Projet “CatFinder” - Rapport

Pako MATHIEU

Stefan MENARD

Mohamed El Mokhtar BOUNIAR

Sommaire

Contexte

Le projet qui nous a été demandé de mettre en place, était un site web, le choix du site était libre. Nous nous sommes dirigé vers un site sur le domaine des chats, son nom est **“CatFinder”**. Il a pour but de rassembler tous les passionnés de félin. Pour cela, nous avons décidé de créer plusieurs parties permettant de plaire à un plus grand nombre de personnes.

La première partie “Statique” a pour but d’attirer des personnes voulant acquérir de nouvelles connaissances, elle se compose de :

- L’histoire des chats
- la liste des différentes races

La seconde partie “Annonces”, va permettre le don ou la vente de ces félins. L’utilité principal est de limiter l’abandon des félins, elle se constitue :

- Affichage de toutes les annonces (Possibilité de filtrer les recherches)
- Création d’une annonce

La troisième partie “E-Commerce”, a pour objectif la vente d’objet écologique en tout genre pour le plus grand plaisir de nos félins, elle est caractérisée par :

- l’affichage de tous les produits mis à dispositions
- La gestion du panier

Acteurs

Les principaux acteurs qui vont utiliser notre site web sont :

- Le visiteur
- Visiteur non connecté
- Client Membre

Nous détaillerons par la suite les fonctionnalités auxquelles chacun à accès.

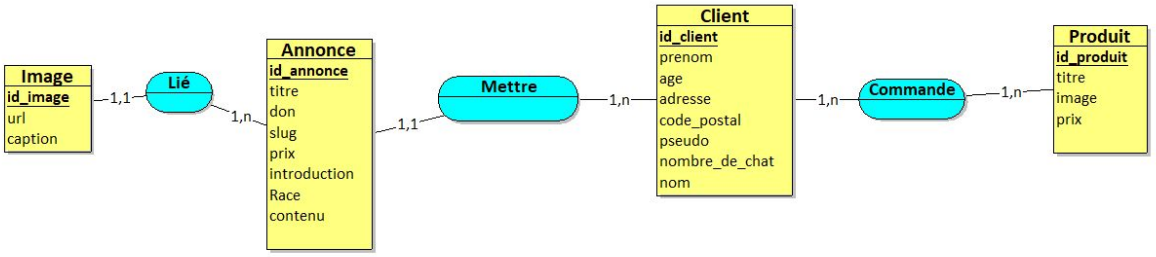
Composition du site

Le site se compose en 2 principales parties :

1. Partie vitrine (acteurs concernés : Visiteur, Visiteur non connecté et Client membre)
- La page d'accueil présentant les créateurs et la composition du site.
- Une page présentant l’histoire des chats
- Listing des différentes races de chats

2. Partie Vente (acteurs concernés : Client membre)
- E-Commerce
- Annonces

Modèle entité relation



Un Utilisateur(User) est identifié par son identifiant. Il a comme attribut son nom, son prénom, son email, son avatar, son mot de passe, introduction, description et son slug. Il peut mettre des annonces et choisir des produits à mettre dans son panier. Un utilisateur(auteur) est inclue dans une Annonce.

Une Annonce est identifiée par son identifiant. Elle a comme attributs un titre, son adresse (slug), un prix, une introduction, un contenu, une image de couverture, un attribut don indiquant si c'est un don ou une vente et la race du chat. Une annonce est inclue dans une Image car elle est composé de plusieurs images.

Une Image est identifiée par un identifiant. Elle a comme attributs son url et sa caption.

Une race de chat est identifiée par un identifiant. Elle a comme attributs un libellé et une url (Elle va être utilisé dans la page statique de listing des chats).

Un produit est identifié par un identifiant. Il a comme attribut son titre, l'url de son image et son prix.

