TD1 – Paquets PHP Composer, Routage via l'URL

Dans les 3 premiers TDs, nous allons développer une API REST en PHP. Afin de pouvoir se concentrer sur l'apprentissage des nouvelles notions, nous allons partir du code existant d'un site Web de type **réseau social** appelé '**The Feed**'. Ce site contiendra un fil principal de publications et un système de connexion d'utilisateur.

L'intérêt de ce site est qu'il ne contient que 2 contrôleurs et un petit nombre d'actions :

- contrôleur `Publication` :
  - lire les publications : action `feed`
  - écrire une publication : action `submitFeedy`
- contrôleur `Utilisateur` :
  - afficher la page personnelle avec seulement ses publications : action `pagePerso`
  - s'inscrire :
    - formulaire (action `afficherFormulaireCreation`),
    - traitement (action `creerDepuisFormulaire`)
  - se connecter :
    - formulaire (action `afficherFormulaireConnexion`),
    - traitement (action `connecter`)
  - se déconnecter : action `deconnecter`

Exercice 1

1. Récupérer le code de base en forkant vous-même ce dépôt GitLab.
2. Il faut donner les droits en lecture / exécution à Apache (utilisateur `www-data`).

   ```
   setfacl -R -m u:www-data:r-x .
   ```

   Comme le site enregistre une photo de profil pour chaque utilisateur, il faut donner les droits en écriture sur le dossier `web/assets/img/utilisateurs/`.

   ```
   setfacl -R -m u:www-data:rwx ./web/assets/img/utilisateurs
   ```

3. Importez les tables `utilisateurs` et `publications` dans votre base de données SQL préférée :
   - Pour *MySQL*, vous devez :
     - exécuter le script d'import MySQL,
     - mettre à jour le fichier de configuration `src/Configuration/ConfigurationBDDMySQL.php` avec votre login et mot de passe.
   - Pour *PostgreSQL*, vous devez :
     - exécuter le script d'import PostgreSQL,
     - mettre à jour le fichier de configuration `src/Configuration/ConfigurationBDDPostgreSQL.php` avec votre login et mot de passe,
     - préciser la bonne classe de configuration `ConfigurationBDDPostgreSQL` au niveau du constructeur de `src/Modele/Repository/ConnexionBaseDeDonnees.php`
     - dans les classes `PublicationRepository` et `UtilisateurRepository`, modifier les `$data['nomDeColonne']` pour mettre tous les noms de colonnes en minuscule. En effet, *PostgreSQL* passe en minuscule tous les identifiants (sauf s'ils sont entourés de guillemets doubles `"`, auquel car il faudra toujours y faire référence avec des guillemets doubles).
4. Créez un nouvel utilisateur et une nouvelle publication.
   Souvenez-vous bien de votre identifiant et mot de passe car nous nous en resservirons.
5. Faites marcher le site. Explorez toutes les pages.

Dans l'optique de développer une *API REST*, nous aurons besoin que les URL des pages de notre site n'utilisent plus le *query string*.

Par exemple, la route

```
web/controleurFrontal.php?controleur=publication&action=feed
```

va devenir `web/`. Et la route

```
web/controleurFrontal.php?controleur=utilisateur&action=afficherFormulaireConnexion
```

deviendra `web/connexion`.

Pour ceci, nous allons utiliser une bibliothèque PHP existante, et donc un gestionnaire de bibliothèques : `Composer`.

1. Le gestionnaire de paquets `Composer`

`Composer` est utilisé dans le cadre du développement d'applications PHP pour installer des composants tiers. `Composer` gère un fichier appelé `composer.json` qui référence toutes les dépendances de votre application.

1.1 Initialisation et *Autoloading* de `Composer`

`Composer` fournit un *autoloader*, *c.-à-d.* un chargeur automatique de classe, qui satisfait la spécification `PSR-4`. En effet, cet *autoloader* est très pratique pour utiliser les paquets que nous allons installer via `Composer`.

