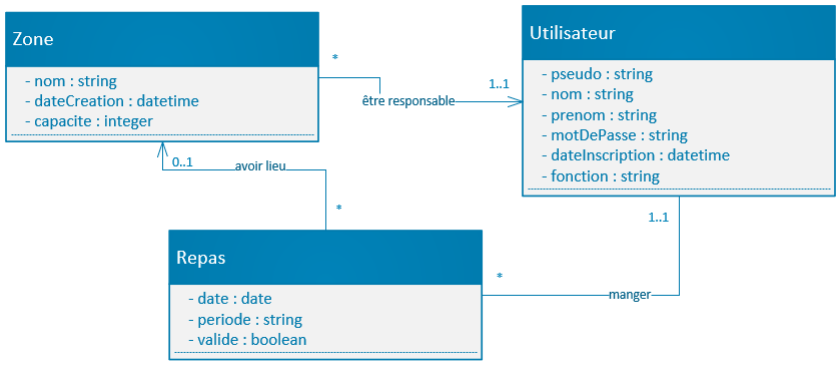
Bloc 2 - SLAM	**Activité n°3 - Partie 6 - ORM Requêtes**
Compétences	Concevoir et développer une solution applicative : - Exploiter les ressources du cadre applicatif (framework) - Identifier, développer, utiliser ou adapter des composants logiciels - Exploiter les fonctionnalités d'un environnement 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 GARBadge (Gestion des Accès de la Restauration) qui nous permet de découvrir l'ORM Doctrine. Dans la suite de l'activité, nous allons créer des requêtes personnalisées.



I. Requêtage avec Doctrine

L'utilisation d'un ORM permet de gagner en rapidité et simplicité par rapport à l'écriture de la couche DAO. L'ORM Doctrine implémente automatiquement les méthodes les plus courantes d'un CRUD (create – read – update - delete). Nous allons aborder les autres moyens pour extraire les données.

1. Modification des données

Les méthodes qui permettent de modifier les données dans la base sont implémentées dans le manager de connexion :

- Create / Update (**C**R**U**D) :

Le manager fournit la méthode ***persist($objet)*** qui permet à la fois la création ou la mise à jour d'un objet en base de données, selon que l'objet a été chargé ou non au préalable par Doctrine.

Objet chargé au préalable depuis la base : ***persist($objet)*** fait un UPDATE dans la base
Objet non chargé au préalable : ***persist($objet)*** fait un INSERT dans la base

- Delete (CRU**D**) :

Le manager fournit la méthode ***remove($objet)*** qui réalise un DELETE dans la base.

Toutes les actions de modification des données doivent être validées par la méthode ***flush()*** pour être réellement écrites en base de données.

2. Méthodes standards d'interrogation des données

Les méthodes Read (CRUD) sont implémentées par des méthodes de l'objet Repository attaché à une Entité métier :

- ***find($id)*** : retourne un objet suivant son id
- ***findAll()*** : retourne une collection de tous les objets
- ***findOneBy($critere, $tri)*** : retourne le premier objet obtenu
- ***findBy($critere, $tri, $limite, $decalage)*** : retourne une collection d'objets

Exemples d'utilisation de findBy() :

```
$entityManager = $doctrine->getManager();
$repository = $entityManager->getRepository(Zone::class);
$liste1 = $repository->findBy(['nom' => 'Collège Jules Ferry']);
$liste2 = $repository->findBy(['nom' => 'Collège Jules Ferry', '...' => '...']);
$liste3 = $repository->findBy(['nom' => 'Collège Jules Ferry'], ['nom' => 'ASC'], 5);
```

Travail à réaliser :

- Utilisez la méthode ***findOneBy()*** dans le contrôleur ZoneController pour faire afficher dans la liste des thèmes avec le dernier message publié de chaque thème.

3. Méthodes personnalisées avec le QueryBuilder

Les méthodes précédentes proviennent de la classe Repository automatiquement créée lors de la conception d'une entité avec ***make:entity***. Il est possible d'ajouter ses propres méthodes pour opérer des requêtes, dans la classe Repository.
Les méthodes ajoutées à la classe Repository doivent retourner les résultats du QueryBuilder. Le QueryBuilder est une classe qui permet la création d'une requête.

Nous allons utiliser les méthodes du QueryBuilder qui sont un mappage des commandes SQL :

- ***where()*** : WHERE
- ***andWhere()*** : AND autre condition
- ***orWhere()*** : OR autre condition
- ***orderBy()*** : ORDER BY
- ***setMaxResults()*** : LIMIT
- ***select()*** pour choisir les données à projeter
- …
-

La méthode ***query()*** permet de transmettre la requête au SGBD.

Le résultat retourné doit utiliser une méthode appliquée à l'objet Query obtenu par :

- ***getResult()*** retourne une collection d'objets
- ***getOneOrNull()*** retourne un seul objet ou null si aucun n'est trouvé

Exemple :

```
    public function findByExampleField($value): array
    {
        return $this->createQueryBuilder('t')
            ->andWhere('t.champExemple = :val')
            ->setParameter('val', $value)
            ->orderBy('t.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult()
        ;
    }
```

Travail à réaliser :

- Ajoutez à ***RepasRepository*** une méthode ***findDernierRepasDeLaZone()*** qui permet de récupérer le dernier repas pour une zone passée en paramètre.
- Utilisez cette méthode dans le contrôleur ***ZoneController*** pour faire afficher dans la liste des thèmes le dernier message publié pour chaque thème.

4. Méthodes personnalisées avec SQL

Il est également possible d'utiliser du SQL.
Exemple :

```
$conn = $this->getEntityManager()->getConnection();
$sql = '
    SELECT * FROM product p
    WHERE p.price > :price
    ORDER BY p.price ASC
    ';
$stmt = $conn->prepare($sql);
$resultSet = $stmt->executeQuery(['price' => $price]);
return $resultSet->fetchAllAssociative();
```

II. Application GARBadge

1. Nombre de repas

Travail à réaliser :

- Dans ***ZoneRepository***, ajoutez la méthode ***zoneNombreDeRepas()*** qui donne le nombre de repas prévus pour une zone donnée en paramètre.
- Modifiez l'affichage de la liste des zones pour afficher sur chaque ligne cette information.

2. Affichage de la liste des repas par thème

On souhaite obtenir une page d'affichage d'une zone à partir de la route ***zone/{id}*** (méthode index()).
Cet affichage visualisera les éléments suivants : la zone, la date du dernier repas prévu et la liste de tous les repas prévus, du plus récent au plus ancien.

Travail à réaliser :

- Modifiez les différentes classes et vues impactées.