Les conventions PSR

Les conventions PSR sont des standards de programmation (ou “bonne pratique”) spécifiques à PHP. L’idée est d’harmoniser les méthodes de développements.

On peut les regrouper en 4 grandes catégories :

Le chargement automatique : PSR-4
Les interfaces : PSR-3, PSR-6, PSR-11 …
HTTP ou la gestion des messages HTTP : PSR-7, PSR-15 …
Et les styles de codage : PSR-1, PSR-12.

Les PSR-1 et PSR-2 / PSR-12 : les bases :

Autant commencer cette présentation des PSR par les numéros 1 et 2 (ou 12 ! Vous comprendrez bientôt où je veux en venir). Je vous les présente ensemble parce qu’elles sont liées.

Elles fixent les conventions minimales de codage, les éléments requis pour assurer un haut niveau d’interopérabilité entre les codes PHP partagés.

La PSR-1 présente les normes de codage de base :

Les fichiers doivent utiliser uniquement les balises <?php et <?= .
Le code doit utiliser les balises longues ou à écho court et ne doit pas utiliser les autres variantes de balises.
Le code PHP doit utiliser uniquement UTF-8 (le codage de caractères universel/international pour éviter les défauts d’affichage des caractères).
Les noms de méthode doivent être déclarés dans camelCase().

```php
<?php
namespace Vendor\Model;

class Foo
{
    const VERSION = '1.0';
    const DATE_APPROVED = '2012-06-01';
}
```

exemple : "Les constantes de classe DOIVENT être déclarées en majuscules avec des séparateurs de soulignement"

La **PSR-2** intitulée “Guide de style de codage” est certainement l’une des plus connues. Elle a été déclarée comme obsolète en 2019 et remplacée par la PSR-12 “Guide de style de codage étendu”.

En voici quelques extraits :

-La limite logicielle sur la longueur de ligne doit être de 120 caractères.
-Les lignes de devraient pas contenir plus de 80 caractères ou alors être divisées en plusieurs lignes de 80 caractères maximum chacune.
-Tous les mots-clés et types PHP réservés doit être en minuscules.
-Toute accolade de fermeture ne doit pas être suivie d'un commentaire ou d'une déclaration sur la même ligne.
-Et bien d'autres !

"l'accolade d'ouverture pour la classe DOIT aller sur sa propre ligne"

```php
<?php

namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable
{
    // constants, properties, methods
}
```

exemple :

La PSR-4, une amélioration de la PSR-0 :

La PSR-4 décrit une spécification pour les classes de chargement automatique à partir de chemin de fichiers. Vous n'avez rien compris ? C'est normal, on rentre un peu dans la technique…

Un projet Web est constitué de plusieurs fichiers rangés dans des dossiers. En PHP, on utilise ce qu'on appelle des "namespaces (ou espaces de noms)" pour représenter la structure, l'organisation des fichiers dans le code.

Les namespaces sont des groupes, des conteneurs de noms. On s'en sert pour distinguer deux éléments qui ont le même nom.

Voici un exemple : un fichier nommé "rose" peut se retrouver à la fois dans le namespace "couleurs" et dans le namespace "fleurs".

La PSR-4 donne des directives sur la façon d'écrire ces namespaces pour donner une présentation de la structure des fichiers la plus claire possible.

"Un nom de classe complet a la forme suivante:"

```
\<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>
```

La PSR-7, interfaces de messages HTTP :

Les messages HTTP sont la base du développement Web.

Vous ne le voyez pas mais le navigateur web que vous utilisez crée des requêtes HTTP, qui sont envoyées à un serveur Web, qui fournit alors une réponse HTTP.

Illustrons un peu la chose pour mieux comprendre le concept :

Vous êtes sur la page web 1. Vous cliquez sur la pagination pour accéder à la page 2. Votre navigateur envoie un message HTTP (une requête) au serveur, de type “Donne-moi la page 2”. Le serveur lui envoie un message HTTP en retour, une réponse : “Voici la page que tu as demandée”.

On ne va pas entrer ici dans des détails techniques mais sachez que la PSR-7 décrit la façon de présenter les messages HTTP dans le code.

Aucun de ces PSRs n’est obligatoire, mais il est fortement recommandé de s’en servir afin d’optimiser sa manière de coder, de gagner du temps et surtout de faciliter la coopération entre diverses équipes de développement.Tous les frameworks et CMS ont pour objectif de mettre en place ces PSRs.S’il ne l’ont pas encore tous atteint, une grande partie est déjà opérationnelle et il est aujourd’hui aisé d’ajouter ces interfaces qui faciliteront grandement la vie des devs.

```php
<?php
//class dont on a besoin (classe Repository.php obligatoire)
require_once("Repository.php");
require_once(ROOT . "/model/entity/Produit.php");

class ProduitRepository extends Repository
{
    public function getLesProduits()
    {
        $lesProduits = array();
        $db = $this->dbConnect();
        $req = $db->prepare("SELECT id, libelle FROM produit order by libelle");
        $req->execute();
        $lesEnregs = $req->fetchAll();
        foreach ($lesEnregs  as $enreg) {
            $unProduit = new Produit(
                $enreg->id,
                $enreg->libelle,
            );
            array_push($lesProduits, $unProduit);
        }
        return $lesProduits;
    }
}
```

Présentation de l’utilisation de la norme PSR avec la classe ProduitRepository.

Lien de documentation:

- https://www.adimeo.com/blog/psr-les-standards-de-programmation-en-php
- https://www.php-fig.org/
- https://www.itefficience.com/article/qu-est-ce-que-les-psrs-et-a-quoi-servent-ils/