Commençons donc par remplacer notre *autoloader* `Psr4AutoloaderClass.php` par celui de `Composer`.

Exercice 2

1. Créer un fichier `composer.json` à la racine du site Web avec le contenu suivant

```
{
   "autoload": {
      "psr-4": {
         "TheFeed\\": "src"
      }
   }
}
```

2. Si vous modifiez le fichier `composer.json`, par exemple pour mettre à jour vos dépendances, vous devez exécuter la commande :

```
composer update
```

Aide : Pour ceux qui sont sur leur machine personnelle, vous devrez installer `composer` sur votre machine. Aller voir la documentation de `composer` à cet effet. Pour Linux, il suffit d'installer un paquet. Pour Windows avec *XAMPP*, l'installateur Windows marche très bien.

3. Quand on installe une application ou un nouveau composant, `composer` place les librairies téléchargées dans un dossier `vendor`. Il n'est pas nécessaire de versionner ce dossier souvent volumineux.
Rajoutez donc une ligne `/vendor/` à votre `.gitignore`. **Dites** aussi à *Git* d'ignorer son fichier de configuration interne `/composer.lock`.

4. Modifiez le fichier `web/controleurFrontal.php` comme suit :

```
-use TheFeed\Lib\Psr4AutoloaderClass;
-
-require_once __DIR__ . '/../src/Lib/Psr4AutoloaderClass.php';
-
-// instantiate the loader
-$loader = new Psr4AutoloaderClass();
-// register the base directories for the namespace prefix
-$loader->addNamespace('TheFeed', __DIR__ . '/../src');
-// register the autoloader
-$loader->register();
+require_once __DIR__ . '/../vendor/autoload.php';
```

Aide : Ce format montre une modification de fichier, similaire à la sortie de `git diff`. Les lignes qui commencent par des `+` sont à ajouter, et les lignes avec des `-` à supprimer.

5. Testez votre site qui doit marcher normalement.

1.2 Archivage du routeur par *query string*

Nous allons déplacer le code de routage actuel dans une classe séparée, dans le but de bientôt la remplacer.

Exercice 3

1. Dans le fichier `web/controleurFrontal.php`, faites le changement suivant. Toutes les lignes supprimées de ce fichier doivent être déplacées dans la méthode statique `traiterRequete` d'une nouvelle classe `src/Controleur/RouteurQueryString.php`.

```
-// Syntaxe alternative
-// The null coalescing operator returns its first operand if it exists and is
not null
-$action = $_REQUEST['action'] ?? 'feed';
-
-
-$controleur = "publication";
-if (isset($_REQUEST['controleur']))
-    $controleur = $_REQUEST['controleur'];
-
-$controleurClassName = 'TheFeed\Controleur\Controleur' . ucfirst($controleur);
-
-if (class_exists($controleurClassName)) {
-    if (in_array($action, get_class_methods($controleurClassName))) {
-        $controleurClassName::$action();
-    } else {
-        $controleurClassName::afficherErreur("Erreur d'action");
-    }
-} else {
-           TheFeed\Controleur\ControleurGenerique::afficherErreur("Erreur   de
contrôleur");
-}
+TheFeed\Controleur\RouteurQueryString::traiterRequete();
```

2. Testez votre site qui doit marcher normalement.

2. Nouveau routeur par Url

Exercice 4

1. Créez une nouvelle classe `src/Controleur/RouteurURL.php` vide avec le code suivant.

```
<?php
namespace TheFeed\Controleur;

class RouteurURL
{
   public static function traiterRequete() { }
}
```

2. Appelez ce nouveau routeur en modifiant `web/controleurFrontal.php` :

```
-TheFeed\Controleur\RouteurQueryString::traiterRequete();
+TheFeed\Controleur\RouteurURL::traiterRequete();
```

Nous allons maintenant coder ce nouveau routeur.

