4 minutes

Retourner une réponse avec les méthodes de la classe AbstractController

Rappels sur le format `JSON`

Le format `JSON` (pour `JavaScript Object Notation`) est le format le plus utilisé sur le `Web`.

Il s'agit d'un format d'échange pour les données dans des applications `Web`.

Le `JSON` n'est pas utilisé qu'en `JavaScript`, vous le retrouverez dans toutes les applications `Web` modernes.

Les spécifications du format `JSON` regroupent un ensemble de règles pour utiliser cette notation.

Règle sur les guillemets doubles

En `JSON`, les clés et les chaînes de caractères doivent obligatoirement entourées de guillemets doubles.

Types en `JSON`

Il existe 6 types en `JSON` : **les chaîne de caractères, les nombres (entier ou décimal), les booléens, `null`, les tableaux et les objets**.

En tant que développeur `PHP` vous pouvez voir les tableaux comme des tableaux `PHP` indexés numériquement et les objets comme des tableaux associatifs.

Exemples de `JSON`

Voici un exemple de `JSON` :

```
{
  "espece": "Chien",
  "race": "Labrador",
  "age": 6,
  "physique": {
  "yeux": "marron",
  "pelage": "jaune",
  "poids": 30
  }
}
```

Copier

Autre exemple avec tous les types possibles :

```
{
  "_id": "6086c872c542806fa4ef686b",
  "index": 0,
  "guid": "12defb60-591d-4b8f-9a29-1b227f529c48",
  "isActive": true,
  "balance": "$1,622.70",
  "picture": "http://placehold.it/32x32",
  "age": 28,
  "eyeColor": "blue",
  "name": "Ivy Wright",
  "gender": "female",
  "company": "FUTURIZE",
  "email": "ivywright@futurize.com",
  "phone": "+1 (979) 428-3499",
  "address": "111 Fountain Avenue, Maybell, Connecticut, 2726",
  "about": "Ex aliquip do laboris proident amet velit reprehenderit ea cupidatat sunt anim laborum et. Pariatur ea eu eu duis enim mollit non ipsum ir
  "registered": "2018-03-10T06:35:15 -01:00",
  "latitude": 22.632444,
  "longitude": 152.00962,
  "tags": [
    "mollit",
    "sunt",
    "esse",
    "incididunt",
    "aliqua",
    "irure",
    "deserunt"
  ],
  "friends": [
    {
    "id": 0,
    "name": "Wilkins Maxwell"
    },
    {
    "id": 1,
    "name": "Richards Marks"
    },
    {
    "id": 2,
    "name": "Davis Sweet"
    }
  ],
  "greeting": "Hello, Ivy Wright! You have 7 unread messages.",
  "favoriteFruit": "strawberry"
}
```

Copier

La méthode `json()` de la classe abstraite `AbstractController`

Allons étudier le code de la méthode `json()` sur la classe abstraite `AbstractController` :

```
<?php
protected function json($data, int $status = 200, array $headers = [], array $context = []): JsonResponse
{
  if ($this->container->has('serializer')) {
    $json = $this->container->get('serializer')->serialize($data, 'json', array_merge([
      'json_encode_options' => JsonResponse::DEFAULT_ENCODING_OPTIONS,
    ], $context));

    return new JsonResponse($json, $status, $headers, true);
  }

  return new JsonResponse($data, $status, $headers);
}
```

Copier

Cette méthode permet de faciliter la création d'une `Response` au format `JSON`.

Elle permet de prendre un argument `$data` contenant des données à encoder au format `JSON`.

Elle vérifie si le composant `Serializer` est disponible. Ce composant `Symfony` permet de prendre un objet dans un format et de l'encoder / décoder dans un autre format. Il est plus robuste que la méthode `json_encode()` c'est pour cette raison qu'il est utilisé s'il est disponible.

S'il n'est pas disponible c'est la classe `JsonResponse()` qui va se charger d'encoder l'objet en `JSON` avec la méthode `json_encode()`.

Cela permet d'être encore plus concis pour créer une réponse au format `JSON`.

