R4.01 - TP 04 - Doctrine, Refactoring
1 séance de 2h

4.0. Travail à réaliser – Vue d'ensemble

L'objectif de ce TP est de remplacer la source de données temporaire que nous utilisions jusqu'alors, c'est-à-dire le service `BoutiqueService`, par des entités persistantes de Doctrine : `Categorie` et `Produit`. L'accès à ces données se fera toujours *via* des services : les `Repository` que Doctrine fournit pour chaque type d'entité.
A la fin de ce TP, vous devrez supprimer le service `BoutiqueService` et votre application devra fonctionner comme auparavant, mais avec des données qui proviendront maintenant d'une BD (MariaDB).

4.1. Configurez votre projet pour utiliser Doctrine

Vous devez tout d'abord configurer votre projet Symfony pour lui indiquer quel serveur de BD vous allez utiliser, quelle sera la BASE sur ce serveur et avec quel LOGIN/PASSWORD vous vous y connecterez. Pour cela vous devez modifier, dans le fichier **`.env`** présent à la racine de votre projet, la variable `DATABASE_URL` de la façon suivante :

```
DATABASE_URL=mysql://LOGIN:PASSWORD@ellsworth.iut2.upmf-grenoble.fr:3306/BASE?serverVersion=10.11.9-MariaDB&charset=utf8mb4
```

*Dans votre fichier **`.env`**, le texte ci-dessus doit tenir sur une seule ligne !!*

Où vous remplacerez :

- **`LOGIN`** par **`etu_login`** où **`login`** est votre login UGA habituel (8 caractères)
- **`PASSWORD`** par **votre numéro étudiant Apogée** (8 chiffres)
- **`BASE`** par **`symfony_login`** où **`login`** est votre login UGA habituel (8 caractères)

Si vous ne connaissez pas votre numéro étudiant Apogée, vous pouvez le retrouver ici :
https://scolarite-informatique.iut2.univ-grenoble-alpes.fr/app/ficheEtudiant.php

Une fois ce paramétrage réalisé, Doctrine pourra utiliser la BD indiquée pour y faire persister les entités de votre application. Vous pourrez consulter le contenu de cette BD (MariaDB, un *fork* de MySQL) en utilisant l'interface Web PhpMyAdmin disponible sur le serveur `ellsworth`, interface à laquelle vous vous connecterez en utilisant les identifiants que vous avez saisis dans le fichier `.env` : https://iut2-dg-scolarite.iut2.univ-grenoble-alpes.fr/info/phpmyadmin/

⚠ Sur phpmyadmin, si vous collez votre mot de passe après l'avoir copié depuis le site scolarite-informatique, un espace sera rajouté au bout : il faudra penser à l'enlever. Sinon le plus simple est de taper votre numéro apogée « à la main ».

4.2. Créez vos entités `Categorie` et `Produit`

- Dans le terminal, à la racine de votre projet, créez une première entité **`Categorie`** en utilisant la commande : **`php bin/console make:entity`** (voir cours 04, page 5 et page 20)
  Cette entité devra comporter les attributs suivants (*attention, utilisez bien les même noms d'attributs que ceux utilisés précédemment dans `BoutiqueService` pour que le « refactoring » ne soit pas pénible ensuite*) :
  - **`libelle`** (type string 255)
  - **`visuel`** (type string 255)
  - **`texte`** (type text)

- De la même façon, créez une deuxième entité **`Produit`** avec les attributs suivants :
  - **`libelle`** (type string 255)
  - **`visuel`** (type string 255)
  - **`texte`** (type text)
  - **`prix`** (type decimal, précision 10, scale 2)
  - **`categorie`** (type relation ManyToOne vers Categorie, **avec relation inverse**)

- Allez regarder le code que Symfony a produit pour vous dans les répertoires src/Entity et src/Repository.

- Il faut maintenant enregistrer cette modification de votre modèle de données dans une migration et exécuter cette migration sur le serveur SQL pour les changements soient pris en compte. Pour cela taper les 2 commandes suivantes :

  `php bin/console make:migration`

  `php bin/console doctrine:migrations:migrate`

4.3. Peuplez votre BD avec des « fixtures »

Auparavant, nous disposions dans `BoutiqueService` de données (4 catégories, 12 produits) qui étaient stockées « en dur » dans des tableaux. C'était bien utile pour tester notre code. Nous aimerions maintenant disposer de ces mêmes données, sous forme d'entités, pour pouvoir continuer à tester notre code. Symfony propose un mécanisme, appelé **Fixtures**, pour disposer, dans un projet, d'entités de test qui pourront être créées à la demande. Pour utiliser ce mécanisme, il faut installer un bundle Symfony par :

```
composer require --dev orm-fixtures
```

Les entités « fixtures » doivent être créées « programmatiquement » dans une classe `AppFixtures.php` qu'il faudra placer dans le répertoire `src/DataFixtures`. On vous fournit ce répertoire et son contenu, sur Chamilo, fichier : `DataFixtures.zip`. Vous devez en décompresser le contenu et le placer dans le répertoire **`src`** de votre projet.

Regardez le code fourni dans `AppFixtures.php` : il donne un exemple de création d'entités avec Doctrine.

Pour exécuter ce code et « charger » dans la BD vos entités de test, utilisez la commande :

```
php bin/console doctrine:fixture:load
```

Vous pouvez alors aller consulter le contenu de vos tables dans votre BD et vérifier que les entités « fixtures » ont bien été sauvegardées en base.

4.4. Refactoring : se passer du service `BoutiqueService`

Nous sommes prêts maintenant à nous passer du service **`BoutiqueService`** et à utiliser à la place les Repository de Doctrine que nous avons peuplé avec nos données de test.

Vous devez donc maintenant reprendre votre code et, à chaque endroit où vous utilisiez **`BoutiqueService`**, vous devez maintenant utiliser à la place les services **`CategorieRepository`** et **`ProduitRepository`** (*cf* cours 04, pages 12-15)

Normalement, si vous avez respecté les consignes données en TP, le code concerné se trouve dans les classes :

- `BoutiqueController`
- `PanierController`
- `PanierService`

4.5. Définir une requête de recherche de produit dans `ProduitRepository`

Si vous avez mis en place dans la barre de navigation le champ « recherche de produit », vous avez dû utiliser, dans `BoutiqueController`, la méthode `findProduitByLibelleOrTexte` du service `BoutiqueService`.

Il n'y a pas d'équivalent à cette méthode parmi les méthodes qui sont fournies par défaut dans `ProduitRepository` et vous allez donc devoir la développer (voir cours 04, pages 16-18).

Dans le fichier `src/Repository/ProduitRepository.php`, développez une nouvelle méthode :

```php
/**
 * @return Produit[] Returns an array of Produit objects
 */
public function findByLibelleOrTexte(string $recherche): array {
  // à compléter
}
```

Indice : Il faut utiliser l'opérateur DQL/SQL « `like` » pour chercher tous les produits dont le libellé ou le texte contient la chaîne `$recherche`

4.6. Supprimer le service `BoutiqueService`

Lorsque tout fonctionne à nouveau comme avant, mais grâce à Doctrine, vous pourrez supprimer le fichier `src/Service/BoutiqueService.php` de votre projet, il ne nous sert plus à rien !