2.1 Le composant `HttpFoundation`

Comme le dit sa documentation, le composant `HttpFoundation` défini une couche orientée objet pour la spécification *HTTP*. En *PHP*, une requête est représentée par des variables globales (`$_GET`, `$_POST`, `$_FILES`, `$_COOKIE`, `$_SESSION`, ...), et la réponse est générée par des fonctions (`echo`, `header()`, `setcookie()`, ...). Le composant `HttpFoundation` de `Symfony` remplace ces variables globales et fonctions par une couche orientée objet.

Pour information, `Symfony` est l'un des 2 principaux *framework* de développement de site Web professionnels en PHP. Dans ce cours, nous nous attacherons aux notions derrière `Symfony` plutôt qu'à `Symfony` lui-même. Ainsi, vos connaissances vous permettront de vous adapter plus facilement à de nouveaux outils, que ce soit `Symfony` ou autre chose... Pour ces raisons, nous n'utiliserons que des composants de `Symfony`.

Dans notre cas, nous allons tout d'abord utiliser la classe `Request` de `HttpFoundation` pour représenter une requête HTTP. Notez que `HttpFoundation` possède des classes aussi pour les réponses HTTP, les en-têtes HTTP, les cookies, les sessions (et les messages flash 😉). Nous utiliserons plus tard les classes liées aux réponses HTTP : `Response`, `RedirectResponse` pour les redirections et `JsonResponse` pour les réponses au format *JSON*.

Exercice 5

1. Exécutez la commande suivante dans le terminal ouvert au niveau de la racine de votre site web

```
composer require symfony/http-foundation
```

Remarque : Certaines dépendances de *Symfony* nécessite une version de PHP `> 8.1`. Si vous n'avez pas cette version sur votre machine personnelle, vous pouvez peut-être demander une version plus ancienne de cette dépendance.

Dans un premier temps, notre site va utiliser des URL comme

```
web/controleurFrontal.php/
web/controleurFrontal.php/connexion
web/controleurFrontal.php/inscription
```

La classe `Request` sera intéressante notamment car elle permet de récupérer chemin qui nous intéresse (`/`, `/connexion` ou `/inscription`).

Exercice 6

1. Dans `RouteurURL::traiterRequete()`, initialisez l'instance suivante de la classe `Requete`

```php
use Symfony\Component\HttpFoundation\Request;

$requete = Request::createFromGlobals();
```

Explication : La méthode `createFromGlobals()` récupère les informations de la requête depuis les variables globales `$_GET`, `$_POST`, ... Elle est à peu près équivalente à

```php
$requete = new Request($_GET,$_POST,[],$_COOKIE,$_FILES,$_SERVER);
```

2. La méthode `$requete->getPathInfo()` permet d'accéder au bout d'URL qui nous intéresse (`/`, `/connexion` ou `/inscription`).

   Affichez cette variable dans `RouteurURL::traiterRequete()` et accédez aux URL précédentes pour voir chemin s'afficher.

2.2 Le composant `Routing`

Comme l'indique sa documentation, le composant `Routing` de `Symfony` va permettre de faire l'association entre une URL (par ex. `/` ou `/connexion`) et une action, c'est-à-dire une fonction PHP comme

`ControleurPublication::feed`.

Exercice 7

1. Exécutez la commande suivante dans le terminal ouvert au niveau de la racine de votre site web

```
composer require symfony/routing
```

2. Créez votre première route avec le code suivant à insérer dans `RouteurURL::traiterRequete()` :

```
use Symfony\Component\Routing\Route;
use Symfony\Component\Routing\RouteCollection;

$routes = new RouteCollection();

// Route feed
$route = new Route("/", [
   "_controller" => "\TheFeed\Controleur\ControleurPublication::feed",
]);
$routes->add("feed", $route);
```

Explication : Une nouvelle `Route $route` associe au chemin `/` la méthode `feed()` de `ControleurPublication`. Puis cette route est ajoutée dans l'ensemble de toutes les routes `RouteCollection $routes`.

