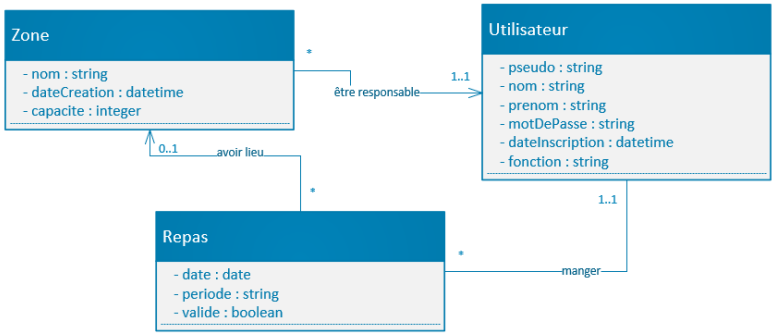
Bloc 2 - SLAM	**Activité n°3 - Partie 5 – ORM Gestion des associations**
Compétences	Concevoir et développer une solution applicative : - Exploiter les ressources du cadre applicatif (framework) - Identifier, développer, utiliser ou adapter des composants logiciels - Exploiter les fonctionnalités d'un 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).

L'association Eclat organise chaque année le Festival d'Aurillac et met en place une restauration pour une partie des personnes présentes : environ 200 salarié·es, 300 artistes et 100 invité·es soit un total de 600 personnes.
Chaque utilisateur a un badge lui permettant d'être identifié mais aussi d'aller manger midi et soir grâce à un code barre imprimé à l'arrière. Tout est réalisé en interne : impression du badge, définition des droits repas.
Actuellement, une application interne existe mais ne peut pas être mise à jour. De nouvelles fonctionnalités doivent également être ajoutées pour une meilleure gestion des repas.
Vous êtes en charge du développement de la nouvelle application GARBadge (Gestion des Accès de la Restauration). Cette application gère les repas proposés aux utilisateurs inscrits.

Une ébauche du diagramme de classes vous est fournie ci-dessous :



I. Création du projet

Travail à réaliser :

- Créez un nouveau projet ***projet_garbadge*** complet (webapp).
- Créez la base de données associée ***base_eclat*** avec Doctrine.

II. Gestion des utilisateurs

1. Spécifications

La classe métier : Classe modèle (entité) ***Utilisateur***

Pour l'ajout d'un utilisateur : Classe formulaire ***UtilisateurType***

Contrôleur d'accès aux fonctionnalités de gestion des utilisateurs : Classe contrôleur ***UtilisateurController***.
La classe contrôleur possèdera les routes et méthodes suivantes :

- ***utilisateur/add*** vers la méthode ***add()***

- ***utilisateur/list*** vers la méthode ***list()***
- ***utilisateur/update/{id}*** vers la méthode ***update()*** pour modifier l'utilisateur identifié par <id>
- ***utilisateur/delete/{id}*** vers la méthode ***delete()*** pour supprimer l'utilisateur identifié par <id>

2. Implémentation

Vous utiliserez les fonctionnalités de Symfony 6.3 pour traiter les différentes parties.

Travail à réaliser :

- Créez l'entité ***Utilisateur*** en réfléchissant bien aux types des données puis mettez à jour la base de données.
- Créez le formulaire de saisie/modification ***UtilisateurType***. Vous devrez sans doute modifier son code pour spécifier les types pour l'affichage.
- Créez la classe contrôleur ***UtilisateurController*** puis dans l'ordre les méthodes ***add()*** et ***list()*** avec les vues correspondantes. Testez.
- Créez la méthode ***delete()*** et testez là.
- Créez la méthode ***update()*** en réutilisant le formulaire (il faut lui transmettre les données existantes).

III. Gestion des zones

1. Spécifications

La classe métier : Classe modèle (entité) ***Zone***

Pour l'ajout d'une zone : Classe formulaire ***ZoneType***

Contrôleur d'accès aux fonctionnalités de gestion des zones : Classe contrôleur ***ZoneController***
La classe contrôleur possèdera les routes et méthodes suivantes :

- ***zone/add/{id}*** vers la méthode ***add()*** pour ajouter une zone gérée par l'utilisateur identifié par <id>
- ***zone/list*** vers la méthode ***list()***

2. Relation entre entités

Les zones de restauration sont proposées par les utilisateurs inscrits et habilités. La modification d'une zone n'est possible que par son responsable : il faut donc que la zone ait connaissance de son responsable.
Reste à traduire l'association entre les classes Zone et Utilisateur.

