SYMFONY 5 commerce part-3

FRAMEWORK PHP

EasyAdmin

- EasyAdmin est un outil puissant que nous allons utilisé afin de gérer notre backOffice.
- Cherchez ce que c'est et installez la dépendance dans votre projet e-commerce.

EasyAdmin

- Maintenant que vous avez installé Easy Admin à l'aide de la commande : composer require easycorp/easyadmin-bundle
- Vous allez créer un « dashboard » comme ceci : symfony console make:admin:dashboard

EasyAdmin

- Vous pouvez décommenter dans le dashboard controller cette ligne et la modifier pour qu'elle soit comme la mienne :

```
yield MenuItem::linkToCrud('Utilisateur', 'fas fa-list', User::class);
```

- Pensez à importer l'entité User ☺
- Si vous essayez la route /admin vous devriez avoir une erreur.
- Il vous manque le crudController

symfony console make:admin:crud

Category

- Créez une nouvelle entité category avec un attribut name (string)
- Faites la migration

Mapper avec easyAdmin

- Créez le crud controller, comme on l'a fait pour user mais pour catégory
- Rajoutez la ligne pour faire fonctionner les catégories dans le DashboardController (copier celle des users)
- Rajoutez une ou plusieurs catégories pour voir si ça marche

Product

Créez l'entité Product avec les champs suivant :

- Name string 255 not null
- Slug string 255 not null
- Image string 255 not null
- Subtitle string 255 not null
- Description text not null
- Price float not null
- Id_category relation not null

Pour l'id catégory, vous la liez avec l'entité catégory et vous choisissez la relation (à votre avis laquelle ?)

```
Do you want to add a new property to Category so that you can access/update Product objects from it - e.g. $category->getProducts()? (yes/no) [yes]
> yes New field name inside Category [products]: > products Do you want to automatically delete orphaned App\Entity\Product objects (orphanRemoval)? (yes/no) [no]: > no
```

Product

Mappez les produits sur easyAdmin, vous connaissez la démarche ☺

Product Ajuster les champs

Dans notre ProductCrudController il y a cette méthode qui est commenté, elle sert à paramétrer nos inputs.

```
/*
    public function configureFields(string $pag
eName): iterable
    {
        return [
            IdField::new('id'),
            TextField::new('title'),
            TextEditorField::new('description')
,
        ];
    }
    */
```

Product Ajuster les champs TP

Je vous laisse chercher les différents FIELDS à retourner dans le tableau de cette fonction, voici un exemple :

```
TextField::new('name',"Nom"),
```

Je veux que les champs soient positionnés comme ceci :

- Nom (TexteField)
- Slug(SlugField)
- Sous-titre(texteField)
- Image(ImageField)
- Description(textareaField)
- Prix(moneyField)
- Category(AssociationField)

Pensez à importer les components avec Use, bon courage.

Product La vue pour les utilisateurs

Vous connaissez la rengaine, créez un ProductController

Product Le controller

Dans le controller de product :

- Créez un attribut privé avec l'entityManager stocké dedans (comme pour modifyUser)
- Récupérez tout les produits grâce à cette commande :

```
$products = $this->entityManager->getRepository(Product::class)->findAll();
```

- Faites passez la variable products à la vue twig.

Product La vue twig TP

Comme des grands, vous allez afficher tout les produits dans une vue.

Utilisez bootstrap et un peu de css pour avoir un affichage propre avec:

- Titre du produit
- Sous titre
- Prix

Product Controller (un article)

Pour Afficher un article nous allons mettre à jour notre ProductController avec une nouvelle fonction show

Je vais vous détailler ce que nous mettons dans cette fonction.

```
/**
 * @Route("/produit/{slug}", name="product")
 */
public function show($slug)
{
    $product = $this->entityManager->getRepository(Product::class)->findOneBy(["slug"=>$slug]);
    if(!$product){
        return $this->redirectToRoute("products");
    }
    return $this->render('product/show.html.twig', [
        "product"=>$product
    ]);
}
```

Product Controller (un article)

- Créez la vue pour un seul article. (show.html.twig) dans le dossier templates product
- Modifier le template de tous les articles afin de faire un lien vers l'article en question en cliquant sur l'image.
- Ci-dessous, la syntaxe pour faire un lien en utilisant le slug.

```
<a href={{path('product',{slug:product.slug})}}>
```

Products, filtrer

Pour pouvoir filtrer sur les articles, nous allons rajouter cette fonctionnalité dans la page de tous les articles.