3. Les informations de la requête essentielles pour le routage (méthode `GET` ou `POST`, *query string*, paramètres *POST*, ...) sont extraites dans un objet séparé :

```
use Symfony\Component\Routing\RequestContext;

$contexteRequete = (new RequestContext())->fromRequest($requete);
```

Ajoutez cette ligne et affichez temporairement son contenu.

4. Nous pouvons alors rechercher quelle route correspond au chemin de la requête courante :

```
use Symfony\Component\Routing\Matcher\UrlMatcher;

$associateurUrl = new UrlMatcher($routes, $contexteRequete);
$donneesRoute = $associateurUrl->match($requete->getPathInfo());
```

Ajoutez ce code et affichez temporairement le contenu de `$donneesRoute`. Où se trouve l'information de la méthode PHP à appeler ?

5. **Ajoutez** le code suivant pour appeler enfin l'action PHP correspondante :

```
call_user_func($donneesRoute["_controller"]);
```

Explication : La fonction `call_user_func($nomFonction)` exécute la fonction dont le nom est stocké dans `$nomFonction`. Elle est proche du code `$nomFonction()`, mais accepte des entrées plus générales – nous la préférerons donc.

6. Votre site doit désormais répondre correctement à une requête à l'URL `web/controleurFrontal.php`.

2.3 Réécriture d'URL

Passons à notre deuxième route : `/connexion`.

Exercice 8

1. Ajoutez la deuxième route :

```php
use TheFeed\Controleur\ControleurUtilisateur;

// Route afficherFormulaireConnexion
$route = new Route("/connexion", [
                                                    "_controller"                =>
"\TheFeed\Controleur\ControleurUtilisateur::afficherFormulaireConnexion",
   // Syntaxes équivalentes
              //     "_controller"     =>     ControleurUtilisateur::class     .
"::afficherFormulaireConnexion",
                //     "_controller"      =>      [ControleurUtilisateur::class,
"afficherFormulaireConnexion"],
]);
$routes->add("afficherFormulaireConnexion", $route);
```

Notez les syntaxes équivalentes :

- l'attribut statique constant `NomDeClasse::class` d'une classe `NomDeClasse` est remplacé par le nom de classe qualifié, c.-à-d. le nom de classe précédé du nom de package.
- De manière générale, la valeur associée à `_controller` devra être au format `callable`, car c'est ce qui est accepté par `call_user_func()`. Parmi les `callable`, on trouve le format `["NomDeClasseQualifie", "NomMethodeStatique"]` pour les méthodes statiques, ou encore `[$instanceDeLaClasse, "NomMethode"]` pour les méthodes classiques.

2. Testez la page `web/controleurFrontal.php/connexion` qui doit marcher, sauf les liens vers le CSS et les photos qui deviennent invalides. Cherchez pourquoi ces liens se sont cassés.

   Aide : Dans le code source de la page Web (`Ctrl+U`), cliquez sur ces liens cassés pour voir sur quel URL ils renvoient.

Nous allons régler ce problème en changeant l'URL de nos pages de `web/controleurFrontal.php/connexion` vers une URL plus classique `web/connexion`. Pour ceci, nous allons configurer *Apache* pour rediriger la requête `web/connexion` vers l'URL `web/controleurFrontal.php/connexion`.

Exercice 9

1. Enregistrez ce fichier de configuration d'*Apache* fourni par *Symfony* à la place de `web/.htaccess`.

   Remarque :

   - Si la réécriture d'URL ne marche pas à l'IUT (message d'erreur `Internal Server Error`), vous avez peut-être enregistré le fichier dans `.htaccess` au lieu de `web/.htaccess`.
   - Si la réécriture d'URL sur votre machine personnelle ne marche pas, une cause possible est qu'il faut activer le module `mod_rewrite` de votre serveur Apache.

2. Testez que la page `web/connexion` marche et que le CSS et les images sont revenus. En effet, l'URL de base des liens relatifs est de nouveau `web/`.

