## I) INTRODUCTION:

Symfony est un framework PHP beaucoup utilisé en entreprise au même titre que Laravel, CakePHP
Créé en 2005 au sein de l'entreprise SensioLabs pour éviter de créer les mêmes fonctionnalité projet après projet ( gestion utilisateur, base de données…) . Il a subit plusieurs évolutions.
Symfony dispose d'une grande communauté . la popularité du framework a fait émerger des plate-forme telle que http://knpbundles.com/ . c'est un site répertoriant tous les bundles créés par des développeur tier . Le bundle est une brique logicielle qui va permettre d'étendre les fonctionnalité du framework ( création d'interface d'administration …). Ce permettra d'éviter de réinventer la "roue".
SensioLabs est créé également une suite d'outils permettant de vérifier en temps réel la qualité du code, la validité des tests unitaires et fonctionnels… https://insight.symfony.com/
https://blackfire.io/ réalise des audits de performance . il analyse le code et met le doigt sur les éventuels problèmes de performance.
Vous trouverez énormément d'informations sur la page de la Symfony Roadmap. https://symfony.com/roadmap
Concrètement cet Roadmap va vous donner des informations sur les différentes versions de Symfony.
C'est assez simple de maintenir facilement son projet pour continuer à développer dessus sans aucun problème et ne jamais être bloqué sur une version antérieure de symfony.
In ne faudra pas hésiter à utiliser la documentation ( https://symfony.com/doc/current/index.html#gsc.tab=0 ). Il y a beaucoup d'exemples, c'est un livre de "recettes"…

Symfony 3 était Symfony 2.8 auquel on avait enlevé les fonctionnalités dépréciées ( obsolète ).
Symfony 4 (nécessite php 7) , c'est Symfony 3.4 auquel on a enlevé les fonctionnalités dépréciées et rajouté une nouvelle manière de développer les applications.

## II) ENVIRONNEMENT DE DEVELOPPEMENT:

- Editeur de texte: Sublime texte
  Avec les extensions:
  - Twig : **PHP-Twig** redemmarrer puis aller dans
    **View** menu → **Syntax** → **Open all with current extension as** → **HTML (Twig)**

  - Importation automatique de namespace: **PHP Companion**
    Puis coller dans **Preference > Key Bindings User (Raccourcis clavier)**

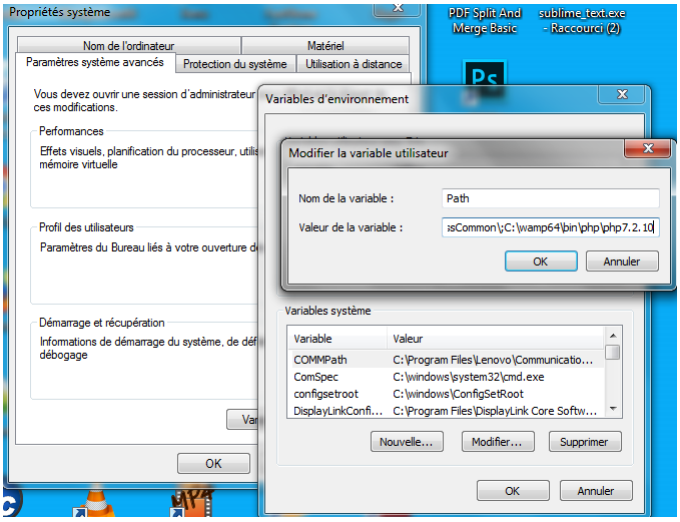
```
{ "keys": ["f6"], "command": "expand_fqcn" },
{ "keys": ["shift+f6"], "command": "expand_fqcn", "args": {"leading_separator": true} },
{ "keys": ["f5"], "command": "find_use" },
{ "keys": ["f4"], "command": "import_namespace" },
{ "keys": ["f3"], "command": "implement" },
{ "keys": ["shift+f12"], "command": "goto_definition_scope" },
{ "keys": ["f7"], "command": "insert_php_constructor_property" }
```

    En se plaçant sur la classe et en appuyant sur F5, cela importera automatiquement la classe correspondante.

- Utiliser un serveur web: (wampserver, laragon … )
  Modules apache à activer:
  - rewrite_module

  Extensions php à activer:
  - intl
  - xmlrpc
  - pdo_mysql
  - pdo_sqlite
  - mbstring

- Un émulateur de terminal ( symfony nécessite php en ligne de commande)
  Si on utilise wampserver, il faut rajouter le chemin de php dans la variable path de windows. ( variables d'environnement)
  Exemple:




Modules nécessaires à la ligne de commande activés dans php.ini

- extension=php_curl.dll
- extension=php_openssl.dll

## III) COMPOSER -GESTIONNAIRE DE DEPENDANCE:

Pour installer Symfony, il existe plusieurs méthodes :

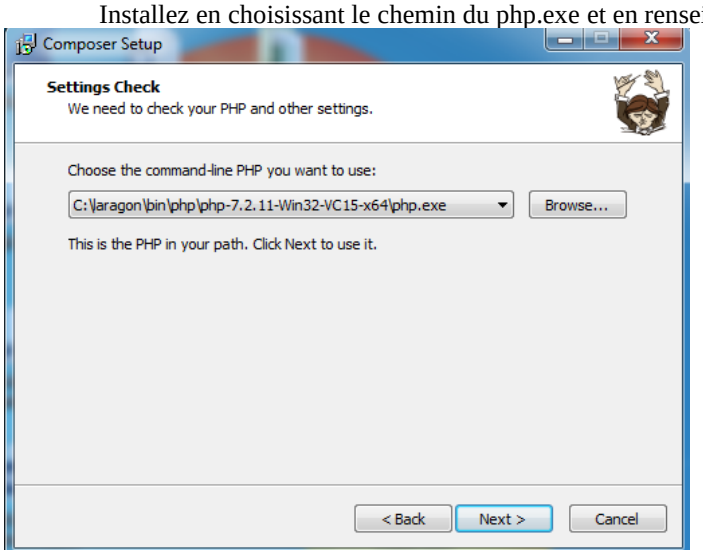
- Installeur en ligne de commande ( mais ne sert qu'à ça)
- **Composer** https://getcomposer.org/ est un gestionnaire de dépendances. Il permet l'installation de librairies PHP en ligne de commande. Apparu en 2012 et nouveau standard en terme de gestion de dépendance.

L'intérêt de **Composer** est qu'il évite de télécharger manuellement toutes les librairies qui peut devenir fastidieux dans le cas par exemple où une librairie dépend d'autres librairies.
L'annuaire https://packagist.org/ permet de savoir si une librairie PHP est disponible via composer .
Exemple: recherche de API Facebook https://packagist.org/packages/facebook/graph-sdk .
Composer permet également de télécharger des Bundles.

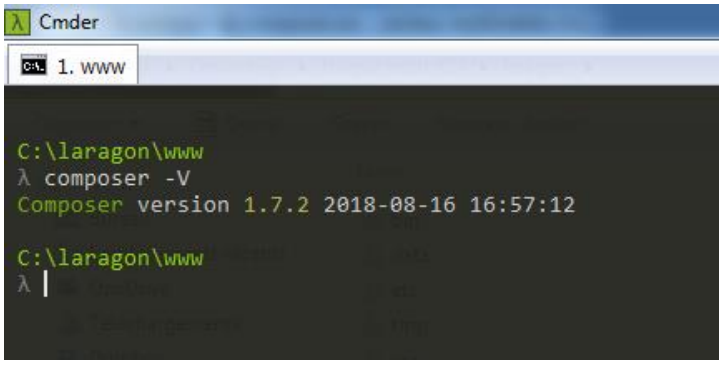
### 1) Installation de composer:

https://getcomposer.org/download/

Pour les utilisateur de windows, il existe un programme d'installation Composer-Setup.exe .
Installez en choisissant le chemin du php.exe et en renseignant éventuellement un proxy.




Vérification du bon fonctionnement :
Dans l'invite de commande lancer : **composer -V**




Mise à jour de **composer** éventuellement: **composer self-update**

Diagnostique composer: **composer diagnose**

Il se peut qu'il y ait un problème de proxy
Exemple dans le terminal:
set http_proxy = 10.40.170.12:8181
set https_proxy = 10.40.170.12:8181

pour supprimer le proxy:
set http_proxy=
set https_proxy=

**Utilisation de composer:**
Quand on utilise **composer**, le cœur de tout son fonctionnement est un fichier **JSON** appelé **composer.json**. C'est dans ce fichier nous allons mentionner nos dépendances et leur version.
Si le fichier **composer.json** n'existe pas **,** on peut initialiser un projet en faisant: **composer init**

- **Ajouter une dépendance**

Pour installer une dépendance, nous avons deux méthodes, en utilisant la commande **composer require** ou en mentionnant notre dépendance dans la section **require** du fichier **composer.json**

- **composer require**

La première méthode consiste à laisser composer lui même écrire dans le fichier **composer.json,** disons que nous voulons installer **swiftmailer** par exemple https://packagist.org/packages/swiftmailer/swiftmailer , pour cela, on écrit:

**composer require swiftmailer/swiftmailer**

en même temps ils nous installe les deux dépendances de **swiftmailer** qui sont **email-validator** et **lexer**

Maintenant dans le fichier **composer.json,** la section **require** a été modifiée pour ajouter **swiftmailer**, mais ce n'est pas le seul changement sur notre projet, un autre fichier **composer.lock** à été créé à la racine du projet.
Un dossier **vendor** a été créé. C'est dans ce dossier **vendor** que **composer** enregistre le code source des dépendances de notre projet.

- **Installer une dépendance en modifiant le fichier composer.json**

La deuxième manière d'ajouter une dépendance, c'est de le faire directement dans le fichier **composer.json,** on va ajouter twig
L'importance d'utiliser cette méthode, c'est de pouvoir spécifier la version que l'on veut nous même.

```
"require":  {
   "nom":  "version"
}
```

```
…….
        "minimum-stability": "stable",
           "require": {
              "swiftmailer/swiftmailer": "^6.1",
              "twig/twig": "^2.6"
           }
        }
……..
```

Ensuite, on fait: **composer update**

- **Plage de versions**

Vous pouvez utiliser les opérateurs de comparaison **<, >, <=, >=** et **!=** pour spécifier la version de la dépendance. Vous pouvez aussi utiliser les opérateurs logique **AND** qui est représenté par un espace ou une virgule et **OR** qui est représenté par ||. Nous pouvons par exemple écrire:

```
>=1.0
>=1.0 <=2.0
>=1.0 <1.1 || >=1.2
```

- **Le trait d'union**

Le trait d'union va nous permettre de spécifier une plage de version. Nous pouvons par exemple écrire **1.0 - 2.0** ce qui équivaut à **>=1.0.0 <2.1.** et **1.0.0 - 2.1.0** équivaut quant à lui à **>=1.0.0 <=2.1.0.**

- **Le joker**

En utilisant un caractère générique, vous pouvez spécifier un pattern. Par exemple 1.0.* inclura toutes les versions au dessus en partant de 1.0.0, puis 1.0.1 et ainsi de suite, mais pas 1.1, cela équivaut donc à **>=1.0.0 <1.1**

- **Le tilde**

Le tilde est idéal pour cibler une version minimale requise et pour autoriser toute nouvelle version mineure, mais pas la version majeure suivante. Si vous écrivez ~1.4, vous permettez alors toute les versions au dessus de 1.4 mais pas les versions 2.0. Cela équivaut donc à **>=1.4 <2.0**

- **L'accent circonflexe**

L'accent circonflexe est destiné à permettre toutes les mises à jour sans changement majeur. Si un projet suit un versionnage sémantique, il ne devrait y avoir aucune amélioration qui rompt la compatibilité au sein d'une branche majeure. Si vous écrivez donc ^1.4.5, vous permettez toutes les versions au-dessus mais n'incluant pas la version 2.0.

