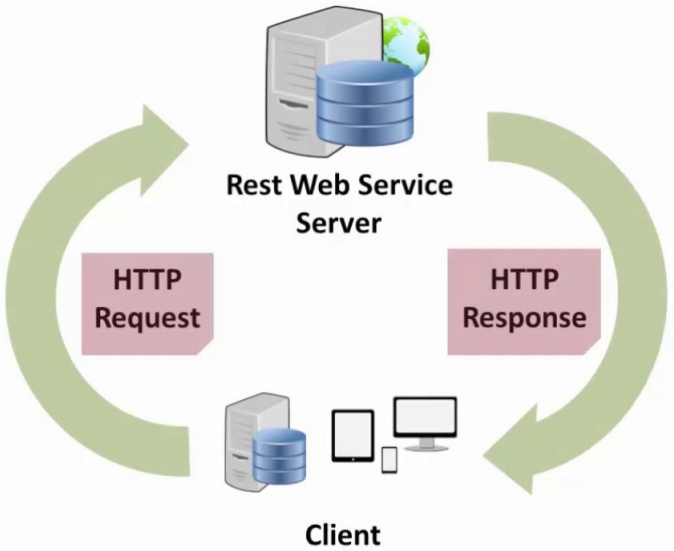


Table des matières

1 Avant–propos

Les requêtes HTTP qui sont utilisées par un navigateur web se limitent aux méthodes `GET` (quand on tape l'adresse d'un site), `POST` (quand on envoie un formulaire) et éventuellement `HEAD` (avant d'envoyer une requête en GET pour télécharger un gros fichier).

Lorsque nous utilisons des services web, nous utilisons davantage de méthodes HTTP. Dans le cadre de ce document, nous allons utiliser les méthodes `OPTIONS`, `GET`, `POST`, `PUT` et `DELETE`.

Pour diverses raisons, il peut être intéressant de sécuriser certains points d'entrée d'une API, nous verrons ici l'utilisation de deux manière : par authentification *basic*, par *token* de sécurité.

Enfin, vu la facilité pour sécuriser la communication avec des outils comme *Let's Encrypt*, il est recommandé de n'accepter que les communications en HTTPS.

2 Les différentes méthodes HTTP

S'il est aisé de comprendre quand utiliser les méthodes `GET` et `DELETE`, dans certaines situations, le développeur peut hésiter entre les méthodes `POST` et `PUT`.
Dans le tableau ci–dessous, nous avons simplement énoncé qu'une création utilisait la méthode `POST` et que pour la modification, il s'agissait de la méthode `PUT`. Si cela est exact dans la majorité des cas, le juge de paix est la notion d'*idempotence*. Cette notion signifie qu'une opération a le même effet qu'on l'applique une ou plusieurs fois.

Méthode	**Body**	**Idempotent**	**Exemple d'utilisation**
`GET`	Non	Oui	Obtenir une ressource
`POST`	Oui	Non	Création d'une ressource
`PUT`	Oui	Oui	Modification d'une ressource
`DELETE`	Non	Oui	Suppression d'une ressource
`OPTIONS`	Non	Oui	Documentation du point d'entrée

3 Les codes réponses HTTP

Nous listons, ici, les codes réponses que nous utilisons habituellement lors de l'écriture d'une API. Cette liste n'est pas exhaustive [1].

Code	**Utilisation**
200	OK – code par défaut Indique que la requête est correcte et que le serveur l'a correctement traitée
201	OK – ressource créée (par exemple après un `POST`)
400	Les paramètres ne sont pas valides
401	Authentification nécessaire
403	Authentification OK, mais l'utilisateur connecté ne possède pas les droits
404	La ressource demandée est inconnue
405	La méthode HTTP utilisée n'est pas supportée par ce point d'entrée
429	L'utilisateur a envoyé trop de requêtes en un temps donné

Pour modifier le code de la réponse, il suffit d'utiliser la méthode `http_response_code`.
⚠pour que l'appel à cette méthode fonctionne, il faut qu'elle soit exécutée avant que votre script PHP n'ait pas encore envoyé la moindre information au client (`echo`, `var_dump`, `readfile`, etc.). Sinon, c'est code réponse par défaut(200) qui sera envoyé et l'appel à cette méthode n'aura aucun effet.

1. Une liste complète peut être trouvée sur `https://developer.mozilla.org/en-US/docs/Web/HTTP/Status`
⚠la version originale en anglais est de meilleur qualité –techniquement parlant– que les versions traduites.

