Création d'un projet

. Installer composer

(vérifier avec composer -v)

symfony new —full mon_projet

cd mon_projet

symfony server:start

Création d'une base de données DOCTRINE

. Se positionner dans votre projet

. Déclarer les accès BDD ainsi que le nom de la base de données dans .env (ici par exemple animaux).

. Création de la base de données

php bin/console doctrine:database:create

. Création d'une table

php bin/console make:entity Animal

création des champs (pas encore dans la BDD)
l'Id est créé automatiquement (pas de setter sur ce champ)

. Migration

Le principe de la migration est de mettre a jour la structure de la BDD si il y a un changement. Ce principe est aussi utilisé la 1ère fois. Un fichier php est généré à chaque migration.

php bin/console make:migration

php bin/console doctrine:migration:migrate

. Fixtures

Les fixtures permettent de remplir les tables pour faire de tests.

composer require orm-fixtures —dev -> creation dossier Fixture

php bin/console make:fixtures AnimalFixtures

modifier AnimalFixtures.php

```
use App\Entity\Animal

dans la fonction load() {
        $a1 = new Animal();
        $a1->setNom('cheval')->setDescription("lorem ipsum…")  ;
```

```
            $manager->persist($a1);
            $a2 = new Animal();
            $a2->setNom('chat')->setDescription("lorem ipsum 2…")  ;
            $manager->persist($a2);

            $a3 = new Animal();
            $a3->setNom('chien')->setDescription("lorem ipsum 2…")  ;
            $manager->persist($a3);

            $manager->flush;
    }
```

php bin/console doctrine:fixtures:load

. Je crée mon contrôleur AnimalController

php bin/console make:controller AnimalController

. Je change la route de Animal en / (dans annotations)

dans les annotations

```
/**
     * @Route("/", name="animal")
     */
```

. Je modifie le template correspondant

```
  <ul>
    {% for animal in animaux %}
        <li>{{animal.nom}} de la couleur {{animal.color}}</li>
    {% endfor %}
    </ul>
```

on peut donner un nom à une route (name dans annotation) et l'utiliser avec path( ) dans notre template menu avec la commande {{ path('lenom') }}

pour accéder aux données dans le répertoire public utiliser asset('chemin/vers/images/ou/css')

Dans la classe **AnimalController**

y insérer la classe Animal

$repository = $this->getDoctrine()->getRepository(Animal::class)

Dans Repository il y a des méthodes déjà définies (exemple find.All( ) et d'autres.)

Après le getRepository

$animaux = $repository->findAll()

Modification de la classe Animal (ajouter le champ poids)

Vider la table Animal avec phpmyadmin

php bin/console make:entity Animal

. ajouter un attribut

. migration

php bin/console make:migration -> génère le fichier de migration

php bin/console doctrine:migration:migrate -> exécute le fichier de migration

modifier le load dans les Fixtures puis -> lancer fixtures

php bin/console doctrine:fixtures:load

Lien vers un animal dans la liste

TWIG

Dans index.html.twig dans un for on a la liste des animaux , on crée un link vers un animal (genre chien

```
        <li><a href="{{ path('afficher_animal',{'id':animal.id})  }}">{{animal.nom}}</a>
de la couleur {{animal.color}}</li>
```

Puis on crée une route dans controller pour 1 animal :

```
 /**
     * @Route("/animal/{id}", name="afficher_animal")
     */
    public function afficherAnimal(AnimalRepository $repository, $id)
    {
        $unanimal = $repository->find($id);
        return $this->render('animal/afficheAnimal.html.twig',[
            "animal" => $unanimal
        ]);
    }
```

Ensuite on copie un twig dans afficheAnimal.html.twig (par exemple) et on le modifie pour afficher les caractéristiques d'un animal.

```
<div class="example-wrapper">
   <p>le {{ animal.nom }} est de couleur {{ animal.color }} de poids
{{ animal.poids}}</p>
</div>
```

Relation 1,1 — 1,n - oneToMany

Création d'une table **espece**

php bin/console make:entity Espece

Créer les champs

libelle

description

animaux —-> on ajoute une autre propriété qui sera animaux (la famille est composée d'**animaux**).

field type enter : (taper ?)
?
(voir propositions des types , il y a relation).

taper:

relation
Animal
oneToMany
il demande ->name inside Animal
espèce

no
no

Vider la table Animal

Effectuer la migration

Vérifier la base de données (structures des tables et schémas de la base)