Schéma relationnel

Utilisateur (**id_utilisateur,** prenom, nom, email, introduction, description, hash, avatar, slug, nom)

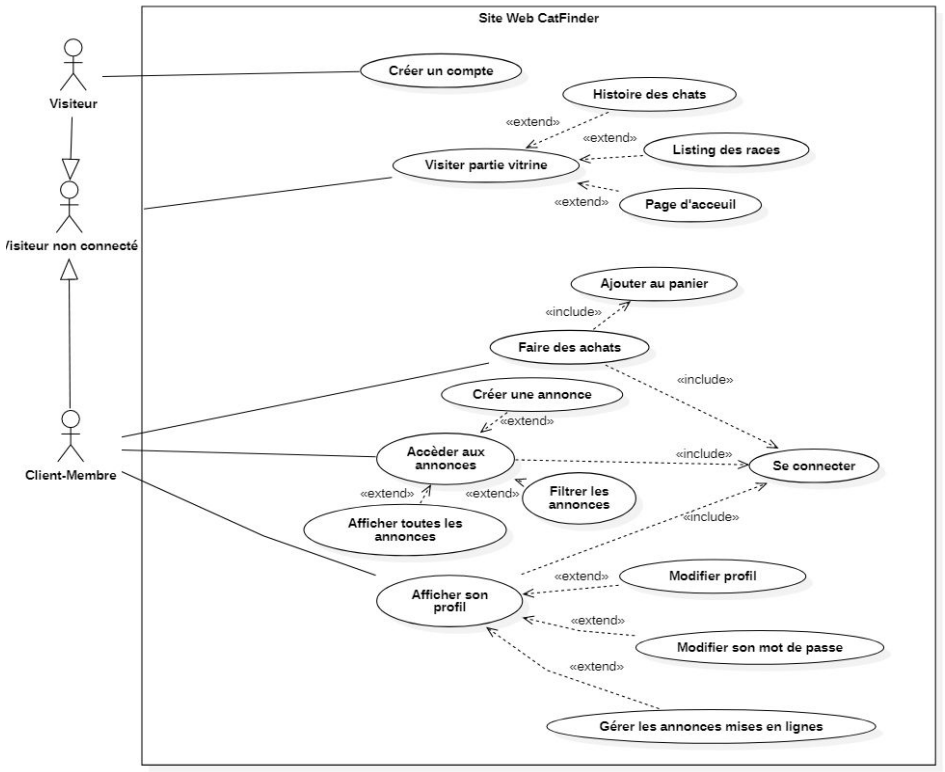
Produit (**id_produit,** titre, image, prix)

race_chat (**id_race**, libelle, url_image)

Annonce (**id_annonce**, titre, don, adresse(slug), prix, introduction, Race, contenu, #id_utilisateur)

Image (**id_image**, url, caption, #id_annonce)

Diagramme des cas d'utilisations



Un "Visiteur" peut créer un compte. Il hérite des cas d'utilisation d'un "Visiteur non connecté".

Un "Visiteur non connecté" peut visiter la partie vitrine du site.

Un "Client-Membre" pourra faire des achats. Il pourra aussi accéder aux annonces postées par les différents utilisateurs, où il pourra filtrer les recherches d'annonces, et créer des annonces. Il pourra aussi accéder à son compte, où il pourra modifier les informations de son profil, modifier son mot de passe et gérer les annonces qu'il a posté sur le site. Toutes ces actions incluent le fait que le "Client-membre" se connecte au site. Cet utilisateur hérite lui aussi du "Visiteur non connecté".

Partie code des points principaux du site

Dans chaque partie, nous allons voir le code permettant de réaliser les principales fonctionnalités.

Partie Utilisateur :

1- Inscription/Connexion :

Pour que le visiteur devienne utilisateur/client, il faut qu'il s'inscrive et pour ça il va falloir créer un formulaire, en symfony il y a la classe "BuildForm" facilitant la création de formulaires.