4 Appeler un service web sans coder une seule ligne

Pour tester un service web sans devoir implémenter un programme de test, vous pouvez installer *Postman*[2].
Cette application permet de faire des appels avec les méthodes HTTP `GET`, `POST`, `PUT`, `DELETE`, `HEAD` en ajoutant des paramètres dans le body de la requête HTTP, ainsi que des headers.

5 Ecrire un point d'entrée

5.1 Documentation du point d'entrée

Avant d'exécuter une requête HTTP, les navigateurs appellent le point d'entrée avec la méthode `OPTIONS` afin de récupérer la documentation du point d'entrée. Les informations retournées contiennent essentiellement :

- les domaines autorisés (header `Access-Control-Allow-Origin`, voir point 5.5)
- les headers autorisés (header `Access-Control-Allow-Headers`)
- les méthodes HTTP autorisées (header `Access-Control-Allow-Methods`)

Par exemple, pour un point d'entrée que nous souhaitons accessible de n'importe où et qui accepte les méthodes `GET` et `POST`, notre script commencerait avec ce code :

```
<?php
  header('Access-Control-Allow-Origin: *');
  if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    header('Access-Control-Allow-Headers: *');
    header('Access-Control-Allow-Methods: OPTIONS, GET, POST');
  }
```

La méthode `OPTIONS` est toujours acceptée, car elle documente le point d'entrée.

5.2 Retourner un résultat

Le standard pour structurer les données retournées par un service web REST est le format JSON. Ainsi, un point d'entrée réduit à sa plus simple expression et qui, pour illustrer notre propos, retournerait l'heure système du serveur (requête en `GET`) s'écrirait ainsi :

```
<?php
  header('Access-Control-Allow-Origin: *');
  if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    header('Access-Control-Allow-Headers: *');
    header('Access-Control-Allow-Methods: OPTIONS, GET');
  } else if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    header('Content-Type: application/json');
    echo json_encode(["systemTime" => date("Y-m-d H:i:s")]);
  } else {
    http_response_code(405);
  }
```

Le résultat aura cette forme :

```
{
  "systemTime": "2022-11-18 16:35:59"
}
```

2. `https://www.postman.com/downloads/`

Le premier header `Access-Control-Allow-Origin` indique que notre point d'entrée peut être appelé depuis n'importe quel domaine (voir point 5.5). Le header `Content-Type` précise que le contenu de la réponse est du JSON, mais un service web peut retourner une image, un PDF, du texte brut, etc. (voir point 5.4).
La fonction `json_encode` permet de convertir un objet PHP en JSON. Dans le cas où nous souhaitons retourner plusieurs objets, il faut les placer dans un tableau.
Ce point d'entrée n'acceptant que les requêtes `OPTIONS` et `GET`, si nous recevons une requête d'un autre type, nous retournons une erreur 405 (voir point 3).

5.3 Récupérer le Body d'une requête

```
$bodyRaw = file_get_contents("php://input");
$body = json_decode($bodyRaw, true); // à faire seulement si le body est en JSON
```

5.4 Retourner un fichier

```
$fileOut = "IMG_20201119_082710_8.jpg"; // image to return
$imageInfo = getimagesize($fileOut);

header('Content-Type: ' . $imageInfo['mime']);
header('Content-Length: ' . filesize($fileOut));

readfile($fileOut);
```

5.5 Limiter l'accès de son API à son domaine

Dans la plupart des cas, nous souhaitons que l'API que nous mettons en place soit disponible depuis n'importe où, mais dans certains cas, nous souhaitons n'offrir cette interface qu'à notre site web.

La seule manière de garantir cela est de mettre un filtre sur l'adresse IP du client au niveau de la configuration du serveur ou dans le code PHP (l'adresse IP du client se trouve dans le tableau associatif `$_SERVER`).

Le header Access-Control-Allow-Origin Dans les navigateurs modernes, une sécurité a été ajoutée pour que les API ne puissent plus être appelées depuis un domaine différent que celui dans lequel elles sont déployées [3]. Ainsi, pour qu'une API puisse être appelée depuis un domaine différent, il faut que le serveur l'indique explicitement par l'intermédaire du header `Access-Control-Allow-Origin` ; comme cela a été fait dans l'exemple précédent.
Ce header est nécessaire pour les appels effectués depuis un appareil iOS, mais pas depuis un appareil Android.

