SYMFONY : 6–Injection de dépendances

Dans Symfony nous avons un service container, en gros tout ce qui est contenu dans Symfony est géré par Symfony, si nous observons, la classe BlogController, nous ne l'avons jamais instanciée, c'est Symfony lui-même qui se charge de l'instancier, Symfony a donc un grand rôle a joué , il instancie des classes et appel ses fonctions.

Votre application *regorge* d'objets utiles: un objet "Mailer" peut vous aider à envoyer des e-mails tandis qu'un autre objet peut vous aider à enregistrer des choses dans la base de données. Presque *tout ce* que votre application «fait» est en fait réalisé par l'un de ces objets. Et chaque fois que vous installez un nouveau bundle, vous avez accès à encore plus!

Dans Symfony, ces objets utiles sont appelés **services** et chaque service vit à l'intérieur d'un objet très spécial appelé **conteneur de service** . Le conteneur vous permet de centraliser la façon dont les objets sont construits. Il vous facilite la vie, favorise une architecture solide et est super rapide!

Symfony dans le service container a la possibilité de nous donner ce dont on a besoin quand on lui dit.

Par exemple la fonction index() a pour rôle de nous afficher la liste des articles de la BDD et pour fonctionner, elle a donc besoin d 'un repository (requête de sélection), quand une fonction a besoin de quelque chose pour fonctionner, on appelle ça une **dépendance**, la fonction dépend d'un repository pour aller chercher la liste des articles.

Donc si nous avons une dépendance, nous pouvons demander à Symfony de nous la fournir plutôt que de la fabriquer moi-même.

La fonction index() ce n'est pas moi qui vais l'appeler mais Symfony.

Nous devons donc fournir à la méthode index() en argument, un objet issue de la classe ArticleRepository

```
 public function index(ArticleRepository $repo)
```

Ne pas oublier l'appel de la classe :

```
use App\Repository\ArticleRepository;
```

Du coup nous n'avons plus besoin de la ligne suivante :

```
$repo = $this->getDoctrine()->getRepository(Article::class);
```

La mettre en commentaire.

Faire la même chose pour la fonction show() :

```
public function show(ArticleRepository $repo, $id)
    {
        //$repo = $this->getDoctrine()->getRepository(Article::class);

        $article = $repo->find($id);
```

```
        return $this->render('blog/show.html.twig', [
            'article' => $article
        ]);
    }
```

On peut encore aller plus loin :

```
public function show(Article $article)
    {
        //$repo = $this->getDoctrine()->getRepository(Article::class);

        // $article = $repo->find($id);

        return $this->render('blog/show.html.twig', [
            'article' => $article
        ]);
    }
```

Tout ça grâce au **ParamConverter** de Symfony, en gros il voit que l'on a besoin d'un article et aussi d'un ID, il va donc chercher l'article avec l'identifiant et l'envoyer à la fonction show().

L' @ParamConverterannotation appelle des *convertisseurs* pour convertir les paramètres de demande en objets. Ces objets sont stockés en tant qu'attributs de requête et peuvent donc être injectés en tant qu'arguments de méthode de contrôleur:

Symfony comprend qu'il y a un article a passé et que dans la route il y a un ID, il va donc chercher le bon article avec le bon identifiant, cela pourrait très bien marcher avec le titre, le nom etc…

Nous avons donc des fonctions beaucoup plus courte.