Partie Contrôleur : (CompteController.php)

```php
/**
 * Permet l'enregistrement des utilisateurs
 * @Route("/inscription", name="inscription_user")
 * @return Response
 */
public function enregistrement(Request $request, UserPasswordEncoderInterface $encoder){

        $user= new User(); //création d'un objet de la classe user

        $form=$this->createForm(EnregistrementType::class, $user); //Creation du fo
                                                                   //On entre dans l'o

        $form->handleRequest($request); //on lui passe la requete HTTP, analyse de
        //Il va relier toutes les informations du formulaires a notre variable $ann

        if ($form->isSubmitted() && $form->isValid()){

                $hash = $encoder->encodePassword($user, $user->getHash());
                $user->setHash($hash);

                $manager = $this -> getDoctrine()->getManager(); //la fonction

                $manager->persist($user); //persiste dans le temps les données

                $manager->flush(); //envoi les données qui était entrain de per

                $this->addFlash( //affichage de la notif indiquant
                    'success', //success car boostrap rendra vers
                    "Votre compte a bien été créé ! Vous pouvez ma
                );

                return $this->redirectToRoute('login');

        }
        return $this->render('compte/enregistrement_user.html.twig', [
        'form'=> $form->createView() //on renvoie une composante du fo
        ]);
}
```

Partie Formulaire : (EnregistrementType.php)

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder //indication des compositions de chaque champ de notre formulaire d'inscription
        ->add('prenom', TextType::class, $this->configurationDesChamps('Prénom', 'Entrez votre prénom ..'))
        ->add('nom',  TextType::class, $this->configurationDesChamps('Nom', 'Entrez votre nom ..'))
        ->add('email', EmailType::class, $this->configurationDesChamps('Email','Entrez votre adresse mail..'))
        ->add('avatar',  UrlType::class, $this->configurationDesChamps('Url Avatar', 'Entrez l\'URL de votre avatar
        ->add('hash',  PasswordType::class, $this->configurationDesChamps('Mot de passe', 'Entrez votre mot de pass
        ->add('passwordConfirm', PasswordType::class, $this->configurationDesChamps("Confirmation du mot de passe",
        ->add('introduction', TextType::class, $this->configurationDesChamps('Introduction', 'Présentez vous en que
        ->add('description', TextareaType::class, $this->configurationDesChamps('Description', 'Présentez vous en d
        //->add('slug') on affiche pas le slug il est automatiquement définit
    ;
}
```

Partie Template : (enregistrement_user.html.twig)

```
{% extends 'base.html.twig' %}

    {% block body %}

<body>

    <div class="container" style = "text-align:center">
            <h1>Inscription sur CatFinder</h1>
    </div>
  <div class="container">

    {{ form_start(form) }} <!--ici form correspond a l'ensemble des champs du formulaire

    {{ form_widget(form) }}

  <div class="container boutton-submit" >
    <button type="submit" class="btn btn-primary" style = >Création du compte</button>

  </div>


    {{ form_end(form) }}

</body>
{% endblock %}
```

Une fois l'utilisateur inscrit, il doit se connecter et pour ça nous n'allons pas utiliser la classe "BuildForm" car il y a la fonction form_login qui permet de faire le login plus simplement.

Controleur (CompteController.php) :

```
/**
 * Permet d'afficher et de gérer le formulaire de connexion
 * @Route("/login", name="login")
 *
 * @return Response
 */
public function login(AuthenticationUtils $authenticationUtils):Response
{
    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();

    //Va permettre dans twig de garder écrit dans l'input mail le dernie
    $lastMail = $authenticationUtils->getLastUsername();

   return $this->render('compte/login.html.twig',[
   'error' => $error,
   'lastMail'=>$lastMail
   ]);
}
```

Template : (voir fichier login.html.twig)

Le template ne va pas appeler le formulaire créé avec la classe "BuildForm" car nous allons utiliser la fonction "form_login" qui utilise seulement les formulaires HTML, donc création d'un formulaire basique HTML.

Rajouts obligatoires :

a- Pour que la partie connexion fonctionne il faut implémenter l'entité "User" des classes UserInterface et Serializable (redéfinition de leurs méthodes) :

