SYMFONY

Les formulaires

Anthony PARIS - Formateur

PIXEL ONLINE CREATION

PRÉSENTATION

Nous avons pour le moment défini nos formulaires en HTML. Cependant en cas d'ajout de nouveaux attributs dans nos entités il faudra retoucher nos formulaires.

Nos formulaires actuels ne sont pas dynamiques.

Avec symfony nous POUVONS construire un formulaire en se basant sur un objet existant et son objectif est d'hydrater cet objet.

Nous possèdons déjà nos objets (Article, Image, Categorie).

En effet, les formulaires n'ont pas du tout besoin d'une entité pour se construire, mais uniquement d'un simple objet.

Heureusement, nos entités sont de simples objets avant d'être des entités, donc elles conviennent parfaitement.

PIXEL ONLINE CREATION

LES FORMULAIRES DANS LE CONTROLLER

Il est possible de générer son formulaire directement dans le controller :
Exemple

```php
namespace App\Controller;

use App\Entity\Task;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\HttpFoundation\Request;

class TaskController extends AbstractController
{
    public function new(Request $request)
    {
        // creates a task object and initializes some data for this example
        $task = new Task();
        $task->setTask('Write a blog post');
        $task->setDueDate(new \DateTime('tomorrow'));

        $form = $this->createFormBuilder($task)
            ->add('task', TextType::class)
            ->add('dueDate', DateType::class)
            ->add('save', SubmitType::class, ['label' => 'Create Task'])
            ->getForm();

        // ...
    }
}
```

Nous n'utiliserons pas cette méthode car elle est peut portable. Si on a besoin de re-utiliser une formulaire il faudra le re-définir.
Il est recommandé de mettre le moins possible de logique dans le controller.

PIXEL ONLINE CREATION

LES CLASSES FORMULAIRES

Pour des raisons de modularité, nous allons définir nos formulaires dans des classes, cela nous permettra de réutiliser la classe (le formulaire) où l'on veut.
Dans n'importe quel contrôleur juste en définissant une instance de la classe du formulaire.
C'est d'ailleurs l'intérêt principal de définir ces formulaires comme étant des classes.

Quand on utilise les formulaires en Symfony, nous passons par trois étapes:

-On construit le formulaire (dans le contrôleur ou en lui définissant une classe)

-On affiche le formulaire dans une vue

-On traite le formulaire pour valider les données et exécuter les actions demandées par l'utilisateur (ajouter un article par exemple).

Ce sera toujours dans cet ordre.

PIXEL ONLINE CREATION

LES CLASSES FORMULAIRES

Nous allons créer notre classe formulaire Article :

```
PS D:\wamp64\www\sf4m2icours> php bin/console make:form

 The name of the form class (e.g. TinyChefType):
 > ArticleType

 The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
 > Article

 created: src/Form/ArticleType.php


 Success!


 Next: Add fields to your form and start using it.
 Find the documentation at https://symfony.com/doc/current/forms.html
PS D:\wamp64\www\sf4m2icours>
```

Par convention on ajoutera le suffixe Type à notre entity
Dans notre cas ArticleType

Ouvrez ensuite le fichier :

PIXEL ONLINE CREATION

LES CLASSES FORMULAIRES

```php
class ArticleType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('content')
            ->add('publicationDate')
            ->add('lastUpdateDate')
            ->add('isPublished')
            ->add('image')
            ->add('categories')
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Article::class,
        ]);
    }
}
```

Nous avons demandé à créer un formulaire pour l'entité Article, une classe ArticleType a donc été créée.

Cette classe hérite de la classe AbstractType de Symfony et définit deux méthodes:

La premiere méthode buildForm() reçoit deux paramètres, dont le $builder qui va nous permettre de définir les champs de notre formulaire.

Chaque attribut que nous avons défini est considéré comme champs dans le formulaire à moins que nous décidons de le retirer nous même.

La deuxième méthode va nous permettre d'explicitement définir la classe à laquelle est rattachée ce formulaire.

PIXEL ONLINE CREATION

LES CLASSES FORMULAIRES

Nous allons supprimer les deux champs «date» car non utile dans le formulaire, ces éléments seront gérés automatiquement.

Nous allons mettre de côté «Image» et «Category» car ce sont des champs un peu spéciaux de part le système de relation

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('title')
        ->add('content')        You, an hour ago • ok
        ->add('isPublished')

    ;
}
```

Nous allons afficher ce «formulaire» dans notre page ajouter pour cela nous devons modifier le controller :

```
public function add()
{
  $form = $this->createForm(ArticleType::class);
  return $this->render("blog/ajout.html.twig", ["form"=>$form->createView()]);
}
```

CreateFrom : Permet de créer un formulaire à partir d'une classe formulaire, il est possible d'ajouter un deuxieme argument (un objet de type «Article» afin de fournir un formulaire pre-rempli.

CreateView : Permet de rendre une «vue» de formulaire

PIXEL ONLINE CREATION

LES CLASSES FORMULAIRES

Ouvrez ajout.html.twig et modifiez :

```
{% block content %}
    <div class="container">
        <div class="row">
            <div class="col ">
                <h1 class="text-center">Ajouter un article</h1>
                <hr>
            </div>
        </div>
        {{form(form)}}
    </div>
{% endblock %}      You, 12 days ago • ok
```

Vous avez un rendu non stylisé de votre «formulaire»

Title

Content

Is published

En modifiant la configuration de base de twig pour la génération des formulaires nous pouvons changer cela :

```
You, il y a 12 secondes | 1 author (You)
twig:
    default_path: '%kernel.project_dir%/templates'
    form_themes: ['bootstrap_5_horizontal_layout.html.twig']

when@test:
    twig:
        strict_variables: true
```

PIXEL ONLINE CREATION

LES CLASSES FORMULAIRES

Actualisez votre page :

Le formulaire s'affiche en utilisant bootstrap mais ne dispose pas de bouton de soumission ni du choix de la catégorie et l'ajout de l'image.

Retrouvez plus d'infos sur le templating de formulaire ici :
https://symfony.com/doc/current/form/form_themes.html

Revenons sur notre formulaire et ajoutons de la configuration :

```
$builder
    ->add('title', TextType::class,["label"=>"Titre"])
    ->add('content', TextareaType::class,["label"=>"Contenu"])
    ->add('isPublished', CheckboxType::class, ["label"=>"Publier l'article?"])
    ->add('save', SubmitType::class);