Cette pseudo–sécurité joue bien des tours aux développeurs, car les anciennes API ne renvoient pas systématiquement ce header. Dès lors, on ne peut pas appeler ces anciennes API qui avaient pourtant été déployées pour qu'elles soient accessibles depuis n'importe quel domaine.

Il est existe une manière de contourner cette sécurité en faisant l'appel depuis notre serveur PHP (car la librairie *cURL*, permettant de faire des appels à des services web en PHP, ne tient pas compte de ce header), mais ⚠attention les appels au service web en question ne seront plus fait par le client directement, ce qui signifie que **c'est l'adresse IP de votre serveur qui effectuera la requête à l'API. En cas de forte utilisation de votre site web, l'IP de votre serveur pourrait être bloquée, car le quota du nombre de requêtes effectuées serait dépassé (erreur 429, voir point 3).**

3. `https://developer.mozilla.org/fr/docs/HTTP/Access_control_CORS`

Si les données que vous récupérez ne sont pas trop spécifiques au client et si elles ne nécessitent pas d'être up–to–date, une solution consiste à faire que votre serveur PHP, qui joue le rôle de proxy, conserve pendant un certain temps ces données afin d'éviter de solliciter l'API à chaque requête d'un client.
⚠cette solution est la plupart du temps interdite dans les conditions générales d'utilisation des API.

Prenons le cas d'une API retournant des taux de change. Si notre application a besoin d'avoir un taux de change à jour, il est acceptable dans la majorité des cas que ces taux ne soient mis à jour qu'une fois ou deux par heure. Dans ce contexte, il serait intéressant de conserver sur notre serveur, jouant le rôle de proxy, les taux de change pendant quelques minutes. Si cela peut être interdit par les conditions générales d'utilisation de l'API, il faut noter qu'il reste difficile à prouver que ce genre de données volatiles (et qu'on peut obtenir par de multiples sources) aient été conservées en mémoire.

Voici un exemple de proxy web en PHP.
Ce proxy peut être appelé ainsi :
`https://mydomain.com/webproxy.php?url=https://restcountries.eu/rest/v2/all`

```
<?php
$url = str_replace(",", "&", $_GET["url"]);
error_log("Call to " . $url);

header('Access-Control-Allow-Origin: *');
header('Content-Type: application/json');
echo callAPI("GET", $url);

function callAPI($method, $url, $data = false)
{
  $curl = curl_init();
  switch ($method)
  {
    case "POST":
      curl_setopt($curl, CURLOPT_POST, 1);
      if ($data) {
        curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
      }
      break;
    case "PUT":
      curl_setopt($curl, CURLOPT_PUT, 1);
      break;
    default:
      if ($data) {
        $url = sprintf("%s?%s", $url, http_build_query($data));
      }
  }

  curl_setopt($curl, CURLOPT_URL, $url);
  curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
  $result = curl_exec($curl);
  curl_close($curl);
  return $result;
}
```

6 Points d'entrée sécurisés

Souvent les services web sécurisés demandent à ce qu'une clé soit ajoutée comme paramètre dans l'URL de la requête. S'il est aisé de comprendre la motivation d'offrir cette possibilité pour des raisons de simplicité, cette approche comporte quelques failles. Notamment le fait que cette clé se retrouve en clair dans les logs du serveur et qu'une personne mal intentionnée puisse la récupérer et l'utiliser.

La bonne manière consiste n'utiliser que le HTTPS et à transmettre la clé (token) ou les credentials de l'utilisateur dans les headers de la requête HTTP (les headers étant cryptés lors d'une communication HTTPS).
Le header standard pour transmettre cette information est le header `Authorization`.
Nous allons voir en exemple ces deux manières pour s'authentifier auprès d'un service web.