- **Le require-dev**

Composer nous permet de spécifier des dépendances que nous allons utiliser en développement. Pour cela, il faut écrire ces dépendances dans la section **require-dev** au lieu de **require**

**Exemple de mise à jour mineure de symfony**

Exemple , on veut passer de la version 4.2.7 à la version 4.4
On remplace dans composer.json

```
"extra": {
   "symfony": {
      "allow-contrib": false,
      "require": "4.2.*"
   }
}
```

Par:

```
"extra": {
   "symfony": {
      "allow-contrib": false,
      "require": "4.4.*"
   }
}
```

Passer éventuellement les dépendances: **de 4.2.* à 4.4.***

Et on fait: **composer update "symfony/*"**

Vous pouvez également mettre à niveau le reste de vos bibliothèques. Si vous avez fait du bon travail avec vos contraintes de version dans composer.json , vous pouvez le faire en toute sécurité en exécutant:

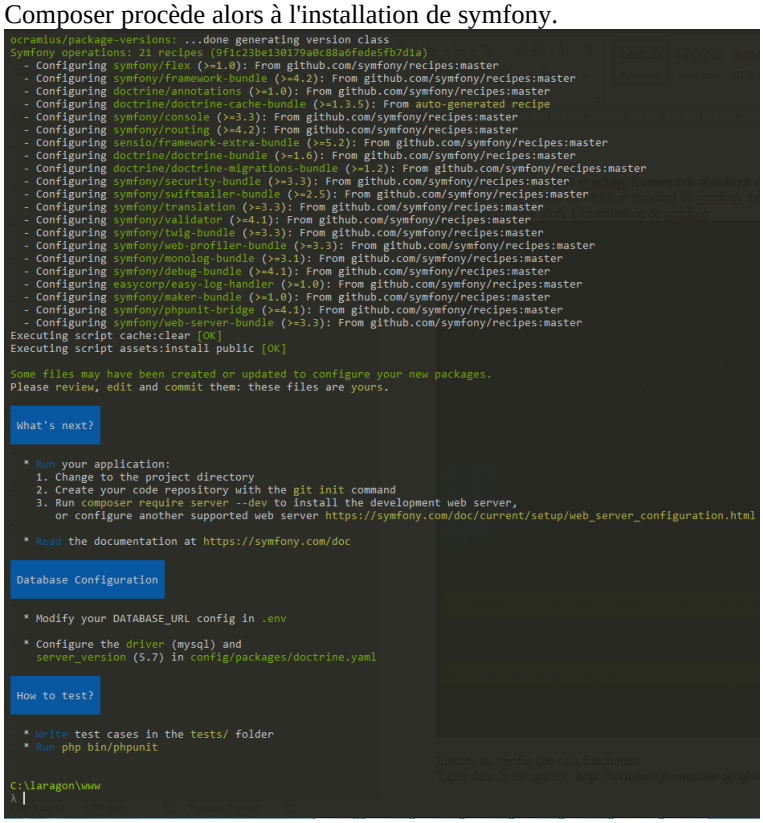
**composer update**

## IV) INSTALLATION DE SYMFONY VIA COMPOSER:

On va dans https://symfony.com/doc/current/setup.html
A l'aide de l'invite de commande aller dans le dossier www et taper:
**composer create-project symfony/website-skeleton formation-symfony4** (dernière version de symfony)
( on créé un projet basé website-skeleton symfony dans le dossier formation-symfony4 )

On peut également choisir la version de symfony que l'on veut:
**composer create-project symfony/website-skeleton formation-symfony4 "^4.4"**

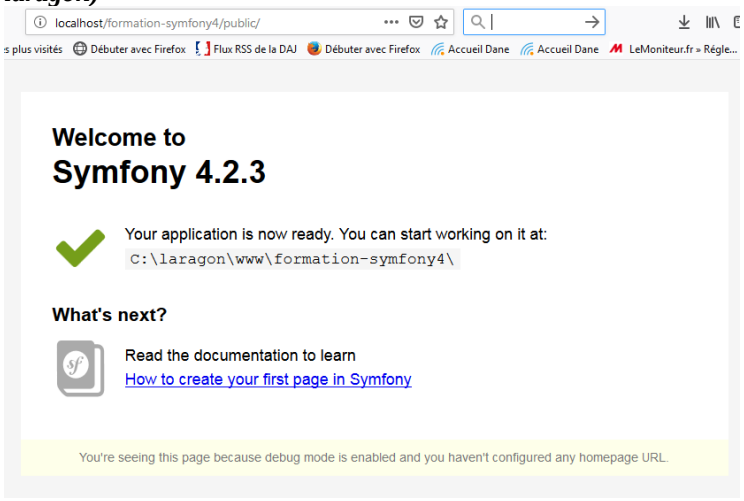
Composer procède alors à l'installation de symfony.




Ensuite on vérifie que cela fonctionne:
On va installer le serveur de symfony
En faisant dans le **dossier créé** (par exemple formation-symfony4 ):

- ⇨ **composer require server --dev**
- ⇨ **php bin\console server:run**
- ⇨ http://127.0.0.1:8000 (on lance sur le port 8000 et non sur le port 80)

Ou en tapant dans le navigateur: ***http://localhost/formation-symfony4/public/*** ***ou*** ***http://formation-symfony4.test/*** ***(vhost laragon)***

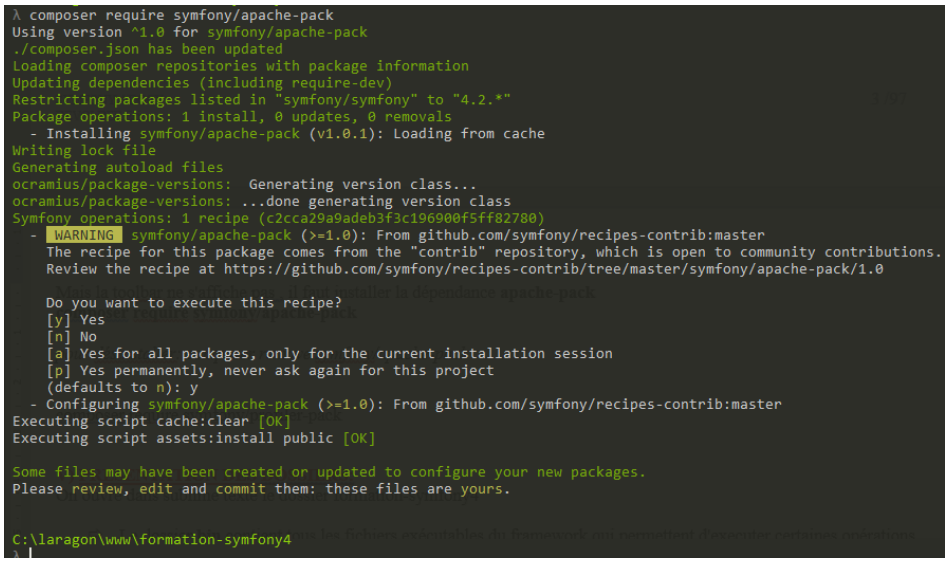


**ATTENTION:** Avec la version 5 de symfony , l'installation se fait différemment. https://symfony.com/download
On peut néanmoins faire:

- **composer create-project symfony/website-skeleton formation-symfony4**
- **composer require symfony/web-server-bundle 4.4**

Mais

la toolbar de débug ne s'affiche pas , il faut installer la dépendance *apache-pack*
**composer require symfony/apache-pack**



*Rem: pour désinstaller: composer remove symfony/apache-pack*

## V) PRESENTATION DES DOSSIERS:

On ouvre dans sublime texte le dossier formation-symfony4

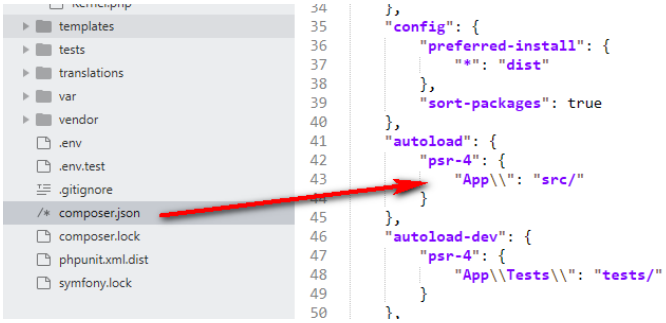
⇨ Le dossier **bin** contient tous les fichiers exécutables du framework qui permettent d'executer certaines opérations. Le fichier console est l'outil en ligne de commande de symfony , on peut voir la liste des commandes dans la console de cette façon:
***php bin\console*** ou ***php bin\console list*** => *ensemble des commandes disponibles*

Il y a également **phpunit** qui permet de lancer les tests unitaires.

⇨ Le dossier **config** contient la configuration du projet

⇨ Le dossier **public** est la racine du serveur web ( il faut faire pointer directement l'hébergement sur ce dossier)

⇨ Le dossier **src** contient le code php de l'application. Il correspond au namespace App. C'est directement spécifié au niveau de l'autoloader de composer. Le namespace App pointe vers le dossier src




⇨ Le dossier **template** contient les vues de l'application. Ce n'est pas du PHP mais du twig

⇨ Le dossier **tests** qui contient les tests unitaires et les tests fonctionnels

⇨ Le dossier **translations** qui contient les traductions pour le site

⇨ Le dossier **var** qui contient les fichiers caches et les logs

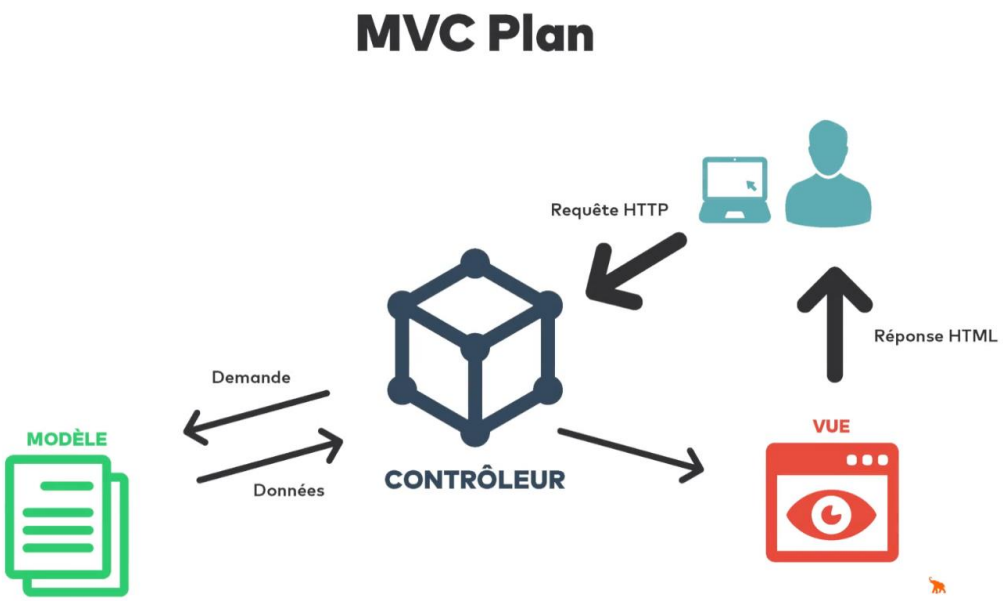
⇨ Le dossier **vendor** qui est propre à composer

On touche en général les dossiers config, src et les templates

Comme spécifier lors de l'installation le fichier **.env** ( variables d'environnement) contient le chemin vers la base de donnée. Par défaut , on a:
**DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name**

Que l'on va remplacer par exemple par:
**DATABASE_URL=mysql://root:@127.0.0.1:3306/blog**