;
```

Nous ajoutons un bouton de soumission, le type de chaque champs ainsi que des options en 3eme argument.

PIXEL ONLINE CREATION

FORMULAIRES IMBRIQUÉS

C'est souvent le cas lorsque vous avez des relations entre vos objets : vous souhaitez ajouter un objet A, mais en même temps un autre objet B qui est lié au premier.
Ici je veux ajouter Image en même temps que Article

Considérez un de vos formulaires comme un champ, et appelez ce simple champ depuis un autre formulaire.

Par conséquent nous devons générer le formulaire pour l'image

```
PS D:\wamp64\www\sf4m2icours> php bin/console make:form

 The name of the form class (e.g. BraveGnomeType):
 > ImageType

 The name of Entity or fully qualified model class name
 > Image

 created: src/Form/ImageType.php

 Success!
```

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('chemin', FileType::class, ["label"=>"Votre fichier"])
        ->add('alt', TextType::class)
        ->add('isPublished', CheckboxType::class, array(
                                                    'label'=>'Publier l\'image?',
                                                    'attr' => array('checked' => 'checked')
                                                    ))
    ;
}
```

Nous modifions le formulaire image pour convenir à notre besoin.
A noter ici pas de bouton de validation car le formulaire sera imbriqué. Pas besoin de spécifier qu'un champ est «required» Symfony le mappe initialement de l'anotation de vos entity.
Il faut se servir du 3eme paramètre comme surcharge.

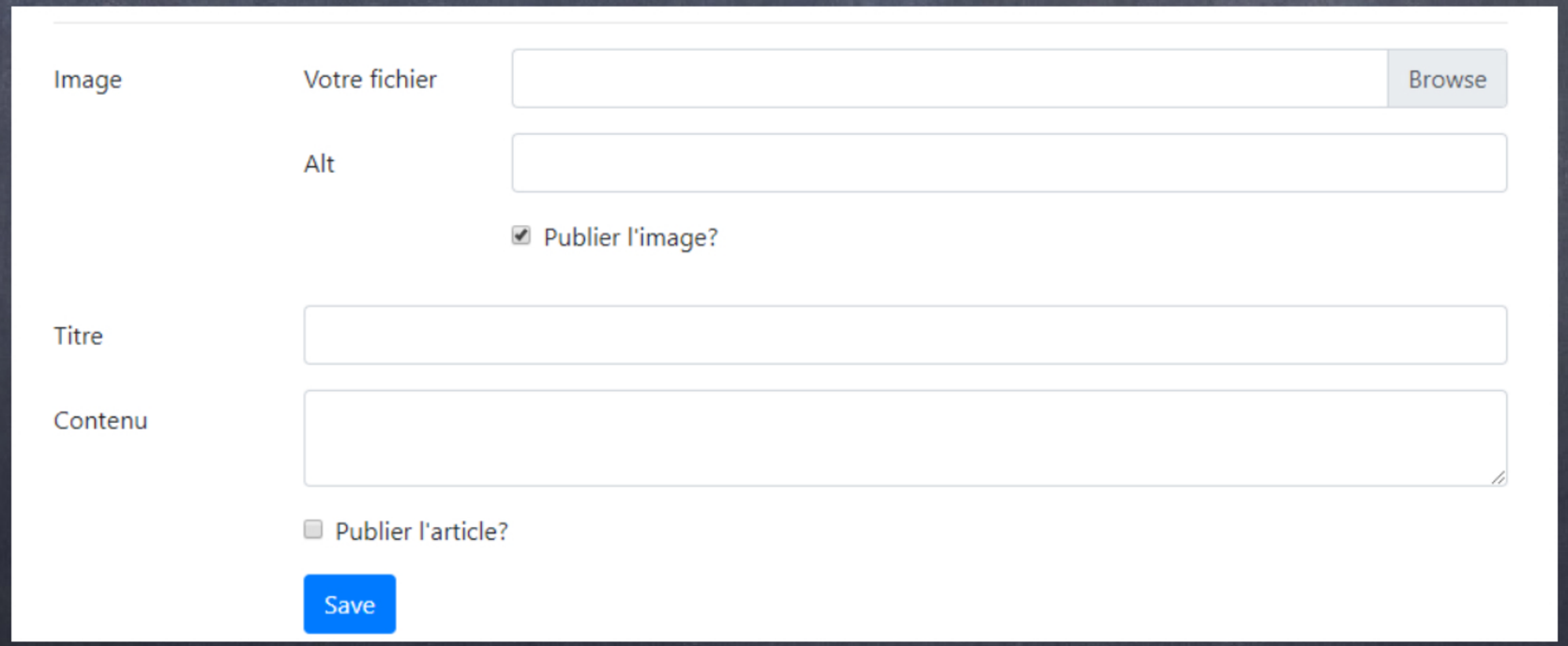
PIXEL ONLINE CREATION

FORMULAIRES IMBRIQUÉS

Je reviens sur «ArticleType.php» et j'ajoute mon formulaire «ImageType» comme simple champ :

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('image',     ImageType::class)
        ->add('title', TextType::class,["label"=>"Titre"])
        ->add('content', TextareaType::class,["label"=>"Contenu"])
        ->add('isPublished', CheckboxType::class, ["label"=>"Publier l'article?"])
        ->add('save', SubmitType::class);

