Les objets de requête et de réponse

Les objets Symfony représentant la requête et la réponse HTTP ont déjà été largement présentés et utilisés dans cet ouvrage. Pour rappel, ils sont respectivement des types **Symfony\Component\HttpFoundation\Request** et **Symfony\Component\Http Foundation\Response**.

Dans les actions des contrôleurs REST, la requête est disponible sous forme d'un paramètre injecté par Symfony dans la méthode, et la réponse constitue le type de retour de ces méthodes, comme dans le fragment de code suivant :

```
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;


...


public function creer(Request $request, SerializerInterface
$serializer): Response
{
  ...
}
```

L'objet de requête n'est pas systématiquement nécessaire. En effet, il n'est utile que lorsque des données seront transportées dans le corps de la requête, typiquement sur des actions liées aux méthodes HTTP **PUT** et **POST**.

1. Le contenu et les en-têtes de requête

Lors de la manipulation des données entrantes, il est nécessaire de pouvoir récupérer les en-têtes et le corps de la requête.

Les en-têtes contiennent notamment le type de données envoyées par le client. Cette information est accessible via la méthode **getContentType()**.

Dans le cas où des données JSON sont transmises dans la requête, elles le sont dans le corps ; ce contenu est accessible avec la méthode **getContent()**.

Voici comment pourrait être structuré le code d'une action recevant un flux JSON en HTTP **POST** :

```
#[Route(
    '/api/article',
    name: 'api_article_creer',
    methods: ['POST']
)]
public function creer(
  Request $request, SerializerInterface $serializer
): Response
{
  // On contrôle le format de données entrant...
  if($request->getContentType() ===application/json') {
      // On récupère le contenu de la requête...
      $data = $request->getContent();
      // On désérialise en JSON...
    $article = $serializer->deserialize(
       $data,
       Article::class,
       "json"
    );

    // Renvoi d'une réponse...
  } else {
    // Renvoi d'un code d'erreur...
  }
}
```

2. Manipulation de la réponse avec Response et JsonResponse

Envoyer une réponse dans notre API REST Symfony consiste simplement à créer une instance de la classe **Symfony\Component\HttpFoundation\Response**, à y

encapsuler les données à retourner, puis à retourner l'objet en fin d'action. C'est Symfony qui se charge de transformer l'objet en flux de réponse HTTP.

Lors de la construction d'un objet de réponse, nous passons au constructeur :

- le contenu à renvoyer ;
- le code de réponse HTTP ;
- les en-têtes de la réponse, notamment le type MIME du contenu renvoyé.

Par exemple, suite à la création réussie d'une ressource, nous renvoyons un message en JSON avec le code HTTP 200 (OK) ; cela se fait avec le code suivant :

```
use Symfony\Component\HttpFoundation\Response;


...


#[Route(
    '/api/article/{id}',
    name: 'api_article_lire',
    methods: ['GET']
)]
public function lire($id): Response
{
  $article = // Récupération d'un article par son id...


  $articleJSON = $this->serializer->serialize($article, "json");


  return new Response(
    $articleJSON,
    Response::HTTP_OK,
    [ 'Content-Type' => 'application/json' ]
  );
}
```

La classe **Response** de Symfony possède des constantes correspondant aux différents codes de réponse HTTP possibles.

Renvoyer une réponse au format JSON

Bien que l'exemple de code précédent soit tout à fait correct, il peut être amélioré. En effet, dans la mesure où nous renvoyons systématiquement des données au format JSON, il est assez répétitif de spécifier à chaque fois le type MIME renvoyé.

Pour simplifier l'envoi d'une réponse au format JSON, Symfony propose une classe qui hérite de la classe **Response**, la classe **Symfony\Component\HttpFoundation\JsonResponse**. Cette classe part du principe que le contenu retourné est systématiquement du JSON et elle positionne automatiquement l'en-tête **Content-Type** à la valeur **application/json**. De plus, la classe **JsonResponse** encode automatiquement les données en JSON. Il n'est donc plus nécessaire de passer par le sérialiseur pour préparer les données.

