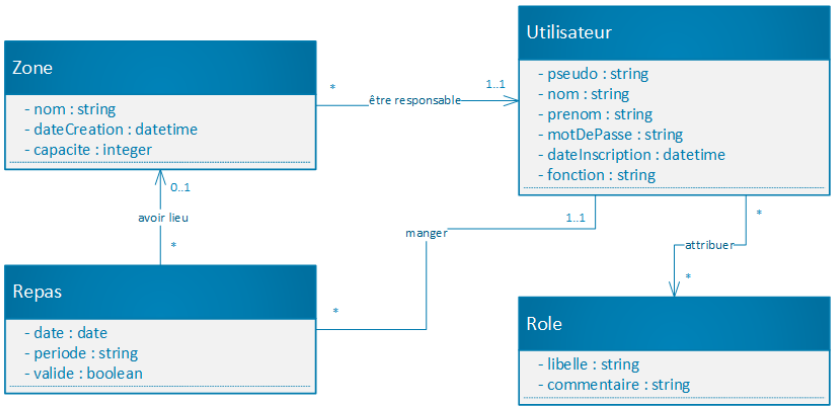
Bloc 2 - SLAM	**Activité n°3 - Partie 7 - ORM Associations ManyToMany**
Compétences	Concevoir et développer une solution applicative : - Exploiter les ressources du cadre applicatif (framework) - Identifier, développer, utiliser ou adapter des composants logiciels - Exploiter les fonctionnalités d'un environnement de développement et de tests - Utiliser des composants d'accès aux données
Objectifs	Maitriser la programmation au sein d'un cadre applicatif (framework) : structure, outil d'aide au développement et de gestion des dépendances, techniques d'injection des dépendances. Développer au sein d'architectures applicatives : concepts de base et typologies. Persister les données liées à la solution applicative et les exploiter à travers un langage de requête présent dans l'outil de correspondance objet-relationnel (ORM).

Nous allons poursuivre l'application de forum qui nous permet de découvrir l'ORM Doctrine et ainsi développer plus rapidement. Dans cette partie, nous allons aborder la gestion des associations de type 0,N - 0,N (Merise) ou * - * (UML).
L'évolution de l'application va concerner la gestion des rôles de nos utilisateurs. Le diagramme de classes fait apparaitre cette modification :



I. Les différents rôles des utilisateurs

Les rôles ci-dessous sont prévus et correspondront aux habilitations des utilisateurs de l'application : utilisateur – ajout_repas – supp_repas – gestionnaire – ajout_zone – supp_zone.

Suite à son inscription, un utilisateur ne se voit attribué aucun rôle. A la validation de son compte, il obtiendra alors communément les rôles utilisateur, ajout_repas et supp_repas.
Le rôle gestionnaire correspond à la possibilité d'ajouter ou de supprimer les repas des autres.
Un administrateur est un utilisateur qui possède tous les rôles existants.

II. Ajout de Role et relation avec Utilisateur

1. Association ManyToMany

Symfony propose la relation ManyToMany, déclarée à l'aide d'une annotation Doctrine :

```
/**
 * @ORM\ManyToMany(targetEntity=NomdeLaClasse::class)
 */
private $lesObjets;
```

Il est possible d'écrire manuellement ce code ou d'utiliser la commande ***symfony console make:entity*** en créant un attribut de type relation.

Au niveau de la base de données, les cardinalités 0,N - 0,N vont se traduire par une table intermédiaire de jointure. Mais nous, nous devons en termes de conception objet ne pas nous occuper de cette table : c'est le rôle de l'ORM Doctrine.
Il est possible de modifier le nom de la table de jointure grâce à une annotation :

```
    * @ORM\JoinTable(name="nouveauNom")
```

2. Application

Travail à réaliser :

- Créez la classe ***Role*** à l'aide de la commande ***make:entity***.
- Mettez à jour la classe ***Utilisateur*** en lui ajoutant l'attribut ***roles*** et déclarez cet attribut de type relation ManyToMany. Conformément au diagramme de classes et particulièrement au sens de navigabilité, la classe ***Role*** n'a pas besoin de posséder la référence aux utilisateurs.
- Analysez le code obtenu dans les deux entity : sous quelle forme Symfony stocke la collection de Role dans la classe Utilisateur ? Quelles sont les méthodes générées afin de gérer la collection de rôles ?
- Analysez maintenant la base de données pour découvrir la table traduisant la relation ManyToMany.