```
class User implements UserInterface,\Serializable
```

```
public function getRoles(){
    return ['ROLE_USER'];
}

//On retourne null car on est pas conc
public function getSalt(){
    return null;
}

//L'username qu'on utilise dans notre
public function getUsername()
{
    return $this->getEmail();
}

//On retourne le password dans notre c
public function getPassword(){
    return $this->getHash();
}

//on retourne rien car on est pas conc
public function eraseCredentials(){}
```

```
public function serialize(){
    return serialize([
        $this->id,
        $this->prenom,
        $this->nom,
        $this->email,
        $this->avatar,
        $this->hash,
        $this->introduction,
        $this->description,
        $this->slug,
        $this->annonces
    ]);
}
```

```
public function unserialize($serialized)
{
    list(
            $this->id,
            $this->prenom,
            $this->nom,
            $this->email,
            $this->avatar,
            $this->hash,
            $this->introduction,
            $this->description,
            $this->slug,
            $this->annonces
        )=unserialize($serialized, ['allowed classes'=> false]);
}
```

b- Et il faut modifier plusieurs choses dans le fichier security.yaml :

Encodage mot de passe :

```
encoders:
    App\Entity\User:
        algorithm: auto
        cost: 12 #Le cos
```

Création d'un fournisseur: (Permet de faire des vérifications sur l'email de l'utilisateur)

```
in_database: #Nouveau provider,
    entity: #ce provider utilis
        class: App\Entity\User
        property: email  #Indic
```

Ajout de l'utilisation du form_login dans le firewall :

```
provider: in_database #O

form_login: #Configurati
        login_path: login
        check_path: login
```

On indique qu'on va utiliser le fournisseur créé "in_databse" et on configure la fonction form_login, en lui indiquant de se rediriger vers le controller du nom de "login" quand il tombe sur ce formulaire.

2- Modifier profil

Pour la modification du profil, il faut aussi créer un formulaire sauf qu'il faut rajouter une ligne dans le controlleur (CompteController.php) permettant de récupérer les informations de l'utilisateur actuellement connecté.

```
$user = $this->getUser(); //Grace a la fonction getUser on récupere l'utilisateur connecté actuellement
```

3- Modifier mot de passe

Pour la modification du mot de passe, il faut aussi créer un formulaire sauf qu'il faut rajouter plusieurs ligne dans le controlleur (CompteController.php). Nous allons utiliser la classe UserPasswordEncoderInterface pour crypter les mots de passes.

Injection de dépendance de la classe UserPasswordEncoderInterface :

```
public function EditerPassword(Request $request, UserPasswordEncoderInterface $encoder)
```

Ajout des lignes qui vont permettre la vérification de l'ancien mot de passe s'il a bien été tapé:

```
$passwordUpdate = new PasswordUpdate();
```

```
//1.Vérifier que l'ancien mdp du formulaire soit le meme password de l'utilisat
if (!password_verify($passwordUpdate->getAncienPassword(), $user->getHash())){
 //gére l'erreur

}
else{ //Sinon si les passwords comparés sont les mêmes
    $newPassword = $passwordUpdate->getNouveauPassword(); //ici on choppe notre
    $hash= $encoder->encodePassword($user, $newPassword); //Ensuite il faut enc

     $user->setHash($hash); //on met a jour le champ hash de l'utilisateur dans
     $manager = $this -> getDoctrine()->getManager(); //la fonction ne fonctior

     $manager->persist($user); //persiste dans le temps les données

     $manager->flush(); //envoi les données qui était entrain de persister dans

     $this->addFlash( //affichage de la notif indiquant la réussite
         'success', //success car boostrap rendra vers la notif, on indique le
         "Votre mot de passe a bien été modifié ! :)" //indication du message
     );
```

Partie E Commerce :

Dans la partie E Commerce les utilisateurs de notre site pourront consulter le magasin du site, ils trouveront des produits de plusieurs catégorie qu'ils pourront les ajouter dans leurs paniers et effectuer les commandes.

L'implémentation d'une entité "Product" était nécessaire pour réaliser ces fonctionnalités. Le fichier "Product.php" présent dans le dossier Entity représente la class "Product".

1- Affichage des différents produits :

Pour afficher les produits déjà enregistrés dans la base de données, nous avons utilisé un contrôleur "ProductController" contenant une seule fonction public "index" qui permet d'enregistrer les produits dans un tableau depuis la BDD en utilisant la fonction "findAll()" proposée par le "ProductRepository", afin de transmettre ce tableau comme paramètre vers le fichier twig 'product/index.html.twig'. Voici le code qui permet l'affichage de la liste de produits :

