Quelques éléments utiles pour la construction d'un backend php

- **UUID : universally unique identifier**
- identifiants globalement uniques, 16 octets
  - unicité probable
  - v1 : génération basée sur le temps
  - v4 : génération basée sur un aléa
  - v3 : génération basée sur un nom
  - v5 : génération basée sur un nom, destiné à identifier 1 nœud de manière indépendante de son hostname

intérêt, utilisation

- construire des identifiants difficiles à deviner ou à prévoir pour améliorer la sécurité
  - tous les ids basés sur une séquence sont prévisibles
- éviter les problèmes dus à des identifiants dupliqués

en php : plusieurs librairies

- un exemple : ramsey/uuid

```php
use Ramsey\Uuid\Uuid;


$uuid1= Uuid::uuid1() ;
echo "uuid v1 : " . $uuid1->toString() ."\n";
$uuid4= Uuid::uuid4();
echo "uuid v4 : " . $uuid4->toString() ."\n";

$uuid3= Uuid::uuid3(Uuid::NAMESPACE_DNS, 'lbs.local');
echo "uuid v3 : " . $uuid3->toString() ;
$uuid5= Uuid::uuid5(Uuid::NAMESPACE_DNS, 'lbs.local');
echo "uuid v5 : " . $uuid5->toString() ."\n";
```

générer une valeur aléatoire

- pour construire des tokens d'authentification, des identifiants …
- problème difficile si on souhaite une qualité cryptographique :
  - presque impossible à prévoir
  - risque très faible de collision
- nécessite un algorithme offrant une probabilité de dispersion uniforme sur un espace très grand

en php

- les fonctions ~~`rand()`~~, ~~`mt_rand()`~~, ~~`uniqid()`~~ ne sont pas de qualité cryptographique, même combinée avec un hash
- utiliser `openssl_random_pseudo_bytes()`, `random_int()`, `random_bytes()` :

```php
$token = random_bytes(32);
$token = bin2hex($token);
echo "token : " . $token ."\n";

$token = openssl_random_pseudo_bytes(32);
$token = bin2hex($token);
echo "token : " . $token ."\n";
```

programmer un client REST

- Pour programmer des interactions entre services côté backend
- Pour tester une api

- **Programmer un client** :
  - Ligne de commande : curl
  - Php : file_get_content(), php curl
  - Utiliser une classe ou un micro-framework php
    - Pest, HTTPFul
    - **Guzzle**

curl en ligne de commande

```
$ curl -i -H "Accept: application/json" http://api.racoin.local/annonces/2

HTTP/1.1 200 OK
Date: Mon, 25 Jan 2016 14:30:21 GMT
Server: Apache/2.4.16 (Ubuntu)
Connection: close
Transfer-Encoding: chunked
Content-Type: application/json;charset=utf-8

{"annonce":{"id":2,"titre":"Sapiente ut fugiat sed quibusdam ea
voluptate.","descriptif":"Doloribus dolor animi quo excepturi. Vel exercitationem
exercitationem sunt qui ratione totam. Minima laboriosam nihil velit
est.","ville":"Garcia","code_postal":"67922","prix":"770.00","date_online":null,"
status":1,"cat_id":2,"type":"Annonce","categorie":{"id":2,"libelle":"v\
u00e9hicules\/moto","descriptif":"toutes les motos, 2 roues \u00e0
moteur"}},"links":{"annonceur":{"href":"\/annonces\/2\/annonceur"},"categorie":
{"href":"\/annonces\/2\/categorie"}}}
```

php curl

```php
// 1. initialisation
$ch = curl_init();

// 2. configutation de la requête
curl_setopt($ch,CURLOPT_URL,"http://www.nettuts.com");
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_HEADER, 1);

// 3. exécution et récupération du résultat
$output = curl_exec($ch);
$info = curl_getinfo($ch) ;
// 4. fermer
curl_close($ch);
```

Client Rest avec Guzzle

- Forge les requêtes avec la méthode choisie
- Ajoute des headers
- Envoie
- Récupère et décode les données de retour

- installation :

```
{
    "require": {
        "guzzlehttp/guzzle": "~6.0"    // php7
    }
}
```

guzzle overview

créer un client :

```
use GuzzleHttp\Client;
$client = new GuzzleHttp\Client([
    // Base URL : pour ensuite transmettre des requêtes relatives
    'base_url' => 'http://api.command.local',
    // options par défaut pour les requêtes
    'timeout'  => 2.0,
]);
```

envoyer des requêtes :

```
$response = $client->get('/commands');
$response = $client->post('/commands');
$response = $client->request('GET', '/commands/45FS-6HGT-NHR5');
```

utiliser les réponses :

```
$code = $response->getStatusCode();             // 200
$type = $response->getHeader('Content-Type');
$body = $response->getBody() ;
```