## VI) LE DESIGN PATTERN MVC:



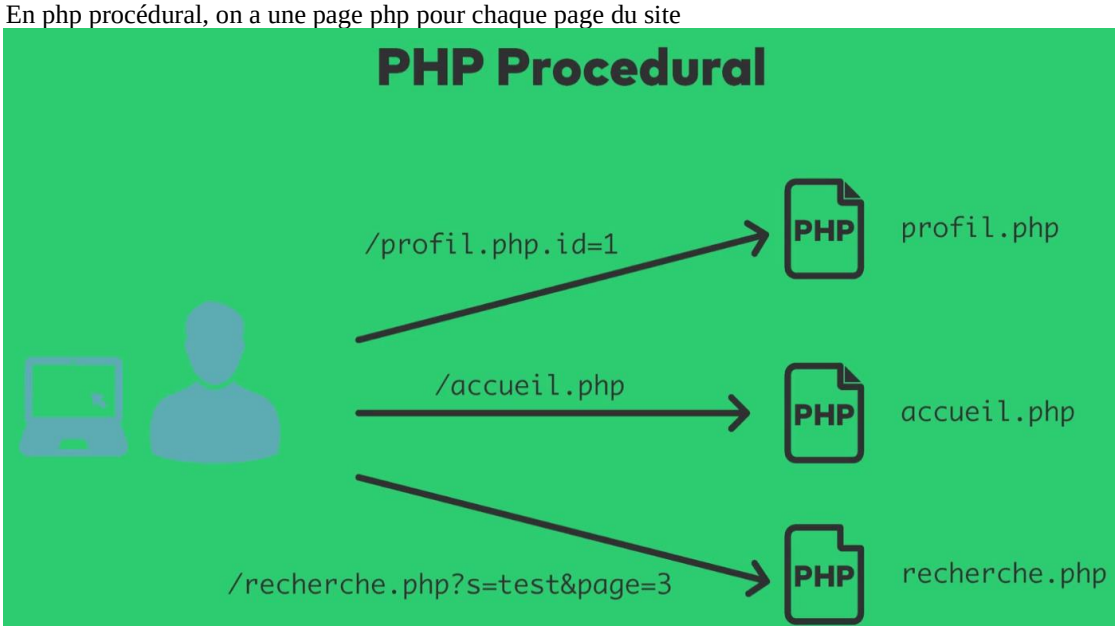

- Le modèle s'occupe de toutes les interactions avec la base de donnée. Il vérifie également l'intégrité des données.
- La vue s'occupe de tout l'aspect visuel du site. Elle permet de définir des gabarit de page HTML dans lesquels on va glisser des données dynamiques.
- Le contrôleur contient toute la logique de l'application. Il fait appelle à la fois au modèle et à la vue pour restituer une réponse

De cette façon, on aura un code plus structuré, des éléments réutilisables et une meilleure lisibilité .

## VII) APPROCHE PHP PROCEDURALE ET SYMFONY:

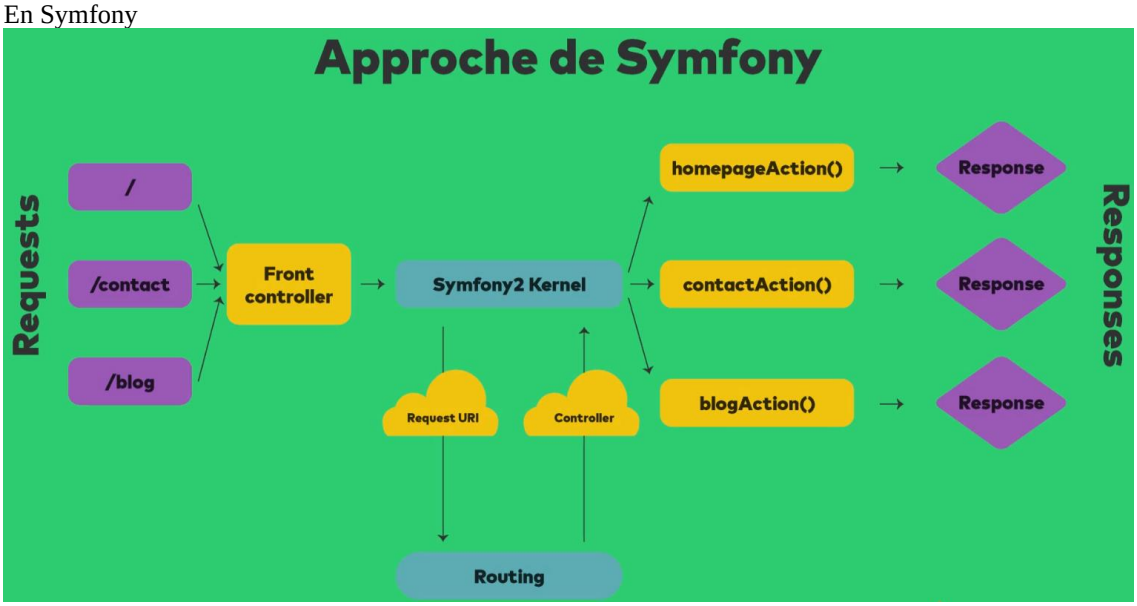
Différence entre l'approche PHP procédural et symfony dans le traitement d'une requête

- En php procédural, on a une page php pour chaque page du site




Dans cette approche , on a du php mélangé à du HTML

- En Symfony

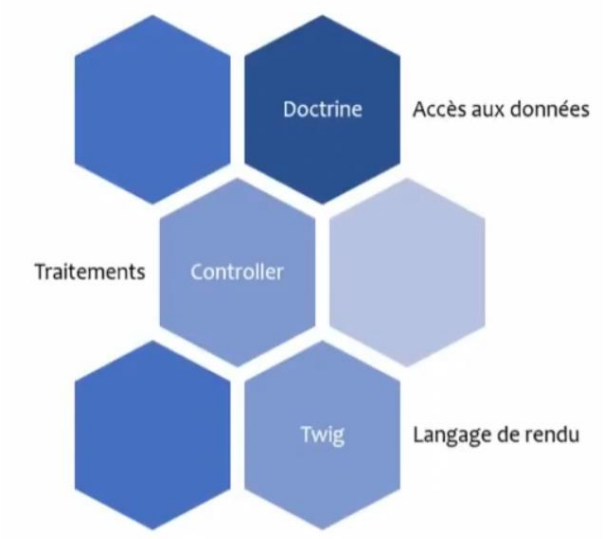



Lorsque l'utilisateur envoi une requête sur le site, celle passe par le front Controller qui est représenté par le fichier **app.php** . Le front Controller va instancier le noyau symfony qui va récupérer l'url de la requête et la transmettre au routeur qui fait la correspondance entre une url et un controller. Le noyau symfony exécute ensuite le contrôleur et lui transmet la requête. Le controleur traite la requête et éventuellement utilise la couche modèle et vue et retourne une réponse .

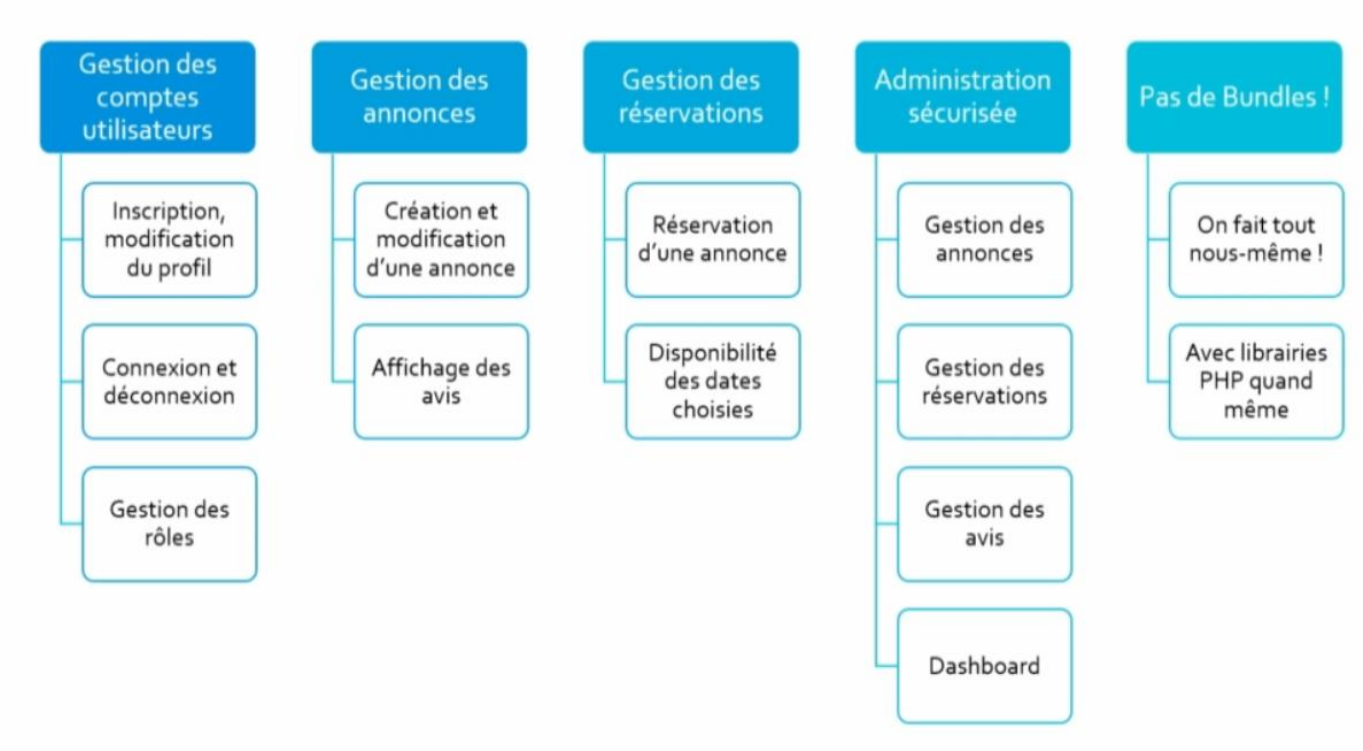
L'application sera beaucoup plus maintenable et évolutive.

## VIII) LES TROIS PILIERS DE SYMFONY:

- ⇨ Les controllers : gérer les traitements
  Ils écoutent une adresse (une route) , écouter et analyser la requête http que la navigateur a envoyé à la route, fabriquer une réponse http et la renvoyer au navigateur
- ⇨ Doctine: Gère l'acces aux données
- ⇨ Twig: gère l'affichage les rendus



## IX) CREATION PROJET:




Création du premier controlleur: **php bin/console make:controller** => HomeController et on met la route à la racine "/"

Deux fichiers ont été créés:

- ⇨ src/Controller/HomeController.php
- ⇨ templates/home/index.html.twig

**HomeController.php:**

```
namespace App\Controller;

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

class HomeController extends AbstractController
{
    /**
     * @Route("/", name="homepage")
     */
    public function index()
    {
        return $this->render('home/index.html.twig', [
            'controller_name' => 'HomeController',
        ]);
    }
}
```

On voit:

- ⇨ une classe HomeController qui hérite de AbstractController
- ⇨ une annotation: @Route("/", name="homepage")
  Quand le navigateur appellera monsite.com/ voici la fonction que tu dois appeler => index()

**index.html.twig**

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

{% block title %}Hello BlogController!{% endblock %}

{% block body %}
<style>
    .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
    <h1>Hello {{ controller_name }}! ✅</h1>

    This friendly message is coming from:
    <ul>
        <li>Your controller at <code><a href="{{ 'C:/laragon/www/formation-symfony4/src/Controller/BlogController.php'|file_link(0) }}">src/Controller/BlogController.php</a></code></li>
        <li>Your template at <code><a href="{{ 'C:/laragon/www/formation-symfony4/templates/blog/index.html.twig'|file_link(0) }}">templates/blog/index.html.twig</a></code></li>
    </ul>
</div>
{% endblock %}
```

**Vérification du fonctionnement:**

http://127.0.0.1:8000/home ou http://localhost/formation-symfony4/public/home

Pour créer des pages, il faut donc créer une fonction public, la relier avec une route et faire en sorte que cette fonction nous réponde un affichage ( une réponse http)

## X) LANGAGE TWIG:

Il apporte de la simplicité et beaucoup de fonctionnalité ( boucles et conditions facilités, filtres …)
Il y a une absence complète de PHP au sein de l'affichage.
Twig est une librairie https://twig.symfony.com/ qui l'on peut utiliser aussi en dehors de symfony

### 1) Afficher le contenu d'une variable:

**{{ title }}**

### 2) Utilisation de commande:

**{% cmd %}**

```
Ex: {% if age > 18 %}
        <p>tu es majeur</p>
    {% else %}
        <p>tu es mineur</p>
    {% endif %}
```

Il faut que le controller transmette à twig le contenu des variables. Ce qui est fait grâce à un tableau.