    ;
}
```

Je peux vérifier le rendu en ligne :



PIXEL ONLINE CREATION

ENTITY TYPE

Nous avons ajouté un upload de l'image, cependant nous ne voulons pas ajouter des catégories en même temps que l'article mais en selectionner des existantes.

(Si nous souhaitons ajouter des catégories en même temps cela est possible en utilisant l'imbrication comme vu avant, sauf qu'ici la relation étant ManyToMany entre Article et Category l'ajout de «category» serait multiple, cependant ceci est un comportement très rare) Le type de champ utilisé dans ce cas est «CollectionType»

Ici nous voulons sélectionner des «category» existantes.
Le type de champ utilisé est «EntityType» :

Le type EntityType permet de sélectionner des entités. D'un <select> ou list radio / checkbox côté formulaire HTML, vous obtenez une ou plusieurs entités côté formulaire Symfony.

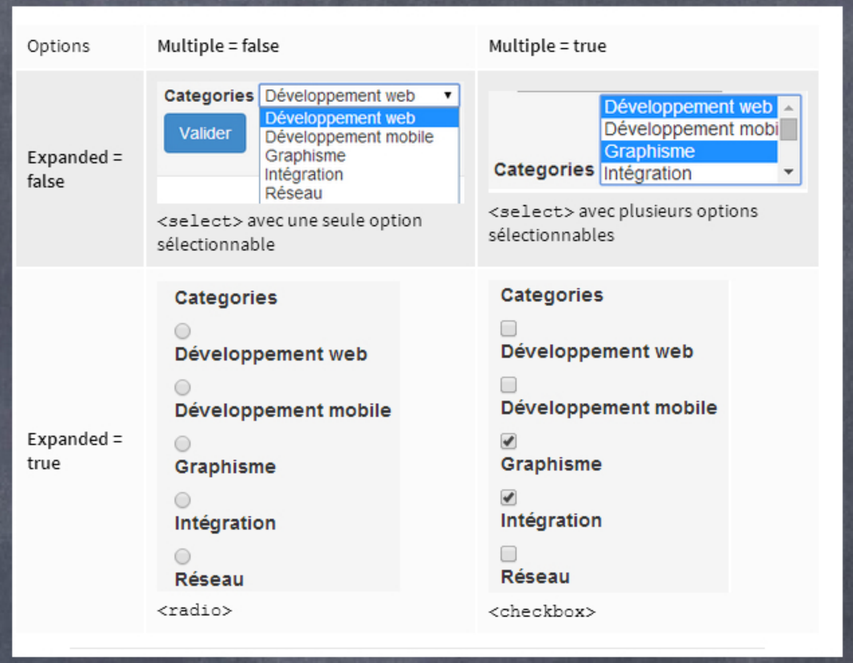
A noter : Il était aussi possible d'ajouter le entityType pour la partie «Image» C'est-à-dire selectionner une image existante, cependant dans l'exemple cela reste un choix judicieux d'utiliser un formulaire imbriqué pour l'image

PIXEL ONLINE CREATION

ENTITY TYPE

Dans ArticleType.php

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('image',      ImageType::class)
        ->add('title', TextType::class,["label"=>"Titre"])
        ->add('content', TextareaType::class,["label"=>"Contenu"])
        ->add('categories', EntityType::class, [
            'class' => Category::class,
            'choice_label' => 'label',
            'multiple' => true,
            'expanded' => true
        ])
        ->add('isPublished', CheckboxType::class, ["label"=>"Publier l'article?"])
        ->add('save', SubmitType::class);

    ;
}
```



Ici multiple est impératif car la relation et ManyToMany
En effet Article souhaite une liste de «category» et non une «category»

PIXEL ONLINE CREATION

QUERY_BUILDER

Comme vous avez pu le constater, toutes les catégories de la base de données apparaissent dans ce champ. Or parfois ce n'est pas le comportement voulu. Imaginons par exemple ne lister que les catégories publiées Tout est prévu : il faut jouer avec l'option query_builder.

Cette option porte bien son nom puisqu'elle permet de passer au champ un QueryBuilder, que vous connaissez depuis la partie sur Doctrine. Tout d'abord, créons une méthode dans le repository de l'entité du champ (dans notre cas,CategoryRepository) qui retourne le bon QueryBuilder :

```php
class CategoryRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Category::class);
    }

    public function getPublish()
    {
        return $this
        ->createQueryBuilder('c')
        ->where('c.isPublished = :val')
        ->setParameter('val', true)
        ;
    }
```

```php
class ArticleType extends AbstractType        You, 19 hours ago • ok
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('image',     ImageType::class)
            ->add('title', TextType::class,["label"=>"Titre"])
            ->add('content', TextareaType::class,["label"=>"Contenu"])
            ->add('categories', EntityType::class, [
                'class' => Category::class,
                'choice_label' => 'label',
                'multiple' => true,
                'expanded' => true,
                'query_builder' => function(CategoryRepository $repository) {
                    return $repository->getPublish();
                }
            ])
            ->add('isPublished', CheckboxType::class, ["label"=>"Publier l'article?"])
            ->add('save', SubmitType::class);

        ;
```

PIXEL ONLINE CREATION

FORMULAIRE

Nous disposons enfin d'un formulaire propre à notre besoin :

Image

Votre fichier

Browse

Alt

☑ Publier l'image?

Titre

Contenu

Categories

☐ Informations

☐ Programmation

☐ Publier l'article?

Save

PIXEL ONLINE CREATION

FORMULAIRE DETAILLE

Qu'est-ce que le CSRF?

Si vous affichez la source vous voyez un champs hidden «token»

Le champ CSRF, pour Cross Site Request Forgeries, permet de vérifier que l'internaute qui valide le formulaire est bien celui qui l'a affiché.

C'est un moyen de se protéger des envois de formulaire frauduleux.

C'est un champ que Symfony rajoute automatiquement à tous vos formulaires, afin de les sécuriser sans même que vous vous en rendiez compte.

Ce champ s'appelle _token dans vos formulaires, vous pouvez le voir si vous affichez la source HTML (il est généré par la méthode form ou form_rest() , donc à la fin de votre formulaire).

PIXEL ONLINE CREATION

FORMULAIRE DETAILLE

Pour informations, nous avons le template ajout.html.twig et edition.html.twig qui utilisent même template.
Nous pouvons inclure notre template détaillé (si l'on souhaite retoucher un peu le visuel)

Ajout et Edition.html.twig

```
You, 13 days ago | 1 author (You)
{% extends "base.html.twig" %}

{% block title %}Ajouter un article{% endblock %}

{% block content %}
    <div class="container">
        <div class="row">
            <div class="col ">
                <h1 class="text-center">Ajouter un article</h1>
                <hr>
            </div>
        </div>
    </div>
    {% include "parts/form_article.html.twig" %}
{% endblock %}      You, 13 days ago • ok
```

```
templates > parts > form_article.html.Twig >
You, 11 minutes ago | 1 author (You)
1  <div class="container">
2      {{ form(form) }}
3  </div>      You, 13 days ago
```

**Remarquez l'erreur de «traductions manquantes», si vous souhaitez la desactiver il suffit d'utiliser : translation_domain : false dans les champs du formulaire.