3. Changez les liens dans `vueGenerale.php` :

```
-<a   href="controleurFrontal.php?controleur=publication&action=feed"><span>The
Feed</span></a>
+<a href="./"><span>The Feed</span></a>

-<a href="controleurFrontal.php?controleur=publication&action=feed">Accueil</a>
+<a href="./">Accueil</a>

-<a                                                 href="controleurFrontal.php?
action=afficherFormulaireConnexion&controleur=utilisateur">Connexion</a>
+<a href="./connexion">Connexion</a>
```

2.4 Route selon la méthode HTTP

L'un des avantages de notre routage est qu'il peut rediriger différemment selon la méthode *HTTP* employée. Voici ce que nous allons faire :

- URL `/connexion`, méthode `GET` → action `afficherFormulaireConnexion` du contrôleur utilisateur
- URL `/connexion`, méthode `POST` → action `connecter` du contrôleur utilisateur

Pour limiter une route à certaines méthodes *HTTP*, on utilise par exemple

```
$route->setMethods(["GET"]);
```

Exercice 10

1. Modifiez votre routeur pour avoir les 2 routes `web/connexion` selon la méthode *HTTP*.

   Attention : Le nom de chaque route doit être unique (`$routes->add("nomRoute", $route);`). Si vous définissez deux routes avec le même nom, la deuxième écrase la première.

2. Corrigez l'URL vers laquelle renvoie `src/vue/utilisateur/formulaireConnexion.php`.

3. Vérifiez que la connexion au site marche bien.

2.5 Ajout des routes manquantes

Exercice 11

1. Ajoutez les routes manquantes (sauf celle vers `pagePerso`) :
   - URL `/deconnexion`, méthode `GET` → action `deconnecter` du contrôleur utilisateur
   - URL `/feedy`, méthode `POST` → action `submitFeedy` du contrôleur publication
   - URL `/inscription`, méthode `GET` → action `afficherFormulaireCreation` du contrôleur utilisateur
   - URL `/inscription`, méthode `POST` → action `creerDepuisFormulaire` du contrôleur utilisateur
2. Modifiez les liens correspondants dans
   - `src/vue/publication/liste.php`,
   - `src/vue/utilisateur/formulaireCreation.php`
   - `src/vue/vueGenerale.php`.

2.6 Routes variables

Avec l'ancien routeur `RouteurQueryString`, nous pouvions envoyer des informations supplémentaires dans l'URL, par exemple l'identifiant d'un utilisateur avec `controleur=utilisateur&action=pagePerso&idUser=19`.

Dans notre nouveau système d'URL, certaines parties de l'URL serviront à récupérer ces informations supplémentaires. Par exemple, nous allons configurer notre site pour que l'URL `web/utilisateur/19` renvoie vers la page personnelle de l'utilisateur d'identifiant `19`. Le routeur fourni par `Symfony` permet des routes variables `/utilisateur/{idUser}` qui permettront d'extraire `$idUser` de l'URL.

Exercice 12

1. Créez une nouvelle route :
   - URL `/utilisateur/{idUser}`, méthode `GET` → action `pagePerso` du contrôleur utilisateur
2. Modifiez `pagePerso()` pour qu'il prenne `$idUser` en argument au lieu de le lire depuis le *query string* avec `$_REQUEST['idUser']`.

```
-public static function pagePerso(): void
+public static function pagePerso($idUser): void

-    if (isset($_REQUEST['idUser'])) {
-        $idUser = $_REQUEST['idUser'];

-    } else {
-        MessageFlash::ajouter("error", "Login manquant.");
-        ControleurUtilisateur::rediriger("publication", "feed");
-    }
```

3. Si vous testez la route, vous verrez qu'elle ne marche pas, car `call_user_func` appelle `pagePerso` sans lui donner d'arguments (il attend `$idUser`).

4. Affichez `$donneesRoute` pour voir comment `UrlMatcher` a extrait `idUser` de l'URL.

Nous allons résoudre ce problème en introduisant un nouveau composant.