```php
/**
 * @Route("/", name="home")
 */
public function home()
{
    return $this->render('blog/home.html.twig',[
        'title' => 'Bienvenue dans ce blog',
        'age' => '31',

    ]);
}
```

### 3) Une structure for:

https://twig.symfony.com/doc/2.x/tags/if.html

Ex: si dans le controller on a:

```
/**
 * @Route("/", name="home")
 */
public function home()
{

    $prenom = ["Eric" => 52, "Fabien" => 32, "Anne" => 25];

    return $this->render('blog/home.html.twig',[
        'title' => 'Bienvenue dans ce blog',
        'age' => '31',
        'tableau' => $prenom,

    ]);
}
```

Dans home.html.twig

```
{% for prenom in tableau %}
        <p>{{ prenom }}</p>
{% endfor %}


{% for prenom, age in tableau %}   => equivalent du foreach dans php
        <p>{{ prenom }} a {{ age }} ans </p>
{% endfor %}


{% for compteur in 0..10 %}
        <li>{{ compteur }}</li>
{% endfor %}
```

Au sein d'une boucle for , on a une variable **loop:**

https://twig.symfony.com/doc/2.x/tags/for.html

loop.index0 représente l'index ou nous nous trouvons:

loop.first: vrai pour la première itération

```
{% for prenom, age in tableau %}
        <p>numéro iteration: {{ loop.index0 }}{{ prenom }} a {{ age }} ans </p>
{% endfor %}


{% for prenom, age in tableau %}
        {% if loop.first %}
                <strong>{{ prenom }} a {{ age }} ans </strong>
        {% else %}
                <p>{{ prenom }} a {{ age }} ans </p>
        {% endif %}
{% endfor %}
```

### 4) Les commentaires:

```
{#
{% for prenom, age in tableau %}
        <p>numéro iteration: {{ loop.index0 }}{{ prenom }} a {{ age }} ans </p>
    {% endfor %}
#}
```

5) **Les filtres:**

https://twig.symfony.com/doc/2.x/filters/index.html

Permet de formater la façon dont une variable va s'afficher au sein du code: syntaxe donnee | filtre

{{ prenom | upper}} => met en majuscule

{{ texte | raw}} => affiche le texte sans protection ( permet donc l'interprétation du html)

6) **Commande path:**

https://symfony.com/doc/current/reference/twig_reference.html#path

génère une URL en fonction du nom d'une route et des paramètres

Ex: si on a une route: * @Route("/ads/{slug}", name="ads_show")

path sera de la forme: {{ path('ads_show', {'slug': ad.slug}) }} car la route dépend du slug

7) **Créer et utiliser une variable Twig:**

Exemple si on a dans notre code à plusieurs endroits {{ path('ads_show', {'slug': ad.slug}) }} , il faut éviter de se répéter.

On va factoriser le code ( créer un code unique que l'on peut appeler plusieurs fois)

On pourra définir une variable :

{% set url= path('ads_show', {'slug': ad.slug}) %} que l'on appellera en faisant {{ url }}

8) **Inclusion de fichier:**

Lorsque les fichiers commencent à être touffus , on peut les répartir dans des fichiers externes et ensuite y faire appel: {% include 'partials/header.html.twig' %}

On peut également inclure des variables:

{% include 'partials/rating.html.twig' with {'rating': ad.getAvgRatings} %}

## XI) RENDU GLOBAL DU SITE:

On utilise le template de base **base.html.twig (gabarit général)** .

Si je veux que quelque-chose se retrouve dupliqué sur toutes les pages, j'ai intérêt à la mettre dans ce template de base. Puis à chaque fois que je créé une autre page, je dirais qu'elle s'inscrit dans ce template.

Dans **base.html.twig** , on remarque qu'il y a différents blocs (block title, block stylesheets, block body, block javascripts )

Les blocs sont des emplacements que toutes les autres pages pourront personnaliser

On choisit un thème bootstrap https://bootswatch.com/ (par exemple Litera) et on copie colle le bootstrap.min.css dans **public>css>bootstrap.min.css**

On récupère également le javascript de bootstrap: https://getbootstrap.com/docs/4.4/getting-started/download/

https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js que l'on copie/colle dans **public>js>bootstrap.min.js**

On récupère également jQuery et popper:

https://code.jquery.com/jquery-3.3.1.slim.min.js que l'on copie/colle dans **public>js>jquery.min.js**

https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js que l'on copie/colle dans **public>js>popper.min.js**

On modifie base.html.twig en rajoutant également la barre nav:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>{% block title %}Welcome!{% endblock %}</title>
   <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css">
   {% block stylesheets %}{% endblock %}
   </head>
   <body>
      <nav class="navbar navbar-expand-lg navbar-light bg-light">
         <a class="navbar-brand" href="{{ path('homepage') }}">Site Annonces</a>
         <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor03" aria-
controls="navbarColor03" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
         </button>
```

```
    <div class="collapse navbar-collapse" id="navbarColor03">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
          <a class="nav-link" href="{{ path('homepage') }}">Accueil <span class="sr-only">(current)</span></a>
        </li>

      </ul>
      <ul class="navbar-nav ml-auto">
        <li class="nav-item active">
          <a class="nav-link" href="#">Mon compte<span class="sr-only">(current)</span></a>
        </li>

      </ul>

    </div>
  </nav>

  {% block body %}{% endblock %}

    <script type="text/javascript" src="/js/jquery.min.js"></script>
    <script type="text/javascript" src="/js/popper.min.js"></script>
    <script type="text/javascript" src="/js/bootstrap.min.js"></script>

    {% block javascripts %}{% endblock %}
    </body>
    </html>
```

Pour plus de simplicité et de clarté, on va répartir le contenu dans des fichiers externes.
On créé dans deux fichiers:

**/templates/partials/header.html.twig** (on mets ce qui concerne la navigation)
**/templates/partials/footer.html.twig**

Le fichier base.html.twig devient alors:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>{% block title %}Welcome!{% endblock %}</title>
  <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css">
  {% block stylesheets %}{% endblock %}
  </head>
  <body>

    {% include 'partials/header.html.twig' %}

    {% block body %}{% endblock %}


      {% include 'partials/footer.html.twig' %}


    <script type="text/javascript" src="/js/jquery.min.js"></script>
    <script type="text/javascript" src="/js/popper.min.js"></script>
    <script type="text/javascript" src="/js/bootstrap.min.js"></script>

    {% block javascripts %}{% endblock %}
  </body>
</html>
```

## XII) MODIFICATION PAGE D'ACCUEIL:

**Index.html.twig**:

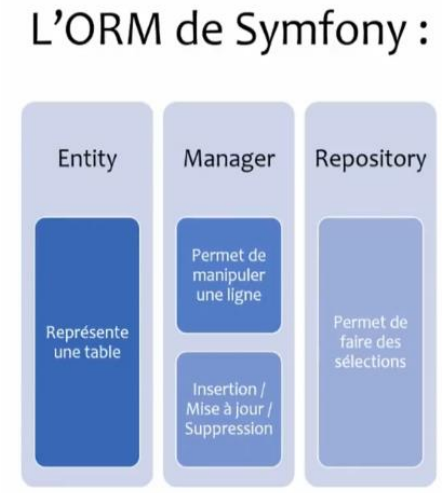
```
{% extends 'base.html.twig' %}

{% block title %}Hello HomeController!{% endblock %}

    {% block body %}

        <div class="container mt-3">
            <div class="jumbotron">
                <h1 class="display-4">Bienvenue sur le site d'annonces</h1>
                <p class="lead">Mise en relation voyageurs et propriétaires</p>
                <hr class="my-4">
                <p>Logez-vous lors de vos déplacements</p>
                <a class="btn btn-primary btn-lg" href="#" role="button">Voir Nos annonces</a>
            </div>

            <h2>Nos Appartements stars</h2>

            <div class="row mt-3 mb-3">

                <div class="col-4">

                    <div class="card text-center">
                        <div class="card-header">
                            2 chambres 50€/nuit<br/>
                            Pas encore noté
                        </div>
                        <div class="card-body">
                            <img src="https://via.placeholder.com/350" style="height:200px;width:100%;display:block">
                            <div class="text-left">
                                <h4>Bel Appartement</h4>
                                <p>
                                    Post quorum necem nihilo lenius ferociens Gallus ut leo cadaveribus pastus multa huius modi scrutabatur. quae singula narrare non refert, me professione modum, quod evitandum est, excedamus.
                                </p>
                                <a href="#" class="btn btn-primary ">Go somewhere</a>
                            </div>
                        </div>

                    </div>

                </div>

                <div class="col-4">

                    <div class="card text-center">
                        <div class="card-header">
                            2 chambres 50€/nuit<br/>
                            Pas encore noté
                        </div>
                        <div class="card-body">
                            <img src="https://via.placeholder.com/350" style="height:200px;width:100%;display:block">
                            <div class="text-left">
                                <h4>Bel Appartement</h4>
                                <p>
                                    Post quorum necem nihilo lenius ferociens Gallus ut leo cadaveribus pastus multa huius modi scrutabatur. quae singula narrare non refert, me professione modum, quod evitandum est, excedamus.
                                </p>
                                <a href="#" class="btn btn-primary ">Go somewhere</a>
                            </div>
                        </div>
```

```
            </div>

        </div>

        <div class="col-4">

            <div class="card text-center">
                <div class="card-header">
                    2 chambres 50€/nuit<br/>
                    Pas encore noté
                </div>
                <div class="card-body">
                    <img src="https://via.placeholder.com/350" style="height:200px;width:100%;display:block">
                    <div class="text-left">
                        <h4>Bel Appartement</h4>
                        <p>
                            Post quorum necem nihilo lenius ferociens Gallus ut leo cadaveribus pastus multa huius modi scrutabatur. quae singula narrare non refert, me professione modum, quod evitandum est, excedamus.
                        </p>
                        <a href="#" class="btn btn-primary ">Go somewhere</a>
                    </div>
                </div>

            </div>

        </div>

    </div>

    <h2>Nos Propriétaires stars</h2>
    <div class="row mt-3 mb-4">

        <div class="col-6">
            <div class="card ">
                <div class="card-header">
                    Eric
                </div>
                <div class="card-body">
                    <div class="card-text">
                        <img src="https://via.placeholder.com/64" class="float-left mr-3">
                        <p>With supporting text below as a natural lead-in to additional content.</p>
                    </div>
                    <div class="text-right">
                        <a href="#" class="btn btn-primary">Go somewhere</a>
                    </div>
                </div>
            </div>
        </div>

        <div class="col-6">
            <div class="card ">
                <div class="card-header">
                    Eric
                </div>
                <div class="card-body">
                    <div class="card-text">
                        <img src="https://via.placeholder.com/64" class="float-left mr-3">
                        <p>With supporting text below as a natural lead-in to additional content.</p>
                    </div>
                    <div class="text-right">
                        <a href="#" class="btn btn-primary">Go somewhere</a>
                    </div>
                </div>
            </div>
```

```
            </div>


        </div>

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

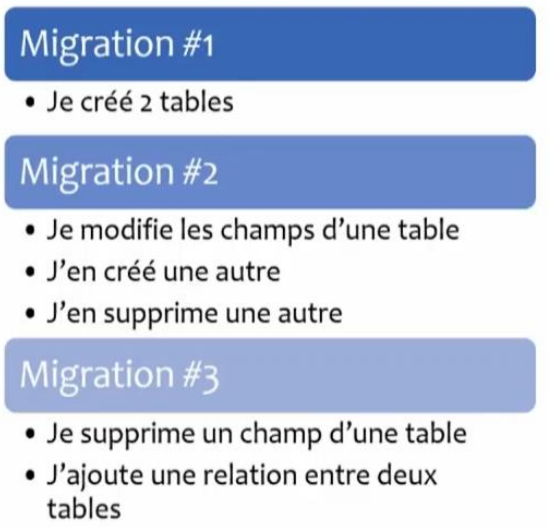
## XIII) ORM DOCTRINE ( Symfony et les bases de données):

Symfony utilise un ORM ( object relational mappy). C'est une brique logiciel qui fait le lien entre une application ( php, java ..) et une base de donnée. On utilisera des classes dans l'application et l'ORM se chargera de faire en sorte que les manipulations faites avec les objets se reflètent dans la base de donnée. L'ORM de Symfony s'appelle Doctrine.
Dans l'application Symfony on va créer des classes que l'on appellent des "**Entités"** et qui représente des tables.
Doctrine donne aussi un **Manager** qui sert à manipuler des lignes ( écrire une ligne dans la base, mettre à jour, ou supprimer une ligne dans la base ).
Doctrine nous offre un **Repository** qui sert à faire des sélection de données
Et tout cela sans avoir à écrire de SQL. C'est Doctrine qui s'en charge.