Regarder les modifs dans Animal et Espece dans Entity

Créer des especes dans AnimalFixtures.php

1) déclarer l'Entity Espece ;

2) Instancier des espèces

```
$e1 = new Espece();
$e1->setLibelle("mammifere")->setDescription("nourris avec du lait") ;
$manager->persist($e1)

$e2 = new Espece();
$e2->setLibelle("poissons")->setDescription("nourris avec du plancton") ;
$manager->persist($e2)
```

ajouter les familles aux objets animaux.

ajouter ->setEspece($espece) a chaque animal ou $espece = $e1 ou $e2 …etc…

php bin/console doctrine:fixtures:load

ajouter {{ animal.famille.libelle }} dans les twigs pour afficher le libelle

Pour supprimer une entité:

Completely delete Entity from symfony 4

You can do it manually by deleting those files:

1. src/Entity/Product.php
2. src/Repository/ProductRepository.php

If you have generated CRUD for your Product entity, you must delete:

1. src/Form/ProductType.php
2. src/Controller/ProductController.php
3. templates/product (the product folder)

If you are in production env, you must run

php bin/console cache:clear

to delete the cache before updating your database schema.
Then run

php bin/console d:s:u --force

php bin/console make:controller EspeceController.

Dans:

annotation : especes

index(EspeceRepository $repo)

$especes = $reposotiry->findAll()

effectuer une boucle dans twig

for (espece in especes…

espèces liste les animaux

for in for

for (animal in espece.animaux voir attribut animaux et getAnimaux
animal.nom

relation 1,n - 1,n avec propriétés

Sur une application normale on crée une clé composée mais en symfony il n'y a pas de clés composée les contraintes d'intégrité fonctionnelles sont gérées par Symfony

```
CREATE TABLE Personnes (
last_name VARCHAR(20) NOT NULL,
first_name VARCHAR(20) NOT NULL,
age int,
address VARCHAR(100),
PRIMARY KEY(last_name, first_name)
);
```

Supposons une personne qui possède plusieurs animaux et on veut connaitre le nombre d'animaux (pour chaque animal)

par exemple une personne possède 2 chiens et 3 serpents

Pour symfony cette relation appelé dispose aura un id avec différentes relations et un nombre

pour cela on va diviser cette relation 1,n - 1,n en 2 relations 1,n - 1,1

1) on crée l'entité personne avec un nom seulement (pas de relation)

php bin/console make:entity Personne

un seul champ à créer : le nom

2) Creation de l'entité dispose avec la propriété nombre

php bin/console make:entity Dispose

animal
relation
Animal
ManyToOne
yes
yes
disposes

personne
relation
Personne
manyToOne
yes
yes
disposes

nombre

integer
null

CIF gérée par symfony

Dans la table dispose s'assurer la contrainte d'intégrité suivante :
on ne peut pas avoir 2 fois toto et serpent et un nombre , on s'assure de cela dans l'entité Dispose.

Dans l'entité Dispose

Dans l'annotation de la classe Dispose (avant class)

use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity

Dans l'annotation

```
/**
 * /**
 * @ORM\Entity(repositoryClass=AnimalRepository::class)
 * @UniqueEntity(fields={"personne","animal"})
 */
```

migrate

fixture
créer des personnes

dans AnimalFixtures

```
$p1 = newPersonne()
$p1->setNom("Noam");
$manager->persiste($p1);
```

idem pour p2, p3 …etc…

puis à la fin de AnimalFixtures

```
$d1 = new Dispose();
$d1->setPersonne($p1)
        ->setAnimal($a1)
        ->setNombre(10);

$manager->persiste($d1);
```

puis

In animal.twig dans la liste des animaux

```
{% for dispose in animal.disposes %}
        {{dispose.personne.nom}}
{% endfor %}
```

controllerPersonne -> PersonRepository
twig
for personne in personnes
personne.nom

```
{% for dispose in animal.disposes %}
        {{dispose.personne.nom}}
          for dispose in person.disposes
                  dispose.animal.nom
                  {{  asset("images/ ~ dispose.animal.image }}
          endfor
{% endfor %}
```

Affichage dun personne

@Route("personne/{id}", name=afficher_personne

Avec paramConverter l'injection de dépendance symfony va automatiquement instance l'objet dont l'id est {id} et sera immédiatement disponible dans twig