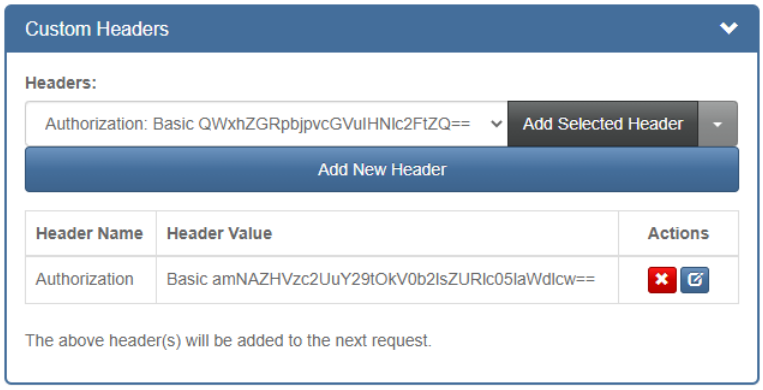
6.1 Authentification Basic

Le concept de cette manière consiste à envoyer à chaque requête les credentials au serveur. Le nom d'utilisateur et le mot de passe (séparés par :) doivent être encodés en Base64 (ASCII), mais ne sont pas cryptés directement. Pour qu'ils le soient, il suffit que la communication soit en HTTPS.

Exemple
Avec le nom d'utilisateur *jc@dusse.com* et le mot de passe *EtoileDesNeiges* :

Credentials à encoder en Base64 (ASCII)	Donnée à placer dans le header
`jc@dusse.com:EtoileDesNeiges`	`amNAZHVzc2UuY29tOkV0b2lsZURlc05laWdlcw==`

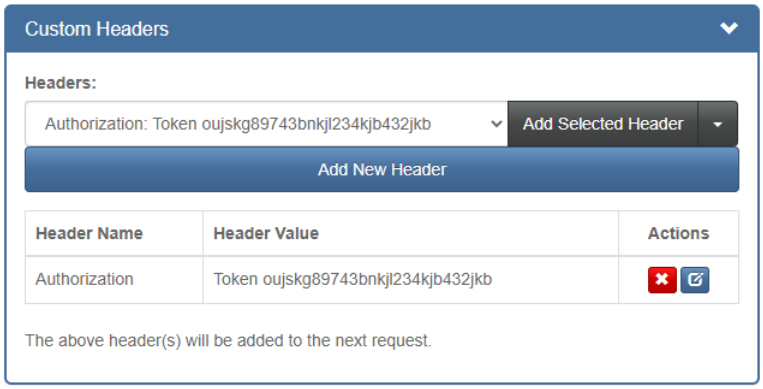
Dans le header `Authorization`, il faut préciser qu'il s'agit de la manière `Basic` d'authentification et ajouter les credentials en Base64. Nous aurions ainsi :



6.2 Utilisation d'un token de sécurité

Cette méthode consiste à envoyer une clé d'authentification au serveur.
Cette clé peut avoir été renvoyée par le serveur après une requête POST contenant un nom d'utilisateur et un mot de passe (comme lorsqu'un humain se connecte avec un formulaire). Soit cette clé peut être créée sur votre site web par le client de votre API, puis il l'intègre à chacune de ses requêtes.
Cette clé ne doit contenir que des chiffres et des lettres non–accentuées.

Dans le header `Authorization`, il faut préciser qu'il s'agit de la manière d'authentification avec un `Token` de sécurité et ajouter la clé. Nous aurions ainsi :



6.3 Récupérer le header Authorization en PHP

⚠selon la configuration du serveur Apache des hébergeurs, le header `Authorization` n'est parfois pas transmis au script PHP ! Dans ces cas–là, il faut ajouter, à la racine de votre dossier `www`, le fichier `.htaccess` (ou le compléter si vous en avez déjà un) avec la règle suivante :

```
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
```

La méthode `getCredentials` retourne les credentials d'une authentification `Basic`. La méthode `getToken` retourne la clé d'une authentification avec un `Token` de sécurité.

```
function getCredentials() {
  $auth = getHeader("Authorization");
  if ($auth == null) {
    return null;
  }

  $auth = explode(" ", $auth);
  if ($auth[0] != "Basic") {
    return null;
  }

  $credentials = explode(":", base64_decode($auth[1]));
  return array (
    'email' => $credentials[0],
    'password' => $credentials[1]
 );
}

function getToken() {
  $auth = getHeader("Authorization");
  if ($auth == null) {
    return null;
  }

  $auth = explode(" ", $auth);
  if ($auth[0] != "Token") {
    return null;
  }

  return $auth[1];
}

function getHeader($key) {
  foreach (getallheaders() as $name => $value) {
    if (strtolower($key) === strtolower($name)) {
      return $value;
    }
  }
  return null;
}
```