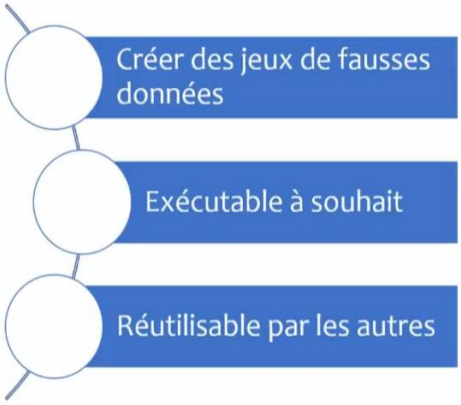

### 1) LES MIGRATIONS:

Dans Symfony , on utilise des **migrations:**
https://symfony.com/doc/master/bundles/DoctrineMigrationsBundle/index.html
La philosophie de Symfony est de privilégier les fichiers car les fichiers sont partagés entre différents développeurs. La base de données doit donc venir de fichiers. Il faut que les fichiers expriment à quoi ressemble la base de données.
Une migration de Symfony est un script qui nous dit, je veux faire passer la base de données d'un état A à un état B.
Exemple:



### 2) LES FIXTURES:

C'est un script qui créé des faux jeux de données au sein de la base de données
https://symfony.com/doc/master/bundles/DoctrineFixturesBundle/index.html




En résumé:
La base de données se créé via un script
Les tables se créent via un script
On créé des jeux de fausses données via un script

## XIV) CREATION BASE DE DONNEE:

Fichier **.env:**
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name =>
**DATABASE_URL=mysql://root:@127.0.0.1:3306/annonces-symfony**

**php bin/console doctrine:database:create**

## XV) CREATION ENTITE ANNONCES:

Création d'une entité:
**php bin/console make:entity** *(on créé une entité qui représente une table)*

On a la création de deux fichiers:

- created: src/Entity/Ad.php *( représente la table des articles)*
- created: src/Repository/AdRepository.php *(permet de faire des sélections sur les donnée de cette table)*

On crée ensuite les propriétés:

Entity Ad
title string 255 no
slug string 255 no
price float no
introduction text no
content text no
coverImage string 255 no
rooms integer no

Mais la table n'est toujours pas créée
Maintenant, il faut créer la migration qui analyse le code (les entités) et vérifie tout ce qui devrait exister dans la base.
Doctrine va comparer avec la base existante et va créer un script SQL pour mettre à jour.

**php bin/console make:migration**
*( On lance les scripts de migrations afin de mettre la base à jour)*
**php bin/console doctrine:migrations:migrate**

Et voilà la table est créée

## XVI) JEUX DE FAUSSES DONNEES (FIXTURE):

On va insérer un jeu de fausses données dans la base.
Il faut installer le composant de création de fixture (que pour le développement)

**composer require orm-fixtures --dev**

**php bin/console make:fixtures**

Il a créé un fichier **src/DataFixtures/AppFixtures.php**

**AppFixtures.php**

Dès que l'on utilise le nom d'une classe, ne pas oublier le "use" pour expliquer à php d'où vient la classe Ad

```php
<?php

namespace App\DataFixtures;

use App\Entity\Ad;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Common\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {

    for ($i=1; $i <30 ; $i++) {


        $ad= new Ad();

        $ad->setTitle("Titre de l'annonce n°$i")
           ->setSlug("titre-de-l-annonce-n-$i")
           ->setCoverImage("https://via.placeholder.com/350")
           ->setIntroduction("C'est une introduction")
           ->setContent("<p>Je suis le contenu</p>")
           ->setPrice(mt_rand(40,200))
           ->setRooms(mt_rand(1,5));

         $manager->persist($ad);
        $manager->flush();

    //mise a jour slug avec l'id créé pour rendre unique le slug
        $slug2 = $ad->getSlug().'_'.$ad->getId();
       $ad->setSlug($slug2);

     $manager->persist($ad);
      $manager->flush();

        }


    }
}
```

**php bin/console doctrine:fixtures:load** ( charge toutes les fixtures dans la base)

## XVII) CREER DES SLUGS FACILEMENT AVEC SLUGIFY:

https://github.com/cocur/slugify

**composer require cocur/slugify**

On modifie **AppFixtures.php**

```php
<?php

namespace App\DataFixtures;

use App\Entity\Ad;
use Cocur\Slugify\Slugify;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Common\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {

        $slugify = new Slugify();
        $title= "Titre de l'annonce n°";
        $slug=$slugify->slugify($title);

    for ($i=1; $i <30 ; $i++) {


        $ad= new Ad();

        $ad->setTitle("Titre de l'annonce n°$i")
           ->setSlug("$slug$i")
           ->setCoverImage("https://via.placeholder.com/350")
           ->setIntroduction("C'est une introduction")
           ->setContent("<p>Je suis le contenu</p>")
           ->setPrice(mt_rand(40,200))
           ->setRooms(mt_rand(1,5));

          $manager->persist($ad);
           $manager->flush();

         //mise a jour slug avec l'id créé pour rendre unique le slug
           $slug2 = $ad->getSlug().'_'.$ad->getId();
           $ad->setSlug($slug2);


          $manager->persist($ad);
           $manager->flush();

       }


    }
}
```

## XVIII) CYCLE DE VIE DES ENTITES:

On peut la gérer directement au niveau de l'entité. Les entités ce sont des classes, donc on va pouvoir faire persister ou supprimer les objets que l'on utilise grâce au manager d'entités. chaque entité subit un cycle de vie ( création, mise à jour et suppression). Pour chaque des actions , l'entité peut régir à ce que l'on appelle des évènements de cycle de vie
https://www.doctrine-project.org/projects/doctrine-orm/en/latest/reference/events.html#lifecycle-events
Ce sont des évènements qui vont être déclenchés à chaque moment de la vie d'une entité
Ex: preRemove, postRemove. Est ce que j'ai une action à faire avant ou après avoir effacer une entité
Exemple dans notre cas, on peut faire en sorte que à chaque fois que l'on cherche à sauvegarder une entité de type annonce, il faut qu'elle vérifie elle-même qu'elle a un slug, si elle n'en a pas, il faut qu'elle le calcule elle-même.

On va dans src/Entity/ad.php et on déclare au -dessus de la classe que l'on va soumettre notre entité à des évènements de cycle de vie. On aura plus besoin de gérer soit -même les slugs
use Doctrine\ORM\Mapping\HasLifecycleCallbacks;
use Cocur\Slugify\Slugify;

```
/**
 * @ORM\Entity(repositoryClass="App\Repository\AdRepository")
 * @ORM\HasLifecycleCallbacks
 */
class Ad
{
   /**
....

/**
 * permet d'initialiser le slug ( on indique à doctrine que cette fonction doit être appelée lors de la mise à jour ou de la création)
 * @ORM\PrePersist
* @ORM\PreUpdate
*/
public function initializeSlug(){

// au moment de la mise à jour
   if (empty($this->slug)){

       $slugify= new Slugify();
       $this->slug = $slugify->slugify($this->title);

   }


}
```

Et maintenant, on peut supprimer tout ce qui concerne slugify dans notre fixture par exemple

### Liste des évènements de cycle de vie

Il existe plusieurs évènements de cycle de vie avec Doctrine. On parle d'écouter un évènement.

| Évènement | Description |
|---|---|
| PrePersist | L'évènement PrePersist se produit juste avant que l'EntityManager ne persiste effectivement l'entité. Concrètement, cela exécute le callback juste avant un $em->persist($entity). Il ne concerne que les entités nouvellement créées. Du coup, il y a deux conséquences : d'une part, les modifications que vous apportez à l'entité seront persistées en base de données, puisqu'elles sont effectives avant que l'EntityManager n'enregistre l'entité en base. D'autre part, vous n'avez pas accès à l'id de l'entité si celui-ci est autogénéré, car justement l'entité n'est pas encore enregistrée en base de données, et donc l'id pas encore généré. |
| PostPersist | L'évènement postPersist se produit juste après que l'EntityManager ait effectivement persisté l'entité. Attention, cela n'exécute pas le callback juste après le $em->persist($entity), mais juste après le $em->flush(). À l'inverse du prePersist, les modifications que vous apportez à l'entité ne seront pas persistées en |

| | |
|---|---|
| | base (mais seront tout de même appliquées à l'entité, attention) ; mais vous avez par contre accès à l'id qui a été généré lors du flush(). |
| PreUpdate | L'évènement preUpdate se produit juste avant que l'EntityManager ne modifie une entité. Par modifiée, j'entends que l'entité existait déjà, que vous y avez apporté des modifications, puis un $em->flush(). Le callback sera exécuté juste avant le flush(). Attention, il faut que vous ayez modifié au moins un attribut pour que l'EntityManager génère une requête et donc déclenche cet évènement.<br>Vous avez accès à l'id autogénéré (car l'entité existe déjà), et vos modifications seront persistées en base de données. |
| PostUpdate | L'évènement postUpdate se produit juste après que l'EntityManager a effectivement modifié une entité. Vous avez accès à l'id et vos modifications ne sont pas persistées en base de données. |
| PreRemove | L'évènement PreRemove se produit juste avant que l'EntityManager ne supprime une entité, c'est-à-dire juste avant un $em->flush() qui précède un $em->remove($entite). Attention, soyez prudents dans cet évènement, si vous souhaitez supprimer des fichiers liés à l'entité par exemple, car à ce moment l'entité n'est pas encore effectivement supprimée, et la suppression peut être annulée en cas d'erreur dans une des opérations à effectuer dans le flush(). |
| PostRemove | L'évènement PostRemove se produit juste après que l'EntityManager a effectivement supprimé une entité. Si vous n'avez plus accès à son id, c'est ici que vous pouvez effectuer une suppression de fichier associé par exemple. |
| PostLoad | L'évènement PostLoad se produit juste après que l'EntityManager a chargé une entité (ou après un $em->refresh()). Utile pour appliquer une action lors du chargement d'une entité. |

## XIX) CREATION DE L'ENTITE IMAGE (relation entre entités):

Elle va servir à représenter l'image que l'on rajoute dans l'annonce (image couverture, images de l'appartement..)
Une annonce peut avoir une liaison avec plusieurs images et une image peut avoir une seule annonce

**php bin/console make:entity Image**

url string 255 no
caption  string 255 no
ad relation Ad ManyToOne no yes images yes