À l'aide de bootstrap coupez la page comme ceci :

Class Search

Nous allons créer une classe (et non pas une entité) à la main :

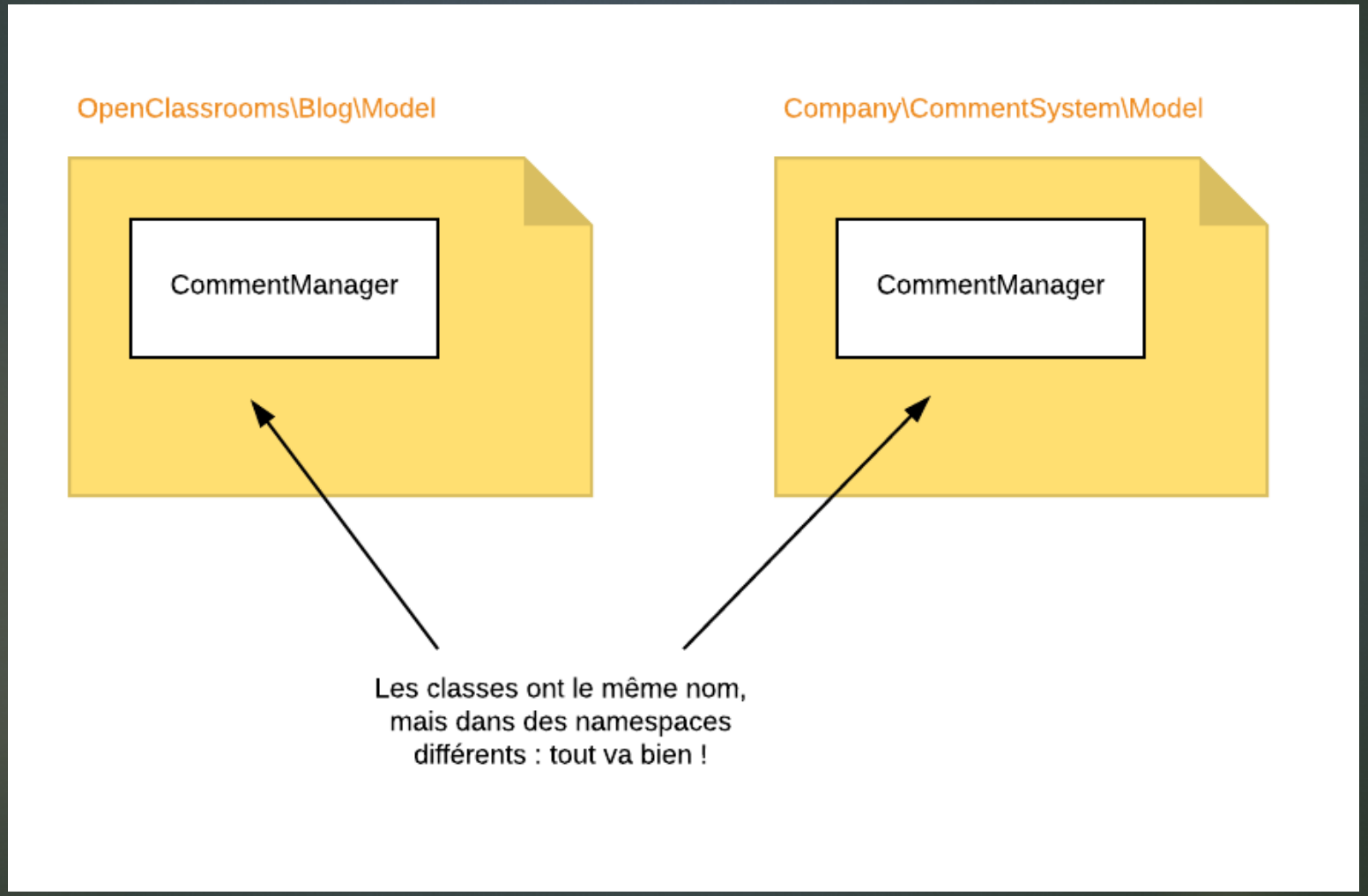
- Créez un nouveau dossier dans src nommé Classe
- Créez un nouveau fichier dedans, Search.php avec une class Search
- Elle aura pour attribut public : string et categories []
- Pensez à rajouter les php doc comme ceci :

```
/**
 * @var Category
 */
```

- Avant la classe il faut rajouter ceci (faites une recherche, pour essayer de comprendre ce qu'est un namespace:

```
namespace App\Classe;

use App\Entity\Category;
```

Namespaces


OpenClassrooms\Blog\Model
CommentManager
Company\CommentSystem\Model
CommentManager
Les classes ont le même nom,
mais dans des namespaces
différents : tout va bien !

Création du form (searchType)

Rajoutez à la main un searchType dans notre dossier form, inspirez vous du template des deux types que nous avons déjà crée.