Travail à réaliser :

- Créez l'entité Zone.

En base de données, l'association entre Zone et Utilisateur est faite à l'aide d'une clé étrangère.
Avec Doctrine, nous allons laisser l'ORM gérer les relations et nous allons nous intéresser uniquement aux notions de POO.

Dans une relation entre deux entités, il y a toujours une entité dite propriétaire, et une dite inverse. L'entité propriétaire est celle qui possède le lien vers l'autre.

Il existe les relations suivantes dans doctrine :

- OneToMany (et son inverse ManyToOne)
- ManyToMany
- OneToOne (rarissime en modélisation)

Pour notre projet, le diagramme de classes indique qu'un utilisateur peut gérer plusieurs zones : nous avons donc une relation de type ManyToOne. Le diagramme UML indique également que c'est Zone qui peut envoyer des messages à l'Utilisateur (sens de navigabilité).

Nous allons donc ajouter une annotation dans la classe Zone :

```
#[ORM\ManyToOne(targetEntity: Utilisateur::class)]
private Utilisateur $responsable;
```

Avec cette annotation, la colonne responsable_id peut être nulle dans la base de données et ce n'est pas logique.
Il faut donc modifier l'annotation en ajoutant une ligne exprimant la contrainte :

```
#[ORM\ManyToOne(targetEntity: Utilisateur::class)]
#[ORM\JoinColumn(nullable: false)]
private Utilisateur $responsable;
```

Travail à réaliser :

- Effectuez l'ajout, générez la migration et vérifiez l'évolution de la base de données.
- Supprimez l'entité *Zone* puis créez-la à nouveau en n'utilisant que l'assistant console. A la question Field type de l'attribut *responsable*, répondre ***relation*** ; cela vous aidera dans la création de la relation.

3. Formulaire d'ajout d'une zone

La difficulté est de pouvoir obtenir la liste des utilisateurs dans ce formulaire, afin d'affecter un responsable (un Utilisateur) à la nouvelle zone créée (même si dans la réalité, l'utilisateur serait connu par sa session). Nous voulons donc obtenir une liste déroulante permettant de sélectionner le pseudo du responsable.
La démarche est :

- Dans le contrôleur, récupérer la liste des utilisateurs avec Doctrine.
- Passer la liste au formulaire lors de sa création, dans les options.
- Dans le formulaire, faire la configuration pour l'option passée (méthode configureOptions()).
- Modifier le champ des responsables pour qu'il devienne un widget de type ChoiceType::class et lui passer les options nécessaires.

Consultez la documentation de Symfony :
https://symfony.com/doc/current/reference/forms/types/choice.html#example-usage

Travail à réaliser :

- Créez le formulaire de saisie/modification ZoneType. Vous devrez sans doute modifier son code.
- Créez la classe contrôleur ZoneController, puis dans l'ordre la méthode add() avec la vue qui va provoquer son affichage.

4. Affichage des zones

La vue devra montrer l'identifiant de la zone, son nom, sa date de création, sa capacité et son utilisateur responsable.

Travail à réaliser :

- Développez la méthode list() et la vue Twig correspondante, afin d'afficher la liste des zones.

IV. Gestion des repas

1. Relations avec les autres entités

La gestion des repas suit la même logique que celle des zones : un repas est associé à un utilisateur. Il va donc falloir traduire cette relation.
De la même manière, il y a une association entre zone et repas : le diagramme UML indique que c'est Repas qui peut envoyer des messages à la Zone (sens de navigabilité).

2. Spécifications

La classe métier : Classe modèle (entité) ***Repas***

Pour l'ajout d'un repas : Classe formulaire ***RepasType***

Contrôleur d'accès aux fonctionnalités de gestion des repas : Classe contrôleur ***RepasController***
La classe contrôleur possèdera les routes et méthodes :

- ***repas/add/{id}/{zone]*** vers la méthode ***add()*** permettant d'ajouter un repas
- les repas seront affichés par utilisateur, ne pas créer de vue pour obtenir la liste des repas.

3. Implémentation

Travail à réaliser :

- Créez l'entité Repas.
- Terminez les différentes classes nécessaires et la vue de création de repas.

Plus d'informations sur les associations sous Doctrine :
https://symfony.com/doc/current/doctrine/associations.html