**php bin/console make:migration**

**php bin/console doctrine:migrations:migrate**

On peut maintenant modifier la fixture pour rajouter des images

```php
<?php
namespace App\DataFixtures;

use App\Entity\Ad;
use App\Entity\Image;
use Cocur\Slugify\Slugify;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Common\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
   public function load(ObjectManager $manager)
   {

       $slugify = new Slugify();
       $title= "Titre de l'annonce n°";
       $slug=$slugify->slugify($title);

       for ($i=1; $i <30 ; $i++) {

        $ad= new Ad();

       $ad->setTitle("Titre de l'annonce n°$i")
       ->setSlug("$slug$i")
       ->setCoverImage("https://via.placeholder.com/350")
       ->setIntroduction("C'est une introduction")
       ->setContent("<p>Je suis le contenu</p>")
```

```
            ->setPrice(mt_rand(40,200))
            ->setRooms(mt_rand(1,5));

            for ($j=0; $j <mt_rand(2,5) ; $j++) {
                $image = new Image();
                $k=350+$j*50;
                $image ->setUrl("https://via.placeholder.com/$k")
                -> setCaption("Legende de $j")
                -> setAd($ad);

                $manager->persist($image);
            }

            $manager->persist($ad);

        $manager->persist($ad);
         $manager->flush();

    //mise a jour slug avec l'id créé pour rendre unique le slug
      $slug2 = $ad->getSlug().'_'.$ad->getId();
      $ad->setSlug($slug2);

     $manager->persist($ad);
      $manager->flush();


        }


     }
    }
```

**php bin/console doctrine:fixtures:load**

## XX) AFFICHAGE DE LA LISTE DES ANNONCES

### 1) RECUPERATION DES ANNONCES GRACE AU REPOSITORY:

- Si je veux faire **des sélections de données sur une table**, j'ai besoin d'un repository
- Si je veux faire **des manipulations**, j'ai besoin d'un manager

Rappel: Pour afficher , il nous faut un controlleur  (fonction publique, une route, une réponse)

**php bin/console make:controller AdController**

**AdController.php**

```
<?php

namespace App\Controller;

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

class AdController extends AbstractController
{
    /**
     * @Route("/ads", name="ads_index")
     */
    public function index()
    {
        return $this->render('ad/index.html.twig', [
            'controller_name' => 'AdController',
```

```
        ]);
    }
}
```

**ad/index.html.twig** (test statique)

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

{% block title %}Nos Annonces{% endblock %}

    {% block body %}

        <div class="container">
            <h1>Voici les annonces</h1>
            <div class="row">

        {% for i in 0..20  %}
                <div class="col-4 mt-3">

                    <div class="card text-center">
                        <div class="card-header">
                            2 chambres 50€/nuit<br/>
                            Pas encore noté
                        </div>
                        <div class="card-body">
                            <img src="https://via.placeholder.com/350" style="height:200px;width:100%;display:block">
                            <div class="text-left">
                                <h4>Bel Appartement</h4>
                                <p>
                                    Post quorum necem nihilo lenius ferociens Gallus ut leo cadaveribus pastus multa huius modi scrutabatur. quae singula narrare non refert, me professione modum, quod evitandum est, excedamus.
                                </p>
                                <a href="#" class="btn btn-primary ">Go somewhere</a>
                            </div>
                        </div>

                    </div>

                </div>
        {% endfor %}

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

Il va falloir maintenant faire appel au repository Ad pour sélectionner les données dans la table
On utilise $repo=$this->getDoctrine()->getRepository(Ad::class);

**AdController.php**

```
<?php

namespace App\Controller;

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

class AdController extends AbstractController
{
    /**
     * @Route("/ads", name="ads_index")
     */
    public function index()
    {
```

```
$repo=$this->getDoctrine()->getRepository(Ad::class);

//$ads=$repo->findOne(12);
 //$ads=$repo->findOneByTitle("Titre de l'article");
 //$ads=$repo->findByTitle("Titre de l'article");
//findOneByX trouve un élément
//findByX trouve plusieurs éléments dans un tableau

      $ads = $repo->findAll(); //recupère tous les enregistrements de la table visée

      return $this->render('ad/index.html.twig', [
         'ads' => $ads,
      ]);
   }
}
```

**ad/index.html.twig**

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

        {% block title %}Nos Annonces{% endblock %}

          {% block body %}

            <div class="container">
              <h1>Voici les annonces</h1>
              <div class="row">

            {% for ad in ads  %}
                 <div class="col-4 mt-3">

                    <div class="card text-center">
                      <div class="card-header">
                        {{ ad.rooms }} chambres {{ ad.price }}€/nuit<br/>
                        Pas encore noté
                      </div>
                      <div class="card-body">
                        <img src="{{ ad.coverImage }}" style="height:200px;width:100%;display:block">
                        <div class="text-left">
                          <h4>{{ ad.title }}</h4>
                          <p>
                            {{ ad.introduction }}
                          </p>
                          <a href="#" class="btn btn-primary ">Go somewhere</a>
                        </div>
                      </div>

                    </div>

                 </div>
            {% endfor %}

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

## 2) INJECTION DE DEPENDANCES:

Dans symfony , il y a un service container https://symfony.com/doc/current/service_container.html
Tout ce qu'il y a dans symfony est géré par Symfony. Il se charge d'instancier les classes et appel des fonctions..
Dans **AdController.php** , la fonction index a besoin d'un repository pour fonctionner, la fonction dépend de … On appelle cela une **dépendance**. Donc si j'ai une dépendance , je peux demander à Symfony de me la fournir plutôt que de la fabriquer.
Pour avoir les liste des services que l'on peut injecter:

**php bin\console debug:autowiring**

*Exemple pour SessionInterface:*
*public function index(AdRepository $repo, SessionInterface $session) {*
*....*

Exemple dans la fonction **index() ,** on veut afficher la liste des articles, elle a donc besoin d'un repository. On peut donc demander à Symfony de me la fournir plutôt que de la fabriquer soi-même.
Symfony est capable de voir s'il doit envoyer quelque-chose.
Nous on a besoin d'une instance d'une classe le repository des articles (AdRepository). Je précise donc dans les paramètres de la fonction index ( *on donne des indices à symfony*). index(AdRepository $repo) en n'oubliant pas de préciser le namespace.
Donc maintenant Symfony sait que la fonction a besoin d'une instance de AdRepository qui s'appellera $repo
Donc on n'a plus besoin de //$repo=$this->getDoctrine()->getRepository(Ad::class);

On peut simplifier grandement et on peut modifier **HomeController.php** :

```
class AdController extends AbstractController
{
    /**
     * @Route("/ads", name="ads_index")
     */
    public function index(AdRepository $repo)
    {

        $ads = $repo->findAll(); //recupère tous les enregistrements de la table visée

        return $this->render('ad/index.html.twig', [
            'ads' => $ads,
        ]);
    }
}
```

## 3) RECUPERATION D'UNE ANNONCE AVEC SON SLUG:

On créé une nouvelle fonction show() dans **AdController.php**

En plus de l'utilisation de **l'injection de dépendances**, on peut faire également appel au **param converter** qui associera au slug l'annonce correspondante.
De la même façon **show() =>** show(AdRepository $repo, $slug)
Donc on n'a plus besoin de

```
//$repo=$this->getDoctrine()->getRepository(Ad::class);
```

Mais on peut aller plus loin: show(AdRepository $repo, $slug) => show(Ad $ad)
Symfony sait l'article que l'on veut grâce au **Param Converter**
https://symfony.com/doc/master/bundles/SensioFrameworkExtraBundle/annotations/converters.html
Fait en sorte qu'un paramètre de votre route soit transformé en une entité. Le développeur peut demander à Symfony de lui injecter grâce au param converter le post dont l'identifiant est celui qui est dans la route.

Il associe la route {slug} à l'annonce.
Donc on n'a plus besoin de
//$ad = $repo -> findOneBy(['slug'=>$slug]);

d'où la fonction suivante:

```
use App\Entity\Ad;

/**
 * @Route("/ads/{slug}", name="ads_show")
 */
 public function show(Ad $ad){

// findOneByX avec X représentant n'importe quel champ de la table, trouve un seul élément
// findByX trouve plusieurs éléments => tableau

// on récupère l'annonce qui correspond au slug
//$ad = $repo -> findOneBy(['slug'=>$slug]);
   // $ad = $repo-> findOneBySlug($slug);

    return $this->render('ad/show.html.twig', [
        'ad' => $ad,
    ]);

 }

}
```

Bien sûr s'il n'y a pas d'annonce correspondant au slug, on va avoir une erreur 404.

On créé le fichier **show.html.twig**

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

{% block title %}
   {{ ad.title }}
{% endblock %}

{% block body %}

   <div class="container">
      <h1>{{ ad.title | striptags}}</h1>

      <div class="card card-body">
         <div class="row">
            <div class="col-8 mt-3">

               {{ ad.rooms }} chambres {{ ad.price }}€/nuit<br/>
               Pas encore noté


               <img src="{{ ad.coverImage }}" style="height:200px;display:block">
               <div class="text-left">
                  <h4>{{ ad.title | striptags }}</h4>
                  <p>
                     {{ ad.content | raw }}
                  </p>
                  <a href="#" class="btn btn-primary ">Réserver!</a>
               </div>

            </div>

            <div class="col-4">

               <div class="row">
                  <div class="col-6">
                     <img src="https://via.placeholder.com/64" >
                  </div>
                  <div class="col-6">
```

```
                Auteur: Eric
                <span class="badge badge-primary">3 annonces</span>
            </div>

        </div>

        <div class="row">
            Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
            tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
            quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
            consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
            cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
            proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
        </div>

    </div>

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

Dans la liste des annonces , il nous faut maintenant mettre les liens vers les annonces grâce aux slug de cette façon: <a href="{{ path('ads_show', {'slug': ad.slug}) }}" ....
Comme nous devons l'utiliser plusieurs fois, on va créer une variable:
{% set url = path('ads_show', {'slug': ad.slug}) %}

**index.html.twig**

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

{% block title %}Nos Annonces{% endblock %}

  {% block body %}

    <div class="container">
      <h1>Voici les annonces</h1>
      <div class="row">

        {% for ad in ads  %}

          {% set url = path('ads_show', {'slug': ad.slug }) %}

          <div class="col-4 mt-3">

            <div class="card text-center">
              <div class="card-header">
                {{ ad.rooms }} chambres {{ ad.price }}€/nuit<br/>
                Pas encore noté
              </div>
              <div class="card-body">
                <a href="{{ url }}">
                  <img src="https://via.placeholder.com/350" style="height:200px;width:100%;display:block">
                </a>
                <div class="text-left">
                  <a href="{{ url }}">
                  <h4>{{ ad.title | striptags }}</h4>
                  </a>
                  <p>
                    {{ ad.introduction }}
                  </p>
                  <a href="{{ url }}" class="btn btn-primary ">En savoir plus</a>
```

```
                    </div>
                </div>

            </div>

        </div>
        {% endfor %}

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

### 4) CARROUSEL DES IMAGES D'UNE ANNONCE:

On va dans bootstrap https://getbootstrap.com/docs/4.0/components/carousel/

On colle dans **show.html.twig** en dessous du contenu

```
<div id="carouselExampleIndicators" class="carousel slide col-7" data-ride="carousel">
                <ol class="carousel-indicators">

                    {% for image in ad.images %}
                        <li data-target="#carouselExampleIndicators" data-slide-to="{{ loop.index0 }}" {% if loop.first
%}class="active"{% endif %}></li>
                    {% endfor %}


                </ol>

                <div class="carousel-inner">

                    {% for image in ad.images %}
                        <div class="carousel-item {% if loop.first %} active {% endif %} ">
                            <img class="d-block w-100" src="{{ image.url }}" alt="First slide">

                            <div class="carousel-caption d-none d-md-block">
                                <h5>{{ ad.title | striptags }}</h5>
                                <p>{{ image.caption }}</p>
                            </div>

                        </div>
                    {% endfor %}
                </div>
                <a class="carousel-control-prev" href="#carouselExampleIndicators" role="button" data-slide="prev">
                    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
                    <span class="sr-only">Previous</span>
                </a>
                <a class="carousel-control-next" href="#carouselExampleIndicators" role="button" data-slide="next">
                    <span class="carousel-control-next-icon" aria-hidden="true"></span>
                    <span class="sr-only">Next</span>
                </a>
            </div><br/>
```

On en profite pour ajouter un lien Annonces dans le menu => base.html.twig

```
<li class="nav-item ">
                <a class="nav-link" href="{{ path('ads_index') }}">Liste des annonces</a>
            </li>
```

## XXI) CREATION D'UNE ANNONCE:

### 1) CREATION DE FORMULAIRE AUTOMATIQUE:

On va créer une nouvelle fonction **create()** dans **AdController.php** que l'on va placer au-dessus de la fonction **show()**
Car il risque d'y avoir confusion dans les routes à cause de la route /ads/{slug} ( le param converter ne va pas trouver l'entité car il va lancer show() )
On peut créer un formulaire de cette façon: $form = $this->createFormBuilder($ad)
Mais pour ne pas encombrer le controleur , on va utiliser la création de formulaire automatique et externaliser la création du formulaire ailleurs.

