# - QueryBuilder-

Etend les possibilités de la classe Repository

C’est dans les classes Repository :

Exemple : Dans ThemeRepository, ajouter une méthode getAllThemes(pour l’exemple)

**public function** getAllThemes() {  
 // Recupérer l'instance de QueryBuilder. 't' est l'alias THEME car on est dans ThemeRepository  
 $qb = $this->createQueryBuilder('t');  
  
 $qb->select("t.name, count(p) as numerofPosts")  
 // 1ere argument est la variable qui stocke l'association  
 // le 2e argument représente l'alias de la 2e entité. En l'occurence c'est Post  
 ->innerJoin("t.posts", "p")  
 ->groupBy("t.id");  
  
 **return** $qb->getQuery();  
 }

A la fin de la méthode, faire un return de getQuery().

Puis dans l’action de notre Controller, appeler le getAllThemes via le repository. Voir code:

$repository = $this->getDoctrine()  
 ->getRepository("AppBundle:Theme");  
  
 $themeList = $repository->findAll();  
// Pour les tests du QueryBuilder  
$list = $repository->getAllThemes()->getArrayResult();

Voir doc à partir de la page 100

Installation de oro/doctrine-extensions

Utiliser certaines fonctions sql dans QueryBuilder car Doctrine ne le gère pas

composer require oro/doctrine-extensions

voir la doc :

<https://github.com/oroinc/doctrine-extensions>

pour la suite de l’installation pour la modification du fichier config.yml

Exemple 2 : Ajout d’une autre méthode dans le Repository avec QueryBuilder dans l’entité Post

**public function** getPostsGroupedByYear() {  
 $qb = $this->createQueryBuilder('p');  
  
 $qb->select('YEAR(p.createdAt) as yearPublished,  
 COUNT(p.id) as numberOfPosts ')  
 ->groupBy('YEAR(yearPublished)');  
  
 **return** $qb->getQuery()->getArrayResult();  
}

SLUG

stof/doctrine-extensions-bundle

Instalation

composer require stof/doctrine-extensions-bundle

Un fois installé, il faut le référencer dans AppKernel.php

**new** \Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),

Dans l’Entité où on veut le SLUG, on fait un use :

**use** Gedmo\Mapping\Annotation **as** Gedmo;

en l’occurrence, dans l’entité POST

Ajouter une colonne dans l’entité post :

*/\*\*  
 \** ***@ORM\Column****(name="slug", type="string", length=255, unique=true)  
 \** ***@Gedmo\Slug****(fields={"author", "title"})  
 \** ***@var*** *string  
 \*/***private** $slug;

Fields={} : on définit la liste des champs qui vont constituer le slug

Dans config.yml, activer stof avec la clé suivante :

**stof\_doctrine\_extensions:  
 default\_locale:** fr\_FR  
 **orm:  
 default:  
 sluggable:** true

On fait : php bin/console doctrine:schema:drop –force

Puis : php bin/console doctrine:schema:create

Enfin : php bin/console doctrine:fixtures:load

Après il faut changer la route et les liens

# FORMULAIRE : Synfony

Création des formulaire : theme et Post  
Mise en forme bootstrap en modifiant la clé twig dans config.yml, en ajoutant la clé suivante :  
 form\_themes:  
 - "bootstrap\_3\_layout.html.twig"

Pour la création des forms, taper la commande suivante pour le post :

* php bin/console doctrine:generate:form AppBundle:Post

Cette commande nous crée, dans notre bundle, le dossier form avec le fichier PostType.php qui représente notre Entité post.

# CKEditor

composer require egeloen/ckeditor-bundle

Modifier le AppKernel

**new** Ivory\CKEditorBundle\IvoryCKEditorBundle(),

puis faire un assets: install

* Php bin console ckeditor:install
* Php bin console assets:install (sert à sopier dans le dossier web, tous les fichiers js et css exposés par le bundle

Ajouter les liens js dans les blocks {%block javascript %}

{% **block** javascripts %}  
 <script src="{{ asset('bower-components/jquery/dist/jquery.min.js') }}"></script>  
 <script src="{{ asset('bower-components/bootstrap/dist/js/bootstrap.min.js') }}"></script>

<script src="{{ asset('bundles/ivoryckeditor/ckeditor.js') }}"></script>

{% **endblock** %}

Dans notre PostType, remplacer TextareaType ::class par CKEditorType::*class*

Installer un bundle de purification : exercise/html purifier-bundle

* Composer require exercise/htmlpurifier-bundle

Ajouter dans AppKernel :

**new** Exercise\HTMLPurifierBundle\ExerciseHTMLPurifierBundle(),

dans le twig qui doit afficher le texte, faire un PIPE purifier

{{ post.text | purify }}