Requête spécifique avec création d’une requête spécifique dans AnimalRepository

```
class AnimalRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Animal::class);
    }

    public function getAnimauxLegers($poids){
        return $this->createQueryBuilder('a')
        ->andWhere('a.poids < :val')
        ->setParameter('val', $poids)
        ->getQuery()
        ->getResult()
        ;
    }
}
```

Sécurité contre l’injection SQL

Administration des animaux (formulaire)

php bin/console make:controller admin\AdminAnimalController

création du **controller** et du **template**

Création d'une forme liée à une table.

php bin/console make:form

name of form class
AnimalType

name of Entity (entité sur laquelle on fait la liaison)
Animal

La forme est créée dans le répertoire Form

dans AdminAnimalController ajouter

```
/**
*  @Route ("/admin/animal/{id}…
*
**/

public function Ajoutmodification(Animal $animal = null , Request $request , EntityManager
$entityManager)
{

        if (!$aliment) {
                $aliment = new Aliment();
        }
        // récupérer le formulaire
        $form = $this->createForm(AnimalType::class);
        render avec paramètres [ "animal" => $aliment,
                                 "monform" => $form->createView(  )];
}
```

Request n'est pas déclaré : faire Alt-Ctrl-i sur le type Request pour choisir le 2ème composant (httpFoundation)
On récupère la requête

EntityManager : permet d'inscrire les données en BDD (injection de dépendance)

-> Dans la vue insérer

```
{{ form_start(monform) }}
        <div>{{ form_row(form.nom, {'attr': {'class':'uneClasse'},"label":"Nom de l'aliment" }) }}</
div>
        {{ form_widget(form) }}
{{ form_end(monform) }}
```

voir la doc pour **form_row()** pour les options

tester

localhost:8000/admin/animal/1

Ajouter un <input> de type submit à la vue dans form (avant le form_end)

puis dans la fonction modification du controller avant $this->render

```
// récupère les données du form
$form->handleRequest($request);
if($form->isSubmitted() && $form->isValid()){
        // persiste les données dans la base
        $entityManager->persist($animal);
        $entityManager->flush();
        return t$this->redirectToRoute("admin_animals");  // admin_animals nom de la route (name)
}
```

AJOUT

```
// Maintenant je désire ajouter un animal.
// Je peux garder la même méthode modification mais avec 2 routes différentes.
// la seule distinction sera de créer un objet Animal vide si il y a une création.

/**
* @Route("/admin/animal/creation", name="admin_animal_creation")
* @Route("/admin/animal/{id}", name="admin_animal_modification")
*/
```

dans la méthode

Si c'est une création l'aliment n'existe pas (argument de la fonction) il faut donc créer l'objet aliment

```
if (!animal) {
        // créer l'animal

}
```

Faire une méthode suppression avec **EntityManager->remove()** à la place de EntityManager->persist() vu dans la modification

```
public function suppression(Animal $animal, Request $request, ObjectManager $objectManager)
{
… à compléter
}
```

```
 <a href="{{path('admin_aliment_modification',{'id' : aliment.id})}}">Modifier</a>
            <form method="POST" style="display:inline-block"
action="{{path('admin_aliment_suppression', {'id' : aliment.id})}}">
                <input type="hidden" name="_method" value="delete">
                <input type="hidden" name="_token" value="{{csrf_token('SUP' ~ aliment.id)}}">
                <input type="submit" class="btn btn-danger" value="supprimer" onsubmit="return
confirm('Confirmer la suppression ?')">
```

</form>

Relation 1,n — 1,n (sans attributs dans la table relation créée)

Rajouter une entité **Continent**

php bin/console make:entity Continent

ajouter un champ **Libelle** (id gérée automatiquement)
ajouter un champ
animaux
Puis cette fois-ci choisir la relation **ManyToMany**

modifier la méthode load() dans AnimalFixtures

```
$continent1=newContinent() ;
$continent1->setLibelle("Europe");
$manager->persist($continent1);

$continent2=newContinent() ;
$continent2->setLibelle("Afrique");
$manager->persist($continent2);


$a1 = new Animal( );
$a1     -> setNom("Chien"
        -> setDescription("efref…")
        -> ….()
        ->addContinent($continent1)
        ->addContinent($continent2)
        ->addContinent($continent3);
```

twig:
for continent in animal.continents
continant.libelle

Creation controllerContinent

twig
for continent in continents
continent.libelle
for animal in continent.animaux
animal.nom
animal.espece.libelle