```php
namespace App\Controller;

use App\Repository\ProductRepository;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ProductController extends AbstractController
{
    /**
     * @Route("/magasin", name="all_product")
     */
    public function index(ProductRepository $productRepository)
    {
        return $this->render('product/index.html.twig', [
            'controller_name' => 'ProductController',
            'products' => $productRepository->findAll()

        ]);
    }
}
```

2-Affichage du panier

L'utilisateur pourra aussi à tout moment consulter son panier pour gérer les produits commandés ainsi que leurs quantités, il pourra alors ajouter et supprimer des produits de son panier.

Pour implémenter ces fonctionnalitées nous avons utilisés un nouveau contrôleur "CartController" qui contient trois fonctions public, chacune représente une fonctionnalitées.

2.1-Affichage du panier :

Pour l'affichage du panier, nous avons utilisés la notion de "SessionInterface" : une classe fournie par Symfony qui permet la manipulation de la session. Voici le code d'implémentation de la fonction "index()":

```php
/**
 * @Route("/panier", name="cart_index")
 */
public function index(SessionInterface $session, ProductRepository $productRepository)
{   $panier = $session->get('panier', []);
    $panierWithData = [];
    foreach($panier as $id => $quantity)
    {$panierWithData[] = [
            'product' => $productRepository->find($id),
            'quantity' => $quantity
        ];
    }
    $total = 0;
    foreach($panierWithData as $item){
        $totalItem = $item['product']->getPrice() * $item['quantity'];
        $total += $totalItem;
    }
    return $this->render('cart/index.html.twig', [
        'items' =>$panierWithData,
        'total' => $total
    ]);
}
```

2.2-Ajouter des produits :

L'ajout d'un produit se fait à travers une deuxième fonction "add()", qui récupère la session (l'état du panier) et ajoute au tableau "$panier" le nouveau produit sélectionné par l'utilisateur. Voici le code d'implémentation de la fonction "add()":

```php
/**
 * Ajouter produit dans e panier
 * @Route("/panier/add/{id}", name = "cart_add")
 *
 * @return void
 */
public function add($id, SessionInterface $session){

    $panier = $session->get('panier', []);

    if(!empty($panier[$id])){
        $panier[$id]++;
    }
    else{
        $panier[$id] = 1;
    }

    $session->set('panier', $panier);

    return $this->redirectToRoute("cart_index");
}
```

2.3-Suppression des produit :

La suppression d'un produit se fait à travers une troisième fonction "remove()", qui récupère la session (l'état du panier) et supprime du tableau "$panier" le produit sélectionné par l'utilisateur. Voici le code d'implémentation de la fonction "remove()":

```php
/**
 * @Route("/panier/remove/{id}", name="cart_remove")
 */
public function remove ($id, SessionInterface $session){
    $panier = $session->get('panier', []);

    if(!empty($panier[$id])){
        unset($panier[$id]);
    }

    $session->set('panier', $panier);

    return $this->redirectToRoute("cart_index");

}
```

Partie Annonce :

1-Créer annonce

Pour créer des annonces, il faut créer un formulaire comme pour l'inscription, nous allons pour cela utiliser la classe "BuildForm".

Partie controller : (AnnonceController.php)

```php
/**
 * On met cette fonction avant car meme chemin que pour l'affichage
 *symfony va prendre la premiere et voir que ce n'est pas un slug e
   * @Route("/annonce/newAd", name="formulaire_creer_annonce")
   *
   * @return Response()
   */                                                    //IMPORTANT\\
  public function CreerAnnonce(Request $request) //C'est ici qu'on
   {
       $annonce  = new Annonce(); //creation d'un objet de type