**php bin/console make:form**

Nom du form: **AnnonceType** (**par convention , on fini par Type**)
Est-ce que le formulaire se base sur une entité ? **Ad**

On va ensuite grâce à la fonction createForm créer un formulaire externe ( instancier la class )

```
/**
 * @Route("/ads/new", name="ads_create")
 */
 public function create(){

   $ad = new Ad();
   $form = $this -> createForm(AnnonceType::class, $ad);

    return $this->render('ad/new.html.twig',[
          'form' => $form -> createView()

    ]);

 }
```

Les développeurs de Symfony ont créé des moteurs de rendu twig. https://symfony.com/doc/4.4/form/form_themes.html
Donc on peut simplifier
Il y a une extension Twig pour les thèmes bootstrap https://symfony.com/doc/current/form/bootstrap4.html
Il faut aller dans la configuration de twig
**form_themes: ['bootstrap_4_layout.html.twig']**

On créé également le fichier **ad/new.html.twig**

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

{% form_theme form 'bootstrap_4_layout.html.twig' %}

{% block title %}Création d'une annonce{% endblock %}

   {% block body %}

      <div class="container">
         <h1>Créer une nouvelle annonce</h1>

{{ form_start(form) }}

{{ form_row(form.title, {'label':'Titre','attr':{'placeholder':"Titre de l'annonce"}}) }}
{{ form_row(form.introduction, {'label':'Introduction','attr':{'placeholder':"Description globale"}}) }}
{{ form_row(form.content, {'label':'Description détaillée','attr':{'placeholder':"Description détaillée"}}) }}
{{ form_row(form.coverImage, {'label':"URL de l'image principale",'attr':{'placeholder':"Adresse d'une image"}}) }}
{{ form_row(form.rooms, {'label':'Chaine URL','attr':{'placeholder':"Contenu de l'article"}}) }}
{{ form_row(form.price, {'label':'Prix par nuit','attr':{'placeholder':"Indiquez le prix par nuit"}}) }}

<button type="submit" class="btn btn-success">Créer la nouvelle annonce</button>
```

```
{{ form_end(form) }}


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

*(Voir https://symfony.com/doc/current/form/form_customization.html )*
*form_row est équivalent à form_label() + form_error() + form_widget() + form_help()*

On modifie le contenu de **AnnonceType.php** pour imposer certains champs
*(liste des types possibles: https://symfony.com/doc/current/reference/forms/types.html )*

```
….
->add('title')
//->add('slug',TextType::class,['required' => false]) // slug non obligatoire et on ne met pas le form_row(form.slug) dans le twig
        ->add('price',MoneyType::class)
        ->add('introduction')
        ->add('content')
        ->add('coverImage',UrlType::class)
        ->add('rooms',IntegerType::class)
…..
```

2) **RECUPERER ET SAUVEGARDER LES DONNEES DU FORMULAIRE:**
Par défaut le Form n'a pas d'action et une méthode post
On demande à Symfony de nous donner la requête ( injection de dépendance).
Les informations envoyées se trouve dans la "**request**"

Dans la fonction create() , on peut mettre:

```
create(Request $request,  EntityManagerInterface  $manager){

…..

// essaye d'analyser la requête est-ce que cela a été soumis ou pas, est ce que tout va bien ?
// le formulaire a la possibilité de parcourir le requête et d'extraire les informations du form
// Il va retrouver toutes les données et les relier à l'entité Ad
// dump($request);
// ex: $request - >request->get('title');

   $form ->handleRequest($request);
   //dump($ad); // équivalent de var_dump


      // si le form est soumis et valide (par rapport aux règles en place)
      if ($form->isSubmitted() && $form->isValid()){

                  // génération du slug automatiquement si pas soumis (voir fixture)
               if (!$ad->getSlug()) {
                  $slugify = new Slugify();
                  $slug=$slugify->slugify($ad->getTitle());
                  $ad->setSlug($slug);
                  }


               $manager->persist($ad); // previent doctrine que l'on veut sauver
               $manager->flush(); // envoi la requête à la base de donnée

             //mise a jour slug avec l'id créé pour rendre unique le slug
            $slug2 = $ad->getSlug().'_'.$ad->getId();
           $ad->setSlug($slug2);
           $manager->persist($ad);
           $manager->flush();
```

```
            // on retourne sur la page de l'article
        return $this->redirectToRoute('ads_show',['slug'=>$slug]);

    }
```

…

### 3) MESSAGE FLASH:

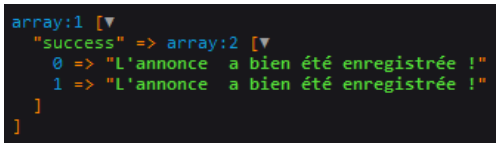
On veut alerter sur les actions effectuées ( notifications qui n'apparaissent qu'une seule fois)
On a une fonction **addFlash** **(**créé un message flash via le controleur)

```
$this->addFlash(
            'success',
            "L'annonce {$ad->getTitle()} a bien été enregistrée !"
        );
```

Rem: Lorsque l'on met des accolades `{$ad->getTitle()}` , on dit à PHP d'évaluer ce qu'il y a à l'intérieur avant de faire le reste.
Sinon , on aurait pu mettre: `'L\'annonce '.$ad->getTitle().' a bien été enregistrée !'`

Maintenant , il faut pouvoir afficher le message.
On va dans le template **header.html.twig** . On peut utiliser la grosse variable d'environnement **app** qui contient énormément de données https://symfony.com/doc/current/templating/app_variable.html
Les données sont enregistrées sous forme d'un tableau exemple:

**{{ dump(app.flashes) }}**




```
{#
    {{ dump(app) }}
    {{ dump(app.flashes) }}
    #}

    {% for label, messages in app.flashes %}

        <div class="alert alert-{{ label }} alert-dismissible fade show" role="alert">

            {% for message in messages %}
                <p>{{ message  }}</p>
            {% endfor %}

            <button type="button" class="close" data-dismiss="alert" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>

    {% endfor %}
```

**Voir (https://getbootstrap.com/docs/4.3/components/alerts/ )**

## 4) AJOUT ET SUPPRESSION D'IMAGES DANS LE FORMULAIRE:

### a) AJOUT D'IMAGE:

Rappel: une annonce peut être relié à plusieurs images (OneToMany)
On va dans **AnnonceType.php** qui représente le formulaire d'une annonce et on va y rajouter un champ de type **collection** ( permet d'ajouter des formulaires à l'intérieur d'autres)
https://symfony.com/doc/current/reference/forms/types/collection.html

On va donc créer un autre formulaire ( qui permettra d'ajouter des images)
**php bin/console make:form**

Nom du form: **ImageType** (**par convention , on fini par Type**)
Est-ce que le formulaire se base sur une entité ? **Image**

On voit dans ImageType qu'il faut l'url, la caption et l'annonce de l'image ( on supprime ->add('ad') car on sera déjà au sein d'une annonce)

Dans AnnonceType.php , on rajoute un champ:

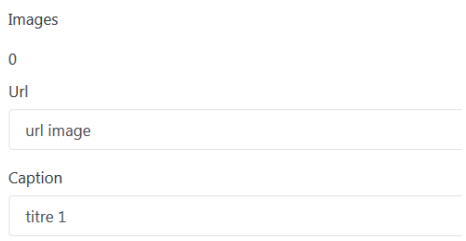
```
->add('images',CollectionType::class,[
                'entry_type' => ImageType::class,
  'allow_add' => true, // on peut rajouter des éléments et création du prototype pour l'ajout dans le template twig
    'allow_delete' => true,

        ])
```

Maintenant, côté affichage du formulaire **aucun champ ne sera visible à moins que le tableau contienne déjà des images. (CollectionType fonctionne de cette manière)**
On va donc pour faire un test, ajouter manuellement des images au niveau de la fonction **create()** dans **AdController.php**

```
// on simule le fait que l'annonce a déjà deux images
   $image = new Image();
   $image ->setUrl('url image');
   $image ->setCaption('titre 1');
   $ad->addImage($image);

   $image2 = new Image();
   $image2 ->setUrl('url image 2');
   $image2 ->setCaption('titre 2');
   $ad->addImage($image2);
```

On rajoute dans le bloc formulaire

```
{# champ annonce_images qui sera sur-chargé #}
        {{ form_row(form.images) }}
<button type="button" id="add_image" class="btn btn-success">Ajouter une image</button><br/><br/>
```




**Personnaliser l'affichage des sous-formulaires:**
On va créer un bloc qui sera appelé par twig
On voit grâce à l'inspecteur que le champ est de la forme **annonce_images**

On peut personnaliser la façon dont les différents bloc du formulaire sont appelés
On va créer un comportement différent pour form_row lorsque l'on aura le champ **annonce_images** , pour cela on créé un bloc. https://symfony.com/doc/4.4/form/form_customization.html => https://symfony.com/doc/4.4/form/form_themes.html

```
{# permet de modifier la sortie de form_row sur le champ annonce_images   #}
{# entry représente un élément de la collection ( ici sous-formulaire)  #}
{# Ici la variable form représente une entry ( le champ annonce_images)  #}
{% block _annonce_images_entry_row %}
   {#{{ dump() }}#}
   <div class="row" id="{{ id }}">
      <div class="col">
         {{ form_row(form.url, {'label':false,'attr':{'placeholder':"url de l'image"}}) }}
      </div>
      <div class="col">
         {{ form_row(form.caption, {'label':false,'attr':{'placeholder':"description de l'image"}}) }}
      </div>
      <div class="col-2">
         <button type="button"  class="btn btn-danger del_image" data-bloc="{{ id }}" >X</button>
      </div>
   </div>

{% endblock %}
```

que l'on appellera grâce à **_self** dans:  {% form_theme form with ['bootstrap_4_layout.html.twig', _self] %}
https://symfony.com/doc/current/form/form_themes.html

On remarque que l'ajout de 'allow_add' => true et du bloc _annonce_images_entry_row a créé un prototype que l'on va utiliser pour rajouter les champs (voir inspecteur de code: data-prototype, cela correspond à un sous-formulaire vierge )
Je veux lorsque l'on clique sur le bouton que les champs se rajoutent , on va faire cela en jQuery

```
{% block javascripts %}
    <script>

    $('#add_image').click(function(){

        // détermination du nombre de ligne afin de connaitre l'indice à mettre dans le prototype
        const index = $('#annonce_images .row').length;
         //console.log(index);

         // je récupère le prototype des entrées
         var tmpl = $('#annonce_images').data('prototype');
        console.log(tmpl);

       // on remplace __name__ plusieurs fois
         tmpl = tmpl.replace(/__name__/g, index);


         //On ajoute tmpl à la fin de la div annonce_images
         $('#annonce_images').append(tmpl);
        });

    </script>

{% endblock %}
```