2.7 Le composant `HttpKernel` de `Symfony`

Selon sa documentation, le composant `HttpKernel` de `Symfony` fournit un processus structuré pour convertir une `Request` en `Response`. Sa classe principale `HttpKernel` est similaire à notre `RouteurURL`, mais en plus évolué. Nous ne nous servirons donc pas de `HttpKernel` puisque nous recodons une version simplifiée plus compréhensible.

Nous allons plutôt nous concentrer sur les classes `ControllerResolver` et `ArgumentResolver`. La responsabilité du résolveur de contrôleur est de déterminer le contrôleur et la méthode à appeler en fonction de la requête. La classe `ControllerResolver` se limite plus ou moins à lire `$donneesRoute["_controller"]`. Nous pourrions nous en passer, mais elle sera utile plus tard quand vous aurez des actions qui sont des méthodes non statiques (*cf.* séance sur les tests avec `PhpUnit`).

La classe `ArgumentResolver` va construire la liste des arguments de l'action du contrôleur. Par exemple, c'est cette classe qui va créer l'argument `$idUser` avec la valeur `19` pour la méthode `ControleurUtilisateur::pagePerso($idUser)`.

Exercice 13

1. Importez le composant `HttpKernel`

```
composer require symfony/http-foundation symfony/routing symfony/http-kernel
```

2. Faites évoluer le code de `RouteurURL` en rajoutant à la fin (juste avec `call_user_func`)

```
use Symfony\Component\HttpKernel\Controller\ArgumentResolver;
use Symfony\Component\HttpKernel\Controller\ControllerResolver;

$requete->attributes->add($donneesRoute);

$resolveurDeControleur = new ControllerResolver();
$controleur = $resolveurDeControleur->getController($requete);

$resolveurDArguments = new ArgumentResolver();
$arguments = $resolveurDArguments->getArguments($requete, $controleur);
```

et en modifiant

```
-call_user_func($donneesRoute["_controller"]);
+call_user_func_array($controleur, $arguments);
```

3. Testez la route `web/utilisateur/19` en remplaçant `19` par un identifiant d'utilisateur ayant quelques publications. La page doit remarcher, mais pas le CSS ni les images.

Plus d'explications (optionnel) : Revenons sur la classe `ArgumentResolver` pour expliquer son fonctionnement (simplifié) sur l'exemple `pagePerso()` :

- En utilisant l'introspection de PHP, le code accède à la liste des arguments (type et nom)
- pour chaque argument, on essaye itérativement l'un des résolveurs d'arguments pour déterminer la valeur de l'argument.
  Dans notre exemple, le premier résolveur (classe `RequestAttributeValueResolver`) va regarder si le nom de l'argument `idUser` est présent dans `$requete->attributes` (équivalent à `$donneesRoute`). Comme c'est le cas alors on renvoie cette valeur.

L'avantage de ce mécanisme est qu'il permet de récupérer beaucoup de types d'arguments dans le contrôleur :

- un attribut extrait de la requête (attribut `GET` ou `POST`). Pour ceci, le nom de l'attribut doit correspondre,
- la requête `Request $requete` (l'argument doit avoir le type `Request`),
- la valeur par défaut d'une route variable,
- des services du conteneur de service (*cf.* future séance SAÉ sur les tests avec `PHPUnit`),
- des éléments de la base de données si le type correspond à celui d'une entité (`DataObject` dans ce cours)

2.8 Générateur d'URL et conteneur global

Les liens vers le style CSS et les images de profil de notre site sont souvent cassés car elles utilisent des URL relatives. En effet, la base de l'URL varie selon le chemin demandé :

- pour le chemin `web/connexion`, les URL relatives utilisent la base `web/`.
- pour le chemin `web/utilisateur/19`, les URL relatives utilisent la base `web/utilisateur`. Du coup, les liens relatifs sont cassés.