       $form = $this->createForm(AnnonceType :: class,$annonce);

       $form->handleRequest($request); //on lui passe la requete
       //Il va relier toutes les informations du formulaires a no

       if ($form->isSubmitted() && $form->isValid()){

         $manager = $this -> getDoctrine()->getManager(); //la fo

         //Pour persister les nouvelles images ajoutés pour les a
         foreach($annonce->getImagess() as $image){

             $image->setAnnonce($annonce);
             $manager->persist($image);
```

```
            }

            //IMPORTANT\\
            $annonce->setAuteur($this->getUser()); // Cette ligne va permettre de recuperer dans

            $manager->persist($annonce); //persiste dans le temps les données

            $manager->flush(); //envoi les données qui était entrain de persister dans le temps


            $this->addFlash( //affichage de la notif indiquant la réussite
                'success', //success car boostrap rendra vers la notif, on indique le type de fi
                "L'annonce <strong>{$annonce->getTitre()}</strong> a bien été enregistrée ! :)"
            );

            return $this->redirectToRoute('afficherAnnonce',[
                    'slug'=>$annonce->getAdresse()

            ]);

            }

        return $this->render('formulaire/creerAnnonce.html.twig', [
            'form'=> $form->createView()//permet de récupérer juste l'affichage du formulaire vi
        ]);
    }
```

Partie Formulaire : (AnnonceType.php)

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('titre', TextType ::class, $this->configurationDesChamps('Titre', 'Indiquer le titre de votre annon
        //->add('adresse', TextType ::class, $this->configurationDesChamps('Adresse', 'adresse URL (récupération
        ->add('introduction', TextType :: class,$this->configurationDesChamps('Introduction', 'Introduction de vo
        ->add('contenu', TextareaType :: class, $this->configurationDesChamps('Contenu', 'Contenu de votre annonc
        ->add('race', TextType :: class,$this->configurationDesChamps('Race', 'Indiquer la race de votre chat') )
        ->add('coverImage', UrlType :: class,$this->configurationDesChamps('Image', 'Url de l\'Image principale d
        ->add('prix', MoneyType :: class, $this->configurationDesChamps('Prix', 'Indiquer le prix', ['required' =
        ->add('don',ChoiceType::class, [
            'label'=>'Type d\'annonce',
            'choices' => ['Vente' =>'Vente',
                          'Don'=>'Don',
                          ],
        ])
        ->add(
            'imagess',
            CollectionType::class,  //collection class permet de rajouter plusieurs champs
            [
                'entry_type'=>ImageType :: class, //le type de champs correspond au formulaire ImageType qu'on ré
                'allow_add'=> true
            ]
        );
}
```

Partie Template : (voir fichier creerAnnonce.html.twig)

2-Affichage de toutes les annonces :

Pour afficher les annonces, il faut récupérer chaque annonce inscrite dans la BDD, pour cela, dans le contrôleur, nous allons utiliser une fonction par défaut de la classe repository qui est le “findAll”.

Partie controller : (AnnonceController.php)

```php
/** Permet d'afficher toute les annonces
 * @Route("/annonce", name="annonce")
 */
public function MainAnnonce(AnnonceRepository $repo) //injection
{
    //$repo=$this->getDoctrine()->getRepository(Annonce::class);

    $annonces=$repo->findAll();


    return $this->render('annonce/mainAnnonce.html.twig', [
        'annonces' => $annonces,
    ]);
}
```

Partie template : (voir fichier mainAnnonce.html.twig)

3-Filtrage des annonces

Pour donner une flexibilité dans les recherches utilisateurs, il faut leur donner le choix de filtrer les recherches. Nous avons pris la décision de créer le filtre de recherche dans une page annexe.

Utilisation une nouvelle fois d'un formulaire avec la classe "BuildForm".