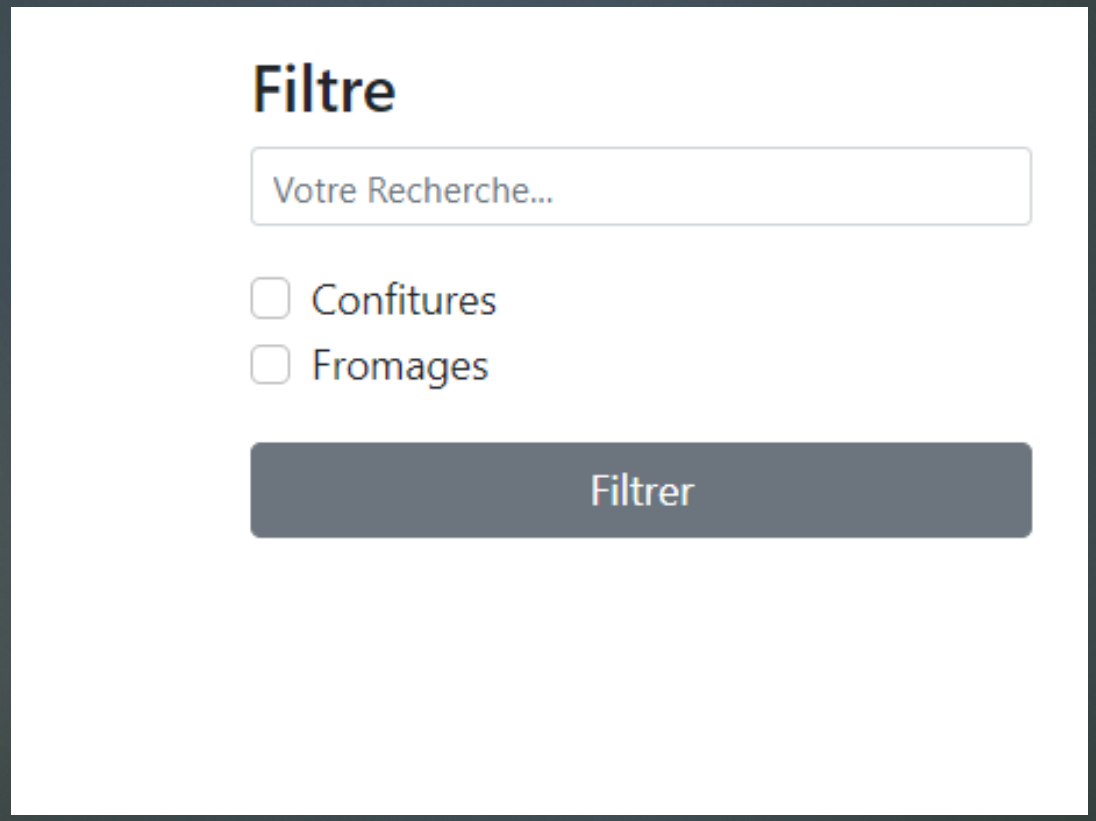
```
public function configureOptions(OptionsResolver $resolver)
 {
     $resolver->setDefaults([
         'data_class' => Search::class,
         "method"=>"GET",
         "csrf_protection"=>false,
     ]);
 }

 public function getBlockPrefix()
 {
     return "";
 }
```

Création du form (searchType)

```
public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
        ->add('string',TextType::class,[
            "label"=>false,
            "required"=>false,
            "attr"=>[
                "placeholder"=>"Votre Recherche...",
                "class"=>"mt-2 form-control-sm"
            ]
        ])
        ->add('categories',EntityType::class,[
            "label"=>false,
            "required"=>false,
            "class"=>Category::class,
            "multiple"=>true,
            "expanded"=>true
        ]);
    }
```

Intégrez le form au bon controller ainsi qu'à la vue twig

Je vous laisse rajouter au Type le bouton Submit.



Trier les produits

En premier lieu nous allons gérer le retour du form, comme nous l'avons fais plusieurs fois jusque là.

```
$form->handleRequest($request);
 if($form->isSubmitted() && $form->isValid()){
     $products = $this->entityManager->getRepository(Product::class)->findWithSearch($search);
 }
```

La fonction findWithSearch n'existe pas encore il faut la placer dans le repository de product et nous allons voir comment faire une requête sql en symfony.

Trier les produits

Faites des recherches pour comprendre le code ci-dessous et ajoutez la suite pour filtrer sur le champs de recherche et pas que sur les catégories.

```
public function findWithSearch(Search $search)
    {
        $query = $this
                ->createQueryBuilder('p')
                ->Select('c','p')
                ->join('p.category','c');

                if(!empty($search->categories)){
                    $query = $query
                    ->andWhere('c.id IN (:categories)')
                    ->setParameter('categories',$search->categories);
                }

        return $query->getQuery()->getResult();
```

Résumé

- Dans classe nous avons crée Search.php qui est un objet de recherche.
- Nous avons crée le SearchType qui fait le lien entre notre objet et le formulaire
- Nous avons découvert dans SearchType un nouveau typage, l'entityType. Ce qui permet de faire le lien entre une entité et un input.
- Dans ProductController nous avons traité les produits, un seul produit et les produits filtrés.
- Dans le repository grâce à la fonction FindWithSearch nous avons crée une requête à l'aide de doctrine et queryBuilder.
- Bonus, dans le productController il y a possibilité d'optimisé un peu, je vous laisse examiner.

C'est la fin de la partie PRODUIT, félicitation.

Exo bonus

- Arrangez vous pour qu'à la suppression d'un produit, sa photo soit également supprimée.