7 Des URL conformes aux standards REST

Imaginons que nous avons une API permettant de répertorier des musiques. Un point d'entrée pour récupérer une musique à partir de sa référence (pour cet exemple, une référence valide d'une musique est `vuohewgbpvw`) serait [4] :

```
GET /music/vuohewgbpvw
```

Comment peut–on faire cela en PHP ? Si nous devions écrire un script PHP étant capable de récupérer la référence d'une musique dans l'URL, par exemple `https://mydomain.com/api/music/index.php?refMusic=vuohewgbpvw`, nous pouvons écrire quelque chose comme :

```
$refMusic = filter_var($_GET['refMusic'], FILTER_SANITIZE_NUMBER_INT);
```

Mais cette manière d'exposer un point d'entrée REST n'est pas conforme aux standards. Si nous souhaitons que l'URL `/music/vuohewgbpvw` soit transformée en `/music/index.php?refMusic=vuohewgbpvw`, il faut ajouter, à la racine de votre dossier `www`, le fichier `.htaccess` (ou le compléter si vous en avez déjà un) avec les règles suivantes :

```
RewriteEngine On
RewriteRule music/([A-Za-z0-9]+)/?$ music/index.php?refMusic=$1 [NC,L]
```

Si le fichier `.htaccess` n'est pas pris en compte, il faut s'assurer que la redéfinition est autorisée :

```
<Directory /var/www/>
    ...
    AllowOverride All
    ...
</Directory>
```

- Windows : `httpd.conf`
- Linux : `apache2.conf`

Pour Linux, nous devons également activer le module `mod_rewrite` avec la commande :

```
> sudo a2enmod rewrite
```

Pour aller plus loin, voici trois ressources intéressantes :

- `https://www.phpflow.com/php/create-php-restful-api-without-rest-framework-dependency/`
- `https://httpd.apache.org/docs/current/mod/mod_rewrite.html`
- `https://httpd.apache.org/docs/2.4/rewrite/flags.html`

7.1 Récupérer les paramètres de l'URL

Pour les requêtes en `GET`, `PUT` ou `DELETE`, la référence de la ressource se trouvera dans le tableau `$_REQUEST`.

⚠chez certains hébergeurs, ce tableau ne contient pas les paramètres de l'URL. Pour cela, il faut ajouter en haut du fichier `.htaccess` la ligne suivante :

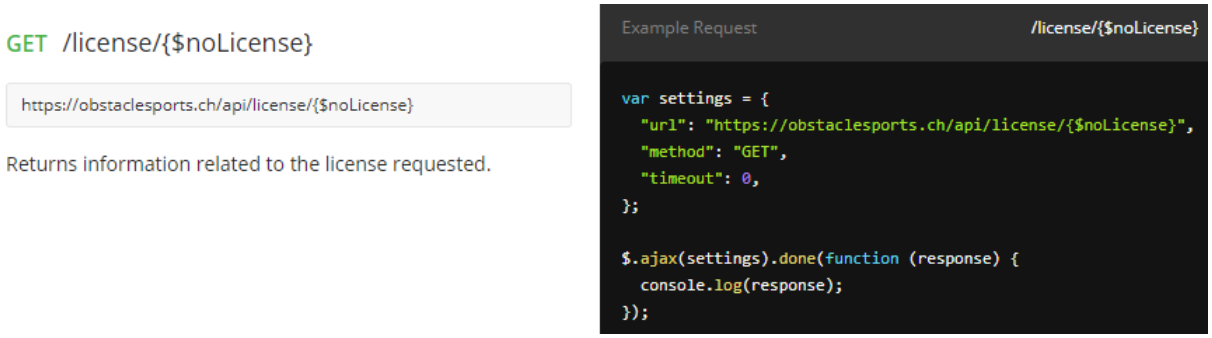
```
Options -MultiViews
```

4. Dans la même veine, nous aurions `PUT /music/vuohewgbpvw` pour mettre à jour les informations concernant cette musique et `DELETE /music/vuohewgbpvw` pour la supprimer.

8 Documenter votre API

Il existe plusieurs outils pour documenter votre API. Un de ceux–ci est *Postman*[5]. La version gratuite n'est pas trop limitée et permet notamment de mettre en ligne une documentation de vos points d'entrée en proposant pour chacun d'entre–eux un snippet (plus de 15 langages sont proposés) pour l'appeler.

Exemple d'une documentation réalisée avec *Postman* : `https://obstaclesports.ch/api/`.



5. `https://postman.com/`