Par exemple, essayez de mettre dans notre `DefaultController` :

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController
{
  #[Route('/', name: 'index')]
  public function index(): Response
  {
    return $this->json('Coucou');
  }
}
```

Copier

Nous pouvons accéder à la classe avec `$this` puis à la méthode `json()` que nous venons de voir.

La méthode `file()` de la classe abstraite `AbstractController`

La méthode `file()` permet d'automatiquement créer une réponse de type binaire et de mettre les bons en-têtes pour renvoyer un fichier au navigateur client :

```
<?php
protected function file($file, string $fileName = null, string $disposition = ResponseHeaderBag::DISPOSITION_ATTACHMENT): BinaryFileResponse
{
  $response = new BinaryFileResponse($file);
  $response->setContentDisposition($disposition, null === $fileName ? $response->getFile()->getFilename() : $fileName);

  return $response;
}
```

Copier

`BinaryFileResponse` est une classe `Symfony` qui permet de créer une réponse binaire pour un fichier. Nous ne la verrons pas en détail car la classe fait 400 lignes.

`setContentDisposition()` permet de mettre l'en-tête `attachment` : `Content-Disposition: attachment; filename="nomdufichier.extension"` qui va permettre au navigateur de déclencher un téléchargement avec le nom de fichier souhaité.

Cette méthode nécessite l'installation du composant `Mime` :

```
composer require symfony/mime
```

Copier

Il permet de gérer le standard `MIME`, qui est le standard permettant d'indiquer la nature et le format d'un document. Les navigateurs utilisent le type `MIME` d'un fichier pour déterminer la façon dont ils vont traiter ou afficher un document.

Quelques exemples de types sont `text/plain`, `image/jpeg`, `video/mp4`, `application/octet-stream` etc.

Si vous êtes sur `GNU/Linux` vous n'avez rien d'autre à faire. Si vous êtes sur `Windows` activez l'extension `fileinfo` dans le fichier `php.ini`.

Essayons maintenant d'envoyer un fichier en utilisant la méthode `file()` de la classe abstraite `AbstractController`.

Créez un fichier `test.txt` avec le contenu que vous voulez dans le dossier `public` puis modifiez le `DefaultController` :

```
<?php

namespace App\Controller;
```

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController
{
  #[Route('/', name: 'index')]
  public function index(): Response
  {
    return $this->file('test.txt');
  }
}
```

Copier

Cela déclenchera le téléchargement du fichier par le navigateur.

La méthode `render()` de la classe abstraite `AbstractController`

La méthode `render()` permet d'automatiquement rendre une vue (c'est-à-dire de créer une vue à partir d'un moteur de rendu, également appelé moteur de `template HTML`) et de la retourner au navigateur.

Pour ce faire, elle a besoin d'un moteur de rendu et nous allons utiliser `Twig` qui est celui utilisé par défaut et recommandé par `Symfony`.

Nous allons donc installer `Twig` :

```
composer require twig
```

Copier

Notez l'installation automatique de nombreuses dépendances et l'exécution de deux recettes qui ont automatiquement ajouté les configurations nécessaires pour que `Twig` fonctionne. A savoir :

Le fichier `bundles.php` :

```
<?php

return [
  Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
  Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
  Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
  Twig\Extra\TwigExtraBundle\TwigExtraBundle::class => ['all' => true],
];
```

Copier

Le fichier de configuration pour `Twig config/packages/twig.yaml` :

```
twig:
  default_path: '%kernel.project_dir%/templates'

when@test:
  twig:
    strict_variables: true
```

Copier

Ce fichier indique notamment que les `templates Twig` seront placés dans le dossier `templates`.

Le dossier `templates` et le fichier `templates/base.html.twig` :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Welcome!{% endblock %}</title>
    {# Run `composer require symfony/webpack-encore-bundle`
      and uncomment the following Encore helpers to start using Symfony UX #}
    {% block stylesheets %}
      {#{{ encore_entry_link_tags('app') }}#}
    {% endblock %}

    {% block javascripts %}
      {#{{ encore_entry_script_tags('app') }}#}
    {% endblock %}
  </head>
  <body>
    {% block body %}{% endblock %}
  </body>
</html>
```

Copier

Modifions maintenant le contrôleur `DefaultController` pour rendre cette vue préconfigurée :

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
```

```
class DefaultController extends AbstractController
{
  #[Route('/', name: 'index')]
  public function index(): Response
  {
    return $this->render('base.html.twig');
  }
}
```

Copier