https://symfony.com/doc/current/reference/forms/types/form.html#translation-domain

PIXEL ONLINE CREATION

FORMULAIRE DETAILLE

Dans notre exemple, notre formulaire s'affiche correctement, cependant il est possible de le décomposer pour gérer son affichage complet :

form_start() affiche la balise d'ouverture du formulaire HTML, soit <form>. Il faut passer la variable du formulaire en premier argument, et les paramètres en deuxième argument. L'index attr des paramètres, et cela s'appliquera à toutes les fonctions suivantes, représente les attributs à ajouter à la balise HTML générée, ici le<form>. Il nous permet d'appliquer une classe CSS au formulaire, ici form-horizontal.

form_errors() affiche les erreurs attachées au champ donné en argument.

form_label() affiche le label HTML du champ donné en argument. Le deuxième argument est le contenu du label.

form_widget() affiche le champ HTML lui-même (que ce soit<input>,<select>, etc.).

form_row() affiche le label, les erreurs et le champ en même temps, en respectant la vue définie dans le thème du formulaire que vous utilisez.

form_rest() affiche tous les champs manquants du formulaire (par exemple le champ CSRF).

form_end() affiche la balise de fermeture du formulaire HTML, soit</form>. (implique un form_rest() par défaut)

PIXEL ONLINE CREATION

FORMULAIRE DETAILLE

A voir :
https://symfony.com/doc/current/form/form_customization.html
https://symfony.com/blog/new-in-symfony-4-4-bootstrap-custom-switches

Modifions form_article.html.twig

```
<div class="container">
    {{ form_start(form) }}

    <h2 class="text-center">::IMAGE::</h2>
    {{ form_row(form.image.chemin) }}
    {{ form_row(form.image.alt) }}
    {{ form_row(form.image.isPublished, {"label_attr": {"class": "switch-custom"}}) }}

    <h2 class="text-center">::ARTICLE::</h2>
    {{ form_row(form.title) }}
    {{ form_row(form.content) }}
    {{ form_row(form.isPublished, {"label_attr": {"class": "switch-custom"}}) }}

    <h2 class="text-center">::CATEGORIE(S)::</h2>
    {{ form_row(form.categories, {"label_attr": {"class": "checkbox-custom"}})}}


    {{ form_end(form) }}
</div>
```

PIXEL ONLINE CREATION

FORMULAIRE DETAILLE

Vous remarquerez que le bouton de chargement fichier n'est pas en français : Pour cela on ouvre notre fichier assets/css/app.css

```
.custom-file-label::after {
    content: "Parcourir";
}       You, 13 days ago • ok
```

On lance :

```
yarn encore dev
yarn run v1.22.4
$ D:\wamp64\www\sf4m2icours\node_modules\.bin\encore dev
Running webpack ...

DONE  Compiled successfully in 2460ms

I  3 files written to public\build
Entrypoint assets/app = runtime.js assets/app.css assets/app.js
Done in 8.47s.
```

PIXEL ONLINE CREATION

FORMULAIRE DETAILLE

Nous obtenons un formulaire beaucoup plus stylisé et surtout pour lisible pour l'administrateur :

Ajouter un article

::IMAGE::

Votre fichier [ ] Parcourir

Alt [ ]

Publier l'image?

::ARTICLE::

Titre [ ]

Contenu [ ]

Publier l'article?

::CATEGORIE(S)::

Categories
- [ ] Informations
- [ ] Programmation

Save

© 2020 Copyright: Le bonheur ne s'acquiert pas, il ne réside pas dans les apparences, chacun d'entre nous le construit à chaque instant de sa vie avec son coeur.

PIXEL ONLINE CREATION

VALIDATOR

Validator est un service indépendant permettant la validation des données. On l'utilisera principalement pour les formulaires.
https://symfony.com/doc/current/reference/constraints.html

Par exemple il est important pour notre projet d'avoir article.title non vide ou null, image.chemin non null, image.alt non null

Nous allons utiliser les attributs dans nos entités pour définir cela.

Entity : Article

```
use Symfony\Component\Validator\Constraints as Assert;

You, il y a 9 secondes | 1 author (You)
#[ORM\Entity(repositoryClass: ArticleRepository::class)]
class Article
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    #[Assert\NotBlank]        You, il y a 11 secondes • Uncommitt
    private ?string $title = null;
```

Entity : Imag

```
use Symfony\Component\Validator\Constraints as Assert;

You, il y a 1 seconde | 1 author (You)
#[ORM\Entity(repositoryClass: ImageRepository::class)]
class Image
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(type: Types::TEXT)]
    #[Assert\NotBlank]
    private ?string $chemin = null;

    #[ORM\Column(length: 255)]
    #[Assert\NotBlank]        You, il y a 1 seconde • Uncommi
    private ?string $alt = null;
```

PIXEL ONLINE CREATION

VALIDATOR

Nous avons ajouté l'attribut «NotBlank» permettant de s'assurer qu'un attribut ne soit pas vide ou null.
Ne pas oublier le «use»
Il existe beaucoup de filtres de validation :

https://symfony.com/doc/current/validation.html#supported-constraints

Modifions config/packages/framework.yaml pour ajouter la prise en charge du CSRF de protection ainsi que la validation par annotation

```yaml
framework:
    secret: '%env(APP_SECRET)%'
    csrf_protection: true
    http_method_override: false

    # Enables session support. Note that the session will ONLY be
    # Remove or comment this section to explicitly disable sessio
    session:
        handler_id: null
        cookie_secure: auto
        cookie_samesite: lax
        storage_factory_id: session.storage.factory.native

    #esi: true
    #fragments: true      You, il y a 3 jours • Add initial set c
    php_errors:
        log: true
```

PIXEL ONLINE CREATION

VALIDATOR

Revenons sur : src/Form/ArticleType.php pour indiquer que «title» peut être null afin d'enlever la validation html et tester notre service validator :