On peut maintenant supprimer les images d'exemples du controller

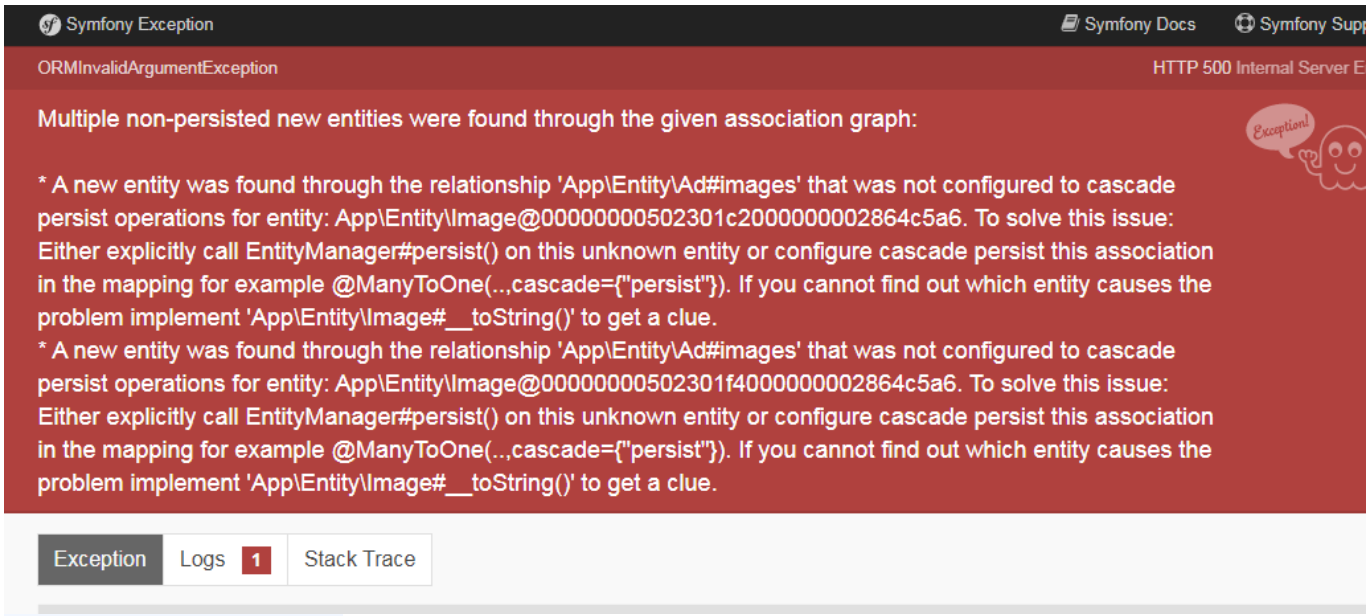
**b) SUPPRESSION D'IMAGE:**

On veut maintenant avoir la possibilité de supprimer les champs
On créé une fonction **deleteBloc()** que l'on va lancer lors du clic sur l'ajout du formulaire

```
{% block javascripts %}
    <script>

    //variable globale
    var counter = 0;

    $('#add_image').click(function(){

        // nbrs de groupe de champs créés afin de connaitre l'indice à mettre dans le prototype
        counter = counter + 1;
        console.log(counter);

        // détermination du nombre de ligne
        //const index = $('#annonce_images .row').length;
         //console.log(index);

         // je récupère le prototype des entrées
         var tmpl = $('#annonce_images').data('prototype');
        //console.log(tmpl);

        // on remplace __name__ plusieurs fois
        tmpl = tmpl.replace(/__name__/g, counter);

         //On ajoute tmpl à la fin de la div annonce_images
         $('#annonce_images').append(tmpl);

        // on gère le bouton supprimer
        deleteBloc();

        });


      function deleteBloc(){

        $('.del_image').click(function(){
        var bloc = $(this).data('bloc');
        //console.log(bloc);

        $('#'+ bloc).remove();


    });

      }

      // on gère le bouton supprimer lors de l'arriver sur le formulaire (s'il y a déjà des images)
        deleteBloc();

      </script>

{% endblock %}
```

## 5) SAUVEGARDE DES ANNONCES AVEC LES IMAGES:

On a maintenant des entités image reliées à une annonce, il faut donc les faire persister sinon on aura une erreur.




```
//dump($ad->getImages());exit;
                foreach ($ad->getImages() as  $image) {
                    $image->setAd($ad);
                    $manager->persist($image);
                }
```

## 6) VALIDATION DES FORMULAIRES:

https://symfony.com/doc/4.4/validation.html

Les validations que l'on a actuellement sont côtés frontend en html5 donc facilement contournable ( tout ce qui se passe sur le navigateur est corruptible) .

On va donc faire une validation sur l'entité elle-même ( les contraintes)

On va travailler sur l'entité **Ad**

On rajoute le namespace: use Symfony\Component\Validator\Constraints as Assert;

Puis on ajoute les contraintes par exemple:

```
    /**
     * @ORM\Column(type="string", length=255)
    * @Assert\Length(
     *      min = 10,
     *      max = 50,
     *      minMessage = "Le titre doit faire plus que {{ limit }} caractères",
     *      maxMessage = "Le titre ne peut pas faire plus que {{ limit }} caractères"
     * )
     */
    private $title;


    /**
     * @ORM\Column(type="string", length=255)
     * @Assert\Url()
    * @Assert\Regex(
     *      pattern="#\.(jpg|png|gif)$#",
     *      match=true,
     *      message="Url doit se terminer par .jpg ou .png ou .gif"
     * )
     */
    private $coverImage;
```

Validation de l'unicité, éviter les doublons:
On ajout le namespace:
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

et au-dessus de la class , on rajoute par exemple:

```
* @UniqueEntity(
*          fields={"title"},
*          errorPath="title",
*     message="Ce titre existe déjà",
*
* )
 */
class Ad
{
……..
```

Pour l'instant symfony valide le formulaire mais pas le sous-formulaire images.
Il faut utiliser une contrainte particulière **Valid** qui permet de s'assurer qu'une collection de sous-formulaires soit valide
https://symfony.com/doc/current/reference/constraints/Valid.html
dans Ad.php on ajoute:

```
    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Image", mappedBy="ad", orphanRemoval=true)
     *@Assert\Valid()
     */
    private $images;
….
```

On force symfony à valider aussi les images suivant les éventuelles contraintes qu'il y auraient

Rem : En rajoutant ces contraintes , cela va modifier également les pattern des champs concernés côté frontend.
On peut donc modifier également l'attribut title des champs pour que cela s'affiche.
Par exemple : dans ad/new.html.twig, on peut mettre :

```
        {{ form_row(form.title,{'label':"Titre de l'annonce",'attr':{'placeholder':"Titre de l'annonce",'title':'Titre avec un minimun de 10 caractères et au maximum 50 caractères'}}) }}
```

### 7) FORMULAIRE D'EDITION DES ANNONCES:

On va créer une nouvelle fonction **edit()** dans **AdController.php** que l'on copie de la fonction **create()**

```
/**
* permet d'afficher le formulaire d'édition
 * @Route("/ads/{slug}/edit", name="ads_edit")
 */
 public function edit(Request $request,EntityManagerInterface $manager,Ad $ad){

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

// debut Résolution du problème du bug du champ slug éventuellement vide lors du handleRequest (il extrait les données POST de la requête précédente, les traite et exécute toute validation (vérifie l'intégrité des données attendues par rapport aux données reçues))
//dump($request); // contenu de la request
//dump($request->request->get('annonce'));
//dump($request->request->get('annonce')['slug']);

//https://symfony.com/doc/current/components/http_foundation.html
     // Si le slug est vide , on le remplace par le titre formaté
     //      $slugify = new Slugify();
```

```
//     $data = $request->request->get('annonce');
        //dump($data);
// if (empty($data['slug'])) $data['slug']=$slugify->slugify($request->request->get('annonce')['title']);
       //dump($data);
        //dump($data['slug']);
//     $request->request->set('annonce',$data);
        //dump($data);



    $form ->handleRequest($request);


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


            //dump($ad->getImages());exit;
            foreach ($ad->getImages() as  $image) {
               $image->setAd($ad);
               $manager->persist($image);
            }


            $manager->persist($ad); // previent doctrine que l'on veut sauver
            $manager->flush(); // envoi la requête à la base de donnée

        $this->addFlash(
            'success',
            'L\'annonce '.$ad->getTitle().' a bien été modifiée !'
        );

            // on retourne sur la page de l'annonce
       return $this->redirectToRoute('ads_show',['slug' => $ad->getSlug()]);


    }

  return $this->render('ad/edit.html.twig',[
        'form' => $form -> createView(),
        'ad' => $ad,

            ]);

}
```

On créé **edit.html.twig** avec le contenu de **new.html.twig** que l'on modifie
ATTENTION petit bug (Si on supprime une image et que l'on rajoute) pour le jQuery à corriger par rapport à **new.html.twig**
On remplace var counter = 0; par

```
//variable globale
        var counter = {{ form.images | length }};
```

On voit que l'on a presque le même code dans les deux fichiers, on va donc les factoriser
Mettre le javascript dans **/js/ad.js** et ensuite y faire appel dans **edit.html.twig** et **new.html.twig**

```
    {% block javascripts %}
        <script type="text/javascript">
```

```
        //variable globale qui donne le nombre de form images
            var counter = {{ form.images|length }};
        </script>

    <script src="/js/ad.js"></script>

    {% endblock %}
```

On a aussi ce qui concerne le template form de la collection d'images que l'on va mettre dans un fichier à part **_collection.html.twig** et ensuite on modifie dans **edit.html.twig** et **new.html.twig**
{% form_theme form with ['bootstrap_4_layout.html.twig', "ad/_collection.html.twig"] %} pour y faire appel

## XXII) CREATION UTILISATEUR:

### 1) CREATION ENTITE USER:

**php bin/console make:entity User**

firstName string 255 no
lastName string 255 no
email string 255 no
picture string 255 yes
hash string 255 no
introduction string 255 no
description text no
slug string 255 no

**php bin/console make:migration**

**php bin/console doctrine:migrations:migrate**

### 2) MANYTOONE ENTRE LES ANNONCES ET LES UTILISATEURS:

Il faut pour les annonces que je sache quel est son auteur. Il faut donc modifier l'entité annonce

**php bin/console make:entity Ad**

author
relation
User
ManyToOne
no
yes
ads
no

**php bin/console make:migration**

**ATTENTION si on lance maintenant la migration , il va y avoir un problème car on rajoute une colonne author_id au niveau de la table Ad qui contient des données fictives ( or il n'y a pas d'auteur pour l'instant)**
**Il faut donc supprimer toutes les données**

**php bin/console doctrine:migrations:migrate**

### 3) AJOUTER DE FAUX UTILISATEURS ( FIXTURE):

On modifie AppFixtures.php

```
class AppFixtures extends Fixture
{
   public function load(ObjectManager $manager)
   {

       $slugify = new Slugify();

      // utilisateurs
for ($l=1; $l <5 ; $l++) {
           $user= new User();
           $slug=$slugify->slugify("prenom$l-nom$l");
           $user->setFirstName("prenom$l")
                -> setLastName("nom$l")
                ->setEmail("test$l@test.fr")
                ->setPicture("https://via.placeholder.com/64")
                ->setIntroduction("introduction$l")
                ->setDescription("description$l")
                ->setSlug($slug)
```

```
                ->setHash("pass");

        $manager->persist($user);


       // Annonces
      // $slugify = new Slugify();
       $title= "Titre de l'annonce n°";
       $slug=$slugify->slugify($title);

      for ($i=1; $i <10 ; $i++) {


       $ad= new Ad();

       $ad->setTitle("Titre de l'annonce n°$i-$l")
       ->setSlug("$slug$i-$l")
       ->setCoverImage("https://via.placeholder.com/350")
       ->setIntroduction("C'est une introduction")
       ->setContent("<p>Je suis le contenu</p>")
       ->setPrice(mt_rand(40,200))
       ->setRooms(mt_rand(1,5))
       ->setAuthor($user);

       for ($j=0; $j <mt_rand(2,5) ; $j++) {
           $image = new Image();
           $k=350+$j*50;
           $image ->setUrl("https://via.placeholder.com/$k")
           -> setCaption("Legende de $j")
           -> setAd($ad);

          $manager->persist($image