Partie Controller : (AnnonceController.php)

```php
    /**
     * @Route("/annonce/recherche", name = "recherche_annonce")
     *
     * Barre de recherche filtrant par rapport au type de vente et le prix si "Vente"
     */
public function RechercherAnnonce(Request $request, AnnonceRepository $annonceRepository){ //
                                                                                            // in
    $searchAnnonces = []; // déclaration d'un tableau vide permettant l'ajout des données réc
    $FormRecherche = $this->createForm(RechercherAnnonceType::class);

    if($FormRecherche->handleRequest($request)->isSubmitted() && $FormRecherche->isValid()){
        $critere= $FormRecherche->getData(); //la variable critere récupère les informations

        $searchAnnonces = $annonceRepository->RechercherAnnonce($critere); //la variable sear
                                                                           //On passe en paramet
        //seachAnnonces va récupérer toutes les annonces selectionnées via le select du repos
        //dd($searchAnnonce);
    }

    return $this->render("recherche/rechercheAnnonce.html.twig", [
        'form_recherche' => $FormRecherche->createView(), //on partage le formulaire pour qu
        'searchAnnonces' => $searchAnnonces, //on partage la variable pour qu'elle soit utili
        ]);
}
```

Partie Formulaire : (RechercheAnnonceType.php)

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
    ->add('don',ChoiceType::class, [
        'label'=>'Type d\'annonce',
        'choices' => ['Vente' =>'Vente',
                      'Don'=>'Don',
                      ]])

    ->add('Prix_mini', ChoiceType::class, [ //menu déroulant avec différen
        'label'=> 'Prix minimum',
        'choices'=> array_combine(AppFixtures::PRICE, AppFixtures::PRICE)

    ])
    ->add('Prix_maximum', ChoiceType::class, [//menu déroulant avec différ
        'label'=> 'Prix Maximum',
        'choices'=> array_combine(AppFixtures::PRICE, AppFixtures::PRICE)

    ])
    ->add('recherche', SubmitType::class)
        ;
}
```

Partie Repository : (AnnonceRepository.php)

Nous allons utiliser le fichier AnnonceRepository car nous avons besoin de faire une requête personnalisée à la BDD.

```php
public function RechercherAnnonce($critere) //une requete
{
    return $this->createQueryBuilder('annonce') //table ar
      // ->leftJoin('a.race', 'race') chercher comment fa
       //->Where('a.race = :race')
       //->setParameter('race', $critere['race'])
      ->Where('annonce.don = :don') // colonne don dans l
      ->setParameter('don', $critere['don'])
      ->andWhere(
      new Expr\Orx([ //permet de faire une double conditi
       "annonce.prix IS NULL", //on demande d'afficher  l
       "annonce.prix > :Prix_mini", //on demade d'affiche
    ])
)
      ->setParameter('Prix_mini', $critere['Prix_mini'])

      ->andWhere(
       new Expr\Orx([//permet de faire une double conditi
        "annonce.prix IS NULL",//on demande d'afficher  l
        "annonce.prix < :Prix_maximum",//on demade d'affi
    ])
)
```

```php
      ->setParameter('Prix_maximum', $critere['Prix_maximum'])

       //->orderBy('a.id', 'ASC')
       //->setMaxResults(10)
       ->getQuery()
       ->getResult()
    ;
}
```

Partie Template (voir fichier rechercheAnnonce.html.twig)

Conclusion

Pour conclure, avec plus de temps, nous aurions pu rajouter une partie forum pour la création de lien entre nos utilisateurs passionnés de félins. Puis améliorer les différentes parties déjà mises en places avec la création de page de paiement pour les produits et de réservation pour les annonce ou bien la création d'une partie administration permettant de gérer les utilisateurs du site, ainsi que les publications que ce soit dans les annonces ou dans la partie e-commerce.

Ce projet nous a permis de nous améliorer dans la conception d'un projet et surtout dans la partie développement web ou pour certains c'était de bonne révision et pour d'autre un apprentissage de zéro du fonctionnement du framework "Symfony".