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('image',     ImageType::class)
        ->add('title', TextType::class,["label"=>"Titre", 'required' => false])
        ->add('content', TextareaType::class,["label"=>"Contenu"])
```

Dans le controller (méthode add) : Lançons la validation d'un formulaire

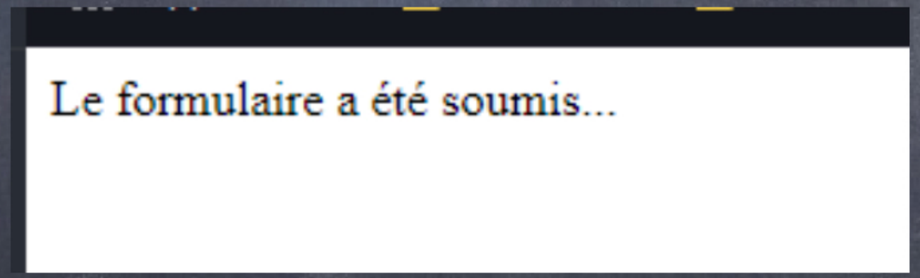
```
/**
 * @Route("/ajouter", name="article_add")        You, 13 days ago • ok
 */
public function add(Request $request)
{
  $article = new Article();
    $form = $this->createForm(ArticleType::class, $article);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        return new Response('Le formulaire a été soumis...');
    }
  return $this->render("blog/ajout.html.twig", ["form"=>$form->createView()]);
}
```

PIXEL ONLINE CREATION

VALIDATOR

En définissant la variable locale en FR (dans config/translations.yaml) on obtiendra un message d'erreur en français.
Si vous testez et ne remplissez pas le title vous obtiendrez : Car la méthode «isValid()» à renvoyé false :

Si vous validez correctement l'article vous aurez :



Voir : https://symfony.com/doc/current/forms.html#validating-forms

PIXEL ONLINE CREATION

VALIDATOR

Vous pouvez d'ailleurs visualiser dans le profiler le debug de validator :

- article
  - image
  - title 1
  - content
  - Categories
  - isPublished
  - save
  - _token

title

"Symfony\Component\Form\Extension\Core\Type\TextType"

Errors

Message	Origin	Cause
Le champ titre ne doit pas être vide!	title	Caused by: Symfony\Component\Validator\ConstraintViolation {#1284 ▼ root: Symfony\Component\Form\Form {#813 …} path: "data.title" value: null }

Default Data

Vous pouvez modifier le message directement dans l'entity

```
#[ORM\Column(length: 255)]
#[Assert\NotBlank(message:"Le champ titre ne doit pas être vide!")]
private ?string $title = null;
```

PIXEL ONLINE CREATION

VALIDATION DU FORMULAIRE

Nous allons modifier notre controller afin de pouvoir enregistrer en base de données notre article et image.
Nous avons déjà vu l'enregistrement en BDD avec les fixtures ici c'est assez identique sauf qu'il y a la validation du formulaire en plus.

```php
#[Route(path: '/ajouter', name: 'article_add')]
public function add(Request $request, ManagerRegistry $doctrine)
{
    $article = new Article();
    $form = $this->createForm(ArticleType::class, $article);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $article->setLastUpdateDate(new \DateTime());

        if($article->isIsPublished()) {
          $article->setPublicationDate(new \DateTime());
        }

        $entityManager = $doctrine->getManager(); //On  Récupère entity manager
        $entityManager->persist($article); //On confie l'article à l'entity manager
        $entityManager->flush(); //On execute la requete

    }
    return $this->render("blog/ajout.html.twig", ["form"=>$form->createView()]);
}
```

PIXEL ONLINE CREATION

VALIDATION DU FORMULAIRE

Nous initialisons la date de dernière modification, puis nous vérifions si l'article est publié et nous initialisons aussi la date de publication.

Ce qui nous intéresse c'est l'entity manager nous récupérons l'entity manager, puis nous demandons à l'entity manager de gérer l'objet article.

On persiste l'objet article, mais jusqu'ici rien n'est fait, aucune requête.

Nous appelons la methode flush() de l'entity manager, à partir de là, l'entity manager va regarder toutes les entités que nous lui avons confier avec persist() et ajouter l'entité si celui-ci n'existe pas en base de données, ou le modifier s'il existe.

Dans notre cas l'entité sera ajoutée en base de données et également l'image car nous avons défini un «cascade:persist» dans l'entité.

Le souci que nous pouvons observer et au niveau de l'image qui ne s'enregistre pas au bon endroit.

PIXEL ONLINE CREATION

VALIDATION DU FORMULAIRE

Nous allons modifier config/services.yaml pour ajouter un chemin en paramètre

```yaml
# Put parameters here that don't need to change on each machine where the app is deployed
# https://symfony.com/doc/current/best_practices/configuration.html#application-related-configur
parameters:
    images_directory: '%kernel.project_dir%/assets/images/articles'
```

Nous allons modifier le controller pour ajouter la gestion de l'image, de notre paramètre et la gestion d'un message «flashBag» et une redirection vers l'accueil

```php
#[Route(path: '/ajouter', name: 'article_add')]
public function add(Request $request, ManagerRegistry $doctrine)
{
    $article = new Article();
    $form = $this->createForm(ArticleType::class, $article);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $article->setLastUpdateDate(new \DateTime());

        if($article->isIsPublished()) {
          $article->setPublicationDate(new \DateTime());
        }

        if ($article->getImage()->getChemin() !== null) {
          $file = $form->get('image')->get('chemin')->getData();
          $fileName =  uniqid(). '.' .$file->guessExtension();

          try {
              $file->move(
                  $this->getParameter('images_directory'), // Le a
                  $fileName
              );
          } catch (FileException $e) {
              return new Response($e->getMessage());
          }

          $article->getImage()->setChemin($fileName);
      }
```

You, il y a 6 minutes • Uncommitted changes

```php
        $entityManager = $doctrine->getManager(); //On  Récupère entity manager
        $entityManager->persist($article); //On confie l'article à l'entity manager
        $entityManager->flush(); //On execute la requete

        $this->addFlash('info', 'Votre article a été ajouté à la base de données!');

        //REDIRECTION
        return $this->redirectToRoute('homepage');

    }
    return $this->render("blog/ajout.html.twig", ["form"=>$form->createView()]);
}
```

You, il

PIXEL ONLINE CREATION

VALIDATION DU FORMULAIRE

On modifie la vue «index» pour la gestion de la session flash