III. Prise en compte des rôles dans l'application

1. Insertion des rôles à l'aide d'une migration

Les rôles sont figés et seront rarement modifiés. Il est donc inutile de prévoir des IHM pour gérer les rôles. Nous allons donc directement insérer les enregistrements des rôles dans la base de données.
Evidemment il est possible d'insérer les rôles par une requête SQL passée à la console de gestion du SGBD. Mais l'ORM Doctrine offre un autre moyen. Il exécute les modifications de la base de données grâce à des classes Migration situées dans le répertoire src/Migrations.

Vous pouvez copier le code d'une migration existante ou utiliser la console de Symfony pour générer une classe vide : ***symfony console doctrine:migrations:generate***

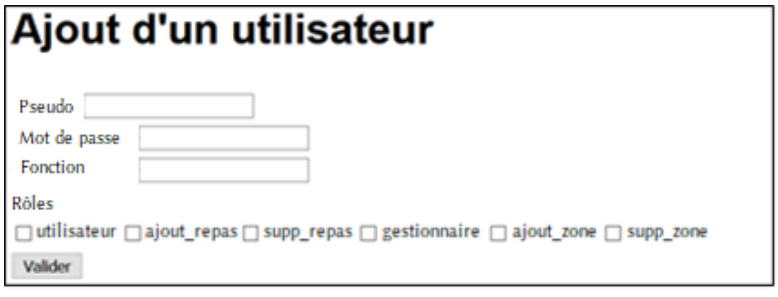
L'exécution de l'ensemble des migrations se fait à l'aide de la commande : ***symfony console doctrine:migrations:migrate*** ou alors une migration précise avec : ***symfony console doctrine:migration:execute <numéro de migration>***.

Travail à réaliser :

- Analysez une classe Migration existante dans le répertoire ***src/Migrations***.
- Créez une nouvelle migration qui insère les rôles indiqués directement dans la table rôle de la base de données.
- Exécutez cette migration et vérifiez le résultat obtenu en base de données.

2. Mise à jour du formulaire UtilisateurType

L'ajout d'un utilisateur doit maintenant permettre de choisir ses rôles :



Pour obtenir ce formulaire, nous allons récupérer l'ensemble des rôles existants et créer pour chacun une case à cocher. Pour cela, vous allez utiliser le champ de type ***EntityType::class*** qui est la solution la plus simple. La classe ***EntityType*** permet de récupérer très simplement l'ensemble des objets d'une classe modèle Entity et de les afficher dans un formulaire sous plusieurs formes : liste simple, ensemble de cases à cocher, boutons radio… Consultez la documentation de Symfony :

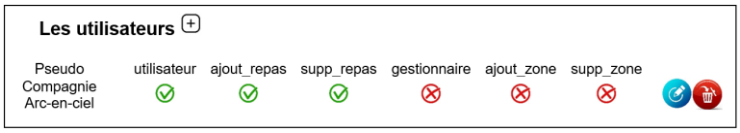
https://symfony.com/doc/current/reference/forms/types/entity.html

Travail à réaliser :

- Modifiez le formulaire d'ajout/modification d'un utilisateur afin de pouvoir lui affecter des rôles via des cases à cocher.
- Testez l'insertion d'un utilisateur puis sa modification.

3. Mise à jour de la liste des utilisateurs

L'affichage des utilisateurs doit permettre de visualiser leurs habilitations :



Travail à réaliser :

- Modifiez la vue Twig correspondant à l'affichage de la liste des utilisateurs (route ***utilisateur/list***), de façon à visualiser les droits de chaque utilisateur. A vous de choisir le visuel…