Le code précédent peut donc être simplifié comme ceci :

use **Symfony\Component\HttpFoundation\JsonResponse;**

...

#[Route(

'/api/article/{id}',

name: 'api_article_lire',

methods: ['GET']

)]

public function lire($id): Response

{

$article = // Récupération d'un article par son id...

return new JsonResponse(

$article, // L'objet est sérialisé automatiquement !

Response::HTTP_OK

);

}

3. Les codes de réponse HTTP dans une API REST

a. Problématique de l'état de la réponse

D'un point de vue d'un client d'une API REST, le contenu renvoyé dans la réponse est évidemment l'information principalement attendue. Cependant des problèmes peuvent survenir lors de l'invocation d'un traitement sur l'API ; cela peut être dû à des informations erronées transmises par le client ou bien à une erreur d'exécution côté serveur.

Classiquement, dans une application PHP, on génère des exceptions pour notifier du problème. Dans le cas d'une API REST, cela n'est pas possible, tout simplement parce que le protocole HTTP ne permet pas de renvoyer des exceptions.

b. Expression de la réponse avec HTTP

L'expression du bon fonctionnement du traitement ou, au contraire d'une erreur, devra passer par l'utilisation des codes de réponse HTTP.

Un résultat d'exécution correct sera exprimé avec les codes HTTP de type **2XX**. Les codes de réponse HTTP de type **4XX** représentent des erreurs liées au client alors que les codes de type **5XX** représentent des erreurs liées au serveur (dans ce contexte, à l'application Symfony). En voici quelques-uns pour rappel :

Code	Message	Explication
200	OK	Requête traitée avec succès. La réponse dépendra de la méthode de requête utilisée.
201	Created	Requête traitée avec succès et création d'un document.
204	No Content	Requête traitée avec succès mais pas d'information à renvoyer.
400	Bad Request	La syntaxe de la requête est erronée.
401	Unauthorized	Une authentification est nécessaire pour accéder à la ressource.
402	Payment Required	Paiement requis pour accéder à la ressource.
403	Forbidden	Les droits d'accès ne permettent pas au client d'accéder à la ressource.

Code	Message	Explication
404	Not Found	Ressource non trouvée.
405	Method Not Allowed	Méthode de requête non autorisée.
500	Internal Server Error	Erreur interne du serveur.
501	Not Implemented	Fonctionnalité réclamée non supportée par le serveur.
503	Service Unavailable	Service temporairement indisponible ou en maintenance.

c. Mise en œuvre

Lorsque les actions des contrôleurs invoquent des services applicatifs, il y a de forts risques que des exceptions soient à gérer. Dans le cas où tout se passe correctement, on renvoie le flux de réponse attendu associé à un code **2XX**.

Dans le cas où une exception est levée, il faut au contraire prévoir une réponse associée à un message d'erreur, ainsi qu'un code HTTP approprié de type **4XX** ou **5XX**.

Prenons le scénario suivant : une action d'un contrôleur est sollicitée en HTTP GET sur l'URI **/article/{id}**, elle renvoie un article en fonction de la valeur d'id transmise dans l'URL. Voici le code permettant d'implémenter le bon fonctionnement de ce traitement ainsi que le cas où l'article n'est pas trouvé :

```
#[Route(
  '/api/article/{id}',
  name: 'api_article_lire',
  methods: ['GET']
)]
public function lire(int $id): Response
{
  try {
    $article = // Récupération de l'article par son id via
          // service...
    return new JsonResponse(
      $article,
```

```
        Response::HTTP_OK
      );
    }
    catch(\Exception $e) {
      return new JsonResponse(
        [ 'message' => $e->getMessage() ],
        Response::NOT_FOUND
      );
    }
}
```

En cas d'exception, on renvoie un code d'erreur **404 (NOT FOUND)** associé à un message JSON contenant le message de l'exception d'origine ; dans ce cas, la structure renvoyée serait de cette forme :

```
{
  "message": "Article introuvable !"
}
```