```
<div class="container">
    <div class="row">
        <div class="col ">
            <h1 class="text-center">Bienvenu sur mon blog</h1>
        </div>
    </div>
</div>
<div class="container">
    <div class="row">
        <div class="col ">
            <div class="alert alert-info" role="alert">
                {% for flashMessage in app.flashes('info') %}

                    {{ flashMessage }}

                {% endfor %}
            </div>
        </div>
    </div>
</div>
```

Maintenant quand vous ajouterez un article , la photo sera ajoutée dans assets/images/articles/... il faudra lancer la commande yarn encore dev pour l'ajouter dans public si cela est problèmatique, ajoutez directement l'image dans public.
Ici je la laisse dans assets car c'est l'admin qui poste le message (yarn encore dev)
Ensuite vous serez redirigé vers l'accueil avec un message flash!
https://symfony.com/doc/current/controller/upload_file.html (vous pouvez créer un service)

PIXEL ONLINE CREATION

LA MODIFICATION

Nous souhaitons maintenant modifier un article.

Nous allons modifier le controller «edit» et notre vue le temps de reconstruire le formulaire

```php
#[Route(path: '/edition/{id}', name: 'article_edit', requirements: ['id' => '\d{1,8}'])]
public function edit($id, ManagerRegistry $doctrine)
{
    $entityManager = $doctrine->getManager();
    $article = $entityManager->getRepository(Article::class)->find($id);

    return $this->render("blog/edition.html.twig", ["article"=>$article]);
}
```

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

{% block title %}Modifier{% endblock %}

{% block content %}
    <div class="container">
        <div class="row">
            <div class="col ">
                <h1 class="text-center">Modification article : {% if article is not null %}{{article.title}}{% endif %}</h1>
                <hr>
            </div>
        </div>
    </div>
    {#% include "parts/form_article.html.twig" %#}
{% endblock %}
```

flas — Aucun résultat

You, 13 days ago • ok

Lancez ensuite une URL avec un id correspondant à une ligne dont l'ID «Article» de votre base de données sera le même.

Ici à partir de l'ID, nous allons chercher via le repository Article, l'objet «Article» dont l'ID est égale à l'ID URL, Si l'ID n'existe pas nous devrons lancer une erreur 404.

Nous pouvons cependant utiliser le PARAM_CONVERTER
https://symfony.com/doc/current/bundles/SensioFrameworkExtraBundle/annotations/converters.html

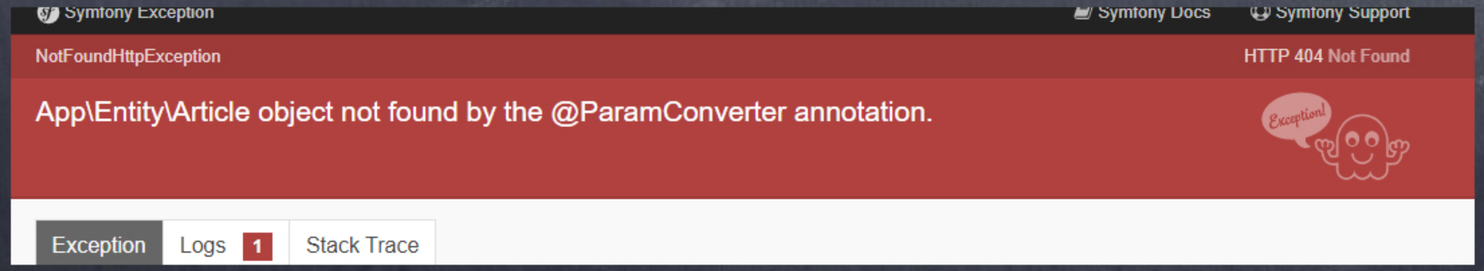
PIXEL ONLINE CREATION

LA MODIFICATION

Il s'agit de transformer automatiquement un paramètre de route, comme {id} en un objet, une entité $article par exemple.

```
#[Route(path: '/edition/{id}', name: 'article_edit', requirements: ['id' => '\d{1,8}'])]
public function edit($id, ManagerRegistry $doctrine)
{
    $entityManager = $doctrine->getManager();
    $article = $entityManager->getRepository(Article::class)->find($id);

    return $this->render("blog/edition.html.twig", ["article"=>$article]);
}
```

Rien ne change dans la vue.
L'avantage ici est une rapidité d'écriture et evite également de devoir créer la requête.
Cela est possible car l'annotation d'attribut id = un attribut dans l'entity.
Il est possible de mapper le paramConverter avec d'autres attributs.
En plus de cela en cas d'un ID de l'url non équivalent à un id de la table Article une erreur 404 est lancée sans rien configurer



PIXEL ONLINE CREATION

LA MODIFICATION

Nous allons maintenant créer le formulaire comme nous l'avons déjà fait pour l'ajout d'un article et l'envoyer à la vue:

```php
public function edit(Article $article)
{

  $form = $this->createForm(ArticleType::class, $article);

  return $this->render('blog/edition.html.twig', [
        'article' => $article,
        'form' => $form->createView(),
    ]);
}
```

Et nous allons retirer le commentaire dans le fichier edition.html.twig:

```twig
{% block content %}
    <div class="container">
        <div class="row">
            <div class="col ">
                <h1 class="text-center">Modification article : {% if article is not null %}{{article.title}}{% endif %}</h1>
                <hr>
            </div>
        </div>
    </div>
    {% include "parts/form_article.html.twig" %}
{% endblock %}      You, 13 days ago • ok
```

Lancez une URL valide (ID existant dans la table «Article» de votre BDD)

PIXEL ONLINE CREATION

LA MODIFICATION

Une erreur se lance :

Les données de vue du formulaire devraient être une instance de la classe Symfony \ Component \ HttpFoundation \ File \ File, mais il s'agit d'une chaîne (n). Vous pouvez éviter cette erreur en définissant l'option "data_class" sur null ou en ajoutant un transformateur de vue qui transforme une chaîne (n) en une instance de Symfony \ Component \ HttpFoundation \ File \ File.

Etant en local nous passerons le «data_class» sur null sinon en production il faudra utiliser un transformateur :

https://stackoverflow.com/questions/54596779/error-when-rendering-symfony-4-edit-form-that-includes-a-file-type

Nous avons une erreur image.chemin est un fichier, c'est comme tel que nous l'avons défini dans le formulaire (datatype file), mais quand on enregistre notre entité en base de données, nous enregistrons une chaine de caractères.