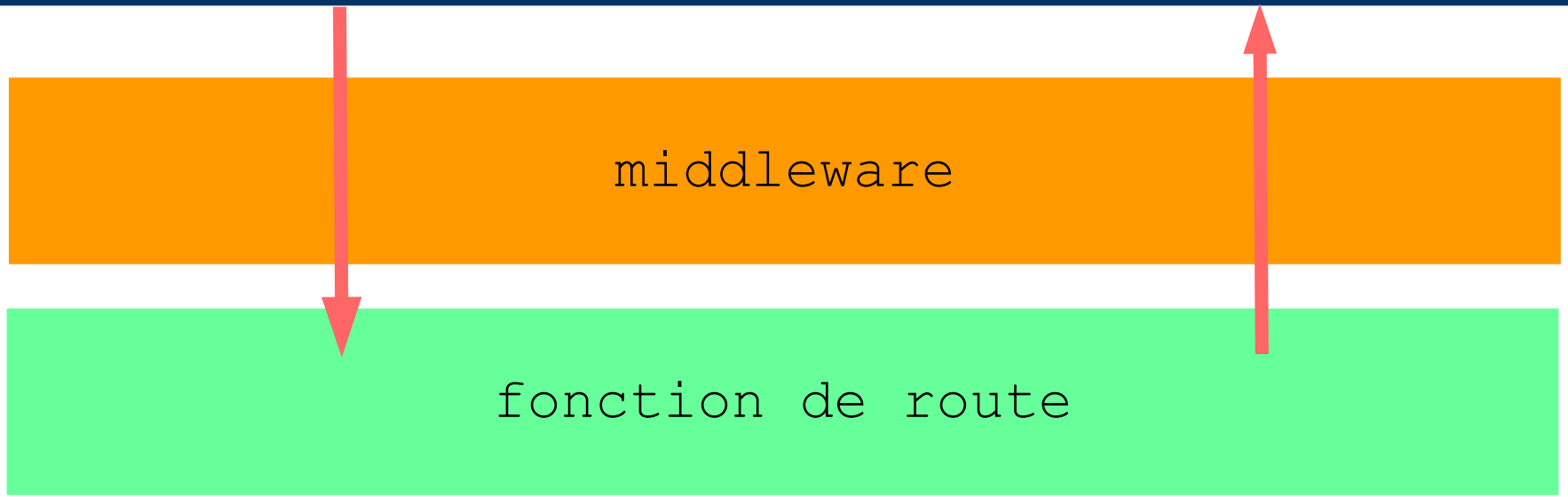
Slim avancé

Middlewares Slim

Middleware Slim

- **Définition :** un *middleware* est une fonction invoquée lors de l'activation d'une route, qui peut modifier la requête ou la réponse courante
  - ***avant*** l'appel de la fonction de route, puis
  - ***après*** l'appel de la fonction de route

```
$app->get('/hello/', function() { … })
    ->add( function() { … } ) ;
```



intérêt, utilisation

- factoriser du code commun à plusieurs routes
- sortir des contrôleurs tout ce qui n'est pas directement lié aux fonctionnalités
- exemples :
  - contrôle d'accès, autorisation
  - contrôle de token
  - ajout de headers : content-type, header CORS
  - validation de données
  - logging d'activité

Middleware Slim

- 1 **middleware** = 1 *callable* (fonction, closure, méthode) recevant **3 paramètres** :
  - 1 requête \Psr\Http\Message\ServerRequestInterface
  - 1 réponse \Psr\Http\Message\ResponseInterface
  - le middleware suivant dans la liste, callable
- 1 middleware **doit** retourner 1 objet réponse

```php
function( Request $rq, Response $rs, callable $next) {

    // actions avant la fonction de route

    // appel du midleware suivant ou de la fonction de route
    $rs = $next($rq, $rs):

    // actions après la fonction de route

    return $rs ;
}
```

- Il est possible d'ajouter **plusieurs** middleware qui sont alors appelés en séquence avant d'appeler la fonction de route
- Chaque middleware appelle le suivant, le dernier middleware appelle la fonction de route

principe

```
$app->get('/hello/', 'hello'
)->add( 'foo' )->add( 'bar' ) ;
```

Activation de la route '/hello/' :

```
bar($rq, $rs, $next){
  // PRE code

  $resp = $next($rq,$rs)

  // POST code
  return $resp ;
```

```
foo($rq, $rs, $next){
  // PRE code

  $resp = $next($rq,$rs)

  // POST code
  return $resp ;
```

```
hello($rq, $rs, $args){
  // application code

  return $resp
```

valeurs partagées entre middleware

- 1 middleware peut transmettre 1 valeur aux middleware suivants ou à la fonction de route en ajoutant 1 attribut à la requête en cours