Nous allons utiliser des classes de `Symfony` pour générer automatiquement des URL absolues. D'un côté, nous allons utiliser `UrlHelper` pour générer des URL absolues :

```
use Symfony\Component\HttpFoundation\RequestStack;

$assistantUrl = new UrlHelper(new RequestStack(), $contexteRequete);
$assistantUrl->getAbsoluteUrl("assets/css/styles.css");
// Renvoie l'URL .../web/assets/css/styles.css, peu importe l'URL courante
```

D'un autre côté, la classe `UrlGenerator` génère des URL absolues à partir du nom d'une route. C'est pratique si on doit changer le chemin de la route *a posteriori*.

```
use Symfony\Component\Routing\Generator\UrlGenerator;

$generateurUrl = new UrlGenerator($routes, $contexteRequete);
$generateurUrl->generate("submitFeedy");
// Renvoie ".../web/feedy"
$generateurUrl->generate("pagePerso", ["idUser" => 19]);
// Renvoie ".../web/utilisateur/19"
```

Comme nous allons avoir besoin de ces services de génération d'URL dans différentes vues, il faut pouvoir les initialiser au début de l'application, et de pouvoir y accéder globalement. Dans le cours de développement Web du semestre 3, nous avions fait le choix d'avoir des classes statiques utilisant le patron de conception *Singleton*. Ce choix a l'inconvénient de rendre difficile les tests.

En attendant la séance de SAÉ sur les tests avec *PhpUnit*, nous allons utiliser une classe `Conteneur` pour stocker globalement les services dont nous aurons besoin.

Exercice 14

1. Créez une classe `src/Lib/Conteneur.php` avec le code suivant :

```
<?php

namespace TheFeed\Lib;

class Conteneur
{
   private static array $listeServices;

   public static function ajouterService(string $nom, $service) : void {
      Conteneur::$listeServices[$nom] = $service;
   }

   public static function recupererService(string $nom) {
      return Conteneur::$listeServices[$nom];
   }
}
```

2. Initialisez les deux services `$assistantUrl` et `$generateurUrl` dans `RouteurUrl` (*cf.* code plus haut). Puis stockez-les dans le conteneur avec le nom que vous souhaitez.

3. Récupérez les deux services en haut de la vue `vueGenerale.php`. Puis utilisez-les dans toutes les vues pour passer tous les liens en URL absolues (`<a href="">`, `<img src="">`, `<form action="">` et `<link href="">`).

   Remarques :

   - `$generateurUrl->generate()` échappe les caractères spéciaux des URLs. Vous devez donc lui donner les données brutes, et non celles échappées par `rawurlencode()`.
   - `$assistantUrl->getAbsoluteUrl()` n'échappe pas les caractères spéciaux des URL. À vous de le faire.
   - Vous pouvez utiliser la syntaxe raccourcie `<?= $var ?>` équivalente à `<?php echo $var ?>` pour améliorer la lisibilité de vos vues.

Il ne nous reste qu'à mettre à jour la méthode de redirection et notre site aura fini sa première migration pour des routes basées sur les URL !

Exercice 15

1. Changer la méthode `ControleurGenerique::rediriger()` pour qu'elle prenne en entrée le nom d'une route et un tableau optionnel de paramètres pour les routes variables (mêmes arguments que `$generateurUrl->generate()`). Cette fonction doit maintenant rediriger vers l'URL absolue correspondante. Vous aurez besoin de récupérer un service du `Conteneur`.
2. Mettez-à-jour tous les appels à `ControleurGenerique::rediriger()`.
3. Testez votre site.

3. Conclusion

Dans ce TD, nous avons découvert comment changer les URL associées à notre site pour qu'elles soient plus standard. Cela a été l'occasion de plonger dans le fonctionnement interne d'un routeur professionnel. Ceci vous sera utile si vous apprenez *Symfony* ou un autre framework *backend* plus tard. Concernant le cours *Complément Web*, le passage à ces URL est une étape nécessaire dans notre chemin pour développer une *API REST*.

Enfin, maintenant que vous connaissez les bases de *Composer*, vous pouvez facilement rajouter des bibliothèques à votre site web *PHP*.

Morceaux de TD issus du cours de développement Web en Licence Pro APIDAE de Malo Gasquet.