Il faut savoir que quand nous créons un formulaire avec une entité qui contient des données, Symfony pré-rempli déjà le formulaire avec les informations de l'entité.

image.chemin est un fichier dans le formulaire et l'attribut image.chemin de l'entité $article est un string.

Nous allons modifier ImageType pour ajouter l'option data_class à null afin de lever l'exception et pour lui indiquer qu'il ne recevra pas forcément un objet de type file

PIXEL ONLINE CREATION

LA MODIFICATION

Dans Form/ImageType.php

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('chemin', FileType::class, ["label"=>"Votre fichier",'data_class' => null])
        ->add('alt', TextType::class)
        ->add('isPublished', CheckboxType::class, array(
                                                    'label'=>'Publier l\'image?',
                                                    'attr' => array('checked' => 'checked')
                                                    ))
    ;
}
```

Le formulaire est pré-rempli avec les informations que nous avons déjà enregistrées.

Comment prendre en compte l'image?

Nous allons prévoir une section sur la vue pour afficher l'ancienne image

Dans le contrôleur, nous allons récupérer l'ancienne image avant de créer le formulaire, et à chaque fois que l'utilisateur soumettra le formulaire, on vérifiera s'il a chargé une nouvelle image, si c'est le cas on chargera alors la nouvelle image et on remplacera l'ancienne, sinon on gardera l'ancienne image.

PIXEL ONLINE CREATION

LA MODIFICATION

Modifications de la vue pour afficher l'image existante (form_article.html.twig):

```
<div class="container">
    {{ form_start(form) }}

    <h2 class="text-center">::IMAGE::</h2>
    {% if article.image.chemin is not null %}
    <img width="500" src="{{ asset('build/images/articles/' ~ article.image.chemin) }}" class="rounded mx-auto d-block img-fluid">
    {% endif %}
```

Pensez à Yarn dev encore si l'image est non visible.
Et dans le controller assez similaire à l'ajout.

```
#[Route(path: '/edition/{id}', name: 'article_edit', requirements: ['id' => '\d{1,8}'])]
public function edit(Article $article, Request $request, ManagerRegistry $doctrine)
{
  $oldImage = $article->getImage()->getChemin();
  $form = $this->createForm(ArticleType::class, $article);
  // $form['image']['chemin']->setData("ok");
  // dump($form['image']['chemin']->getData());

   $form->handleRequest($request);
  // $form['image']['chemin']->setData("ok");
  // dump($article);

  if ($form->isSubmitted() && $form->isValid()) {
        $article->setLastUpdateDate(new \DateTime());

        if ($article->isIsPublished()) {
            $article->setPublicationDate(new \DateTime());
        }

        if ($article->getImage()->getChemin() !== null && $article->getImage()->getChemin() !== $oldImage) {
          $file = $form->get('image')->get('chemin')->getData();
          $fileName =  uniqid(). '.' .$file->guessExtension();        You, il y a 5 minutes • Uncommitted chang
```

```
            try {
                $file->move(
                    $this->getParameter('images_directory'), // Le dossier dans le quel le fichier va etre c
                    $fileName
                );
            } catch (FileException $e) {
                return new Response($e->getMessage());
            }

            $article->getImage()->setChemin($fileName);
        } else {
            $article->getImage()->setChemin($oldImage);
        }


        $em = $doctrine->getManager(); // On récupère l'entity manager
        $em->persist($article); // On confie notre entité à l'entity manager (on persist l'entité)
        $em->flush(); // On execute la requete        You, il y a 6 minutes • Uncommitted changes

        $this->addFlash('info', 'Votre article a été modifié!');

        //REDIRECTION
        return $this->redirectToRoute('homepage');
    }
return $this->render("blog/edition.html.twig", ["form"=>$form->createView(), "article" => $article]);
```

PIXEL ONLINE CREATION

LA MODIFICATION

Cela renverra une erreur du champ chemin qui ne peut etre null, cela nécessite de modifier légèrement notre entité et le formulaire pour accepter la valeur null lors de la soumission

```php
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add('chemin', FileType::class, ["label"=>"Votre fichier",'data_class' => null,'required'=>false,'empty_data' => ''])
        ->add('alt', TextType::class)
        ->add('isPublished', CheckboxType::class, array(
                                                'label'=>'Publier l\'image?',
                                                'attr' => array('checked' => 'checked')
                                                ))
    ;
}
```

```php
public function setChemin(?string $chemin): self
{
    $this->chemin = $chemin;

    return $this;
}
```

https://ourcodeworld.com/articles/read/1441/how-to-solve-symfony-5-exception-expected-argument-of-type-string-null-given-at-property-path

PIXEL ONLINE CREATION

LA MODIFICATION

Vous pouvez également ajouter ceci sur l'image pour l'ajout d'un article sans image ou sinon définir une image par défaut dans la controller

```
#[ORM\Column(type: Types::TEXT, nullable: true)]
#[Assert\NotBlank]
private ?string $chemin = null;
```

Si vous choisissez cette option pensez à faire :
php bin/console doctrine:migrations:migrate
Pour que la modification affecte la structure de la base de données

LA MODIFICATION

Ici nous sommes en modification il s'agit donc d'un update de l'article et de l'image existante.

Si une nouvelle image a été envoyée, l'image sera remplacée dans la base sinon rien ne change!

Une fois validé, vous êtes redirigé vers l'accueil

LISTING ARTICLES

Nous allons maintenant lister les articles publiés sur la page d'accueil.
Pour cela, nous allons aussi utiliser une fonction déjà définie Symfony findBy() qui prend 4 parametres:

Un tableau des critères, c'est dans ce tableau nous allons mentionner que nous voulons juste les articles publiés par exemple, ou les articles d'un auteur spécifique

Un tableau pour définir l'ordre.

Un entier qui représente le nombre d'éléments à récupérer, 10 articles par exemple pour une pagination

Un entier qui représente le début de la sélection, on prend à partir du 10ème article

Les trois derniers paramètres ne sont pas obligatoires.

Dans notre cas, nous allons récupérer tous les articles qui ont été publiés par ordre décroissant de la date de publication.

PIXEL ONLINE CREATION

LISTING ARTICLES

Pour des besoins pratiques nous modifierons «show» du controller en passant par le param converter