- positionner la valeur dans le middleware :

```
$request = $request->withAttribute( 'yo' , 73 ) ;
```

- récupérer la valeur dans le middleware suivant ou dans la fonction de route :

```
$bar = $request->getAttribute( 'yo'  ) ;
```

accéder à la route courante dans un middleware

- la route courante est disponible sous la forme d'un objet dans l'attribut `'route'`

```php
function( Request $rq, Response $rs, callable $next) {

    $route = $rq->getAttribute('route') ;

    $name    = $route->getName() ;
    $methods = $route->getMethods() ;

    // accès aux arguments de la route

    $id = $route->getArgument('id') ;
}
```

accéder au conteneur de dépendances

- le conteneur de dépendances est disponible dans les middleware comme dans les fonctions de route :
  - injecté dans le constructeur si le middleware est une méthode dans une classe
  - lié à `$this` si le middleware est une fonction

exemple

```
function checkToken ( Request $rq, Response $rs, callable $next ) {

    // récupérer l'identifiant de cmmde dans la route et le token

    $id = $rq->getAttribute('route')->getArgument( 'id');
    $token = $rq->getQueryParam('token', null);

    // vérifier que le token correspond à la commande
    try {
        Commande::where('id', '=', $id)
            ->where('token', '=',$token)
            ->firstOrFail();
    } catch (ModelNotFoundException $e) {

        // générer une erreur
        return $rs ;
    };

    return $next($rq, $rs);
};
```

exemple

```
$app->get('/commandes/{id}', function ( $rq,  $rs, $args) {
    return (new \lbs\api\controllers\CommandController($this))
                ->getCommand( $rq, $rs, $args);
}
)->setName('commande')
 ->add('checkToken');


$app->post('/commandes/{id}/items', function( $rq, $rs, $args){
    return (new \lbs\api\controllers\CommandController($this))
                ->addItem( $rq, $rs, $args);
}
)->setName('com2item')
 ->add('checkToken');
```

middleware de route et d'application

- Un middleware peut être attaché à 1 route
  - il est activé lorsque la route est sélectionnée :

```
$app->get('/hello/', 'hello'
)->add( 'foo' ) ;
```

- Un middleware peut être attaché à l'application
  - il est activé pour toutes les routes :

```
$app = new \Slim\App( $c ) ;

$app->add( 'bar' ) ;
```

PSR-7

- PSR-7 standardise
  - les messages (request, response) http
  - [les middleware] [en cours de transfert vers PSR-15]

- Slim peut utiliser à priori tous les middleware PSR-7
  - il existe des collections de middleware PSR-7 pour différents usages

validation de données

- vérifier que les données reçues pour créer/modifier une ressource sont présentes et conformes aux valeurs attendues
  - ne remplace pas le filtrage/nettoyage pour raisons de sécurité (injection …)
- tâche compliquée, répétitive

→ **Utiliser une librairie**

→ **déporter le code dans des middleware pour éviter de surcharger les contrôleurs**

exemple : le middleware slim-validation basé sur la librairie respect/validation

- respect/validation : librairie de validation de données en php
  - des validateurs : conditions à vérifier (type, format, valeurs ...)
  - des fonctions de validation : appliquent les validateurs à des données [structurées] et génèrent des messages d'erreurs
- slim-validation : respect/validation dans un middleware psr-7
  - application des validateurs sur les données de la requêtes
  - messages d'erreurs dans un attribut

```php
use \DavidePastore\Slim\Validation\Validation as Validation ;

$validators = … ;


$app->post('/customers[/]',function( $rq, $rs ) {

    if ($rq->getAttribute( 'has_errors' )) {

      $errors = $rq->getAttribute( 'errors' ) ;
      /*
       [ 'field' => [ error messages ] ;
         'field' => [ error messages ]
       ]
       */
    } else {
       /* traiter les données  */
    }

})->add(new Validation( $validators) ) ;
```

```php
use \Respect\Validation\Validator as v;
$validators = [
    'name'       => v::StringType()->alpha() ,
    'login'      => v::alnum()->length(3,8) ,
    'email'      => v::email() ,
    'firstname' => v::optional(v::StringType()->alpha()) ,
    'birth'      => v::date('d-m-Y')->max( 'now' ) ,
    'address'    => [
      'streetAdddress'=>v::alnum()->length(10,128),
      'city'          =>v::alnum()->length(10,64),
      'zipcode'       =>v::alnum()->length(4,10),
     ] ];
```

```json
{ "name" : "michel"  ,
   "login" : "michou54" ,
   "email": "michou54@gmal.com",
   "birth": "25-08-1972" ,
   "address": { "streetAdddress": "12, place Henri",
                "city": "Paris",
                "zipcode": "F-75007"
    } }
```