```
#[Route(path: '/article/{title}-{id}', name: 'article_show', requirements: ['id' => '\d{1,8}', 'title' => '.{1,}'])]
public function show(Article $article)
{
    return $this->render("blog/lecture.html.twig", ["article"=>$article]);
}
```

Ensuite nous utilisons notre méthode «Index» pour récupérer le listing des articles publiés grâce à «findBy», nous passons le tableau à la vue

```
#[Route(path: '/', name: 'homepage')]
  public function index(ManagerRegistry $doctrine)
  {
    //PAS BESOIN DE PASSER PAR GETMANAGER CAR LES DONNEES SONT DEJA EN BASES (C'EST FACULTATIF DANS CE CAS)
    //MAIS SINON VOUS POUVEZ FAIRE $em = $doctrine->getManager() ca marche aussi
    $articles = $doctrine->getRepository(Article::class)->findBy(
      ['isPublished' => true],
      ['publicationDate' => 'desc']);
      return $this->render("blog/index.html.twig",["articles"=>$articles]);
  }
```

LISTING ARTICLES

Le haut de la vue «index.html.twig» ne change pas sauf petite correction pour la partie «flashbag» pour éviter l'affichage du «alert» bootstrap sans message (bandeau bleu)

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

{% block title %}Ma page d'accueil | {{ parent() }}{% endblock %}

{% block content %}
{% include "parts/banniere.html.twig" %}

<div class="container">
    <div class="row">
        <div class="col ">
            <h1 class="text-center">Bienvenu sur mon blog</h1>
        </div>
    </div>
</div>
<div class="container">
    <div class="row">
        <div class="col ">
            {% for flashMessage in app.flashes('info') %}
                <div class="alert alert-info" role="alert">
                    {{ flashMessage }}
                </div>
            {% endfor %}
        </div>
    </div>
</div>
```

Pour garder notre affichage initial de notre vue, nous allons jouer sur le paramètre «loop»

PIXEL ONLINE CREATION

LISTING ARTICLES

```
{% for article in articles %}

    {% if loop.length >= 2 %}
        {% if loop.index == 1 %}
            <div class="container">
                <div class="row" style="height:300px;">
                    <div class="col col-12 col-sm-7 col-md-9" style="color: white;">
                        <div class="h-75 article-card  image-card" style="background: url( {{ asset('build/images/articles/' ~ article.image.chemin) }} );">
                            <h2 class="pl-2 align-text-bottom"><a href="{{ path('article_show', {'id': article.id,'title':article.title}) }}" style="color: white">{{ article.title}}</a></h2>
                            <span class="date pl-2 align-text-bottom">{{ article.publicationDate|date('d/m/Y') }}</span>
                        </div>
                    </div>
        {% elseif loop.index == 2 %}
                    <div class="col col-12 col-sm-5 col-md-3" style="color: white">
                        <div class="h-75 article-card  image-card" style="background: url( {{ asset('build/images/articles/' ~ article.image.chemin) }} );">
                            <h2 class="pl-2 align-text-bottom"><a href="{{ path('article_show', {'id': article.id,'title':article.title}) }}" style="color: white">{{ article.title}}</a></h2>
                            <span class="date pl-2 align-text-bottom">{{ article.publicationDate|date('d/m/Y') }}</span>
                        </div>
                    </div>
                </div>
            </div>
        {% endif %}
    {% endif %}
```

On boucle sur le tableau d'articles.
if(loop.length >= 2) = si au moins deux articles, cela permet d'avoir l'affichage des articles côte à côte.
On affiche les images via assets (yarn dev encore) et ajoute un lien (path) en ciblant la méthode «show»
Pour rappel, nous avons modifié juste avant le controller pour afficher plus facilement l'article.

PIXEL ONLINE CREATION

LISTING ARTICLES

```
    {% if (loop.index == 1 and loop.length < 2) or (loop.index == 2  and loop.length >= 2)  %}
    <div class="container">
        <div class="row">
    {% endif %}

    {% if ((loop.length >= 2 and loop.index > 2) or loop.length < 2) %}
            <div class="col col-12 col-sm-6 col-md-3">
                <div class="article-card">
                    <img src="{{ asset('build/images/articles/' ~ article.image.chemin) }}" class="img-fluid">
                    <h2><a href="{{ path('article_show', {'id': article.id,'title':article.title}) }}">{{ article.title}}</a></h2>
                    <span class="date">{{ article.publicationDate|date('d/m/Y') }}</span>
                </div>
            </div>
    {%  endif %}

    {% if loop.last  %}
        </div>
    </div>
    {% endif %}

{% endfor %}


{% endblock %}      You, il y a 3 jours • ok …
```

On réalise ensuite des conditions afin de bien cibler le nombre minimum d'articles pour l'affichage des éléments restants.

Sauf mise en forme CSS, le reste de la vue étant la même chose que l'affichage des premiers articles.

PIXEL ONLINE CREATION

LISTING ARTICLES

Voici le rendu de la page d'accueil

PIXEL ONLINE CREATION

AFFICHAGE ARTICLE

```
{# templates/blog/show.html.twig #}
{% extends "base.html.twig" %}

{% block title %}{{ article.title}} | {{parent()}}{% endblock %}

{% block content %}
    <article class="container">
        <div class="row">
            <div class="col col-12">
                <h2 class="text-center">{{ article.title }}</h2>
                <span class="text-center d-block">{{ article.publicationDate|date('d/m/Y') }}</span>
                <p class="text-center">
                    <img src="{{ asset('build/images/articles/' ~ article.image.chemin) }}" class="img-fluid center" style="max-width:450px">
                </p>
                <span class="text-center d-block"><b>Catégories : {% for category in article.categories %} {{ category.label }}{{ loop.last ? '' : ',' }} {% endfor %}</b></span>
            </div>
        </div>
        <div class="row">
            <div class="col col-12">
                <p class="text-justify">
                    {{ article.content }}
                </p>
            </div>
    </article>
{% endblock %}      You, 13 days ago • ok
```

PIXEL ONLINE CREATION

AFFICHAGE ARTICLE

Nous pouvons visualiser l'affichage en cliquant sur le lien d'un article. Encore une fois le ParamConverter est utile ici car en l'absence d'un ID valide nous recevrons une ERREUR 404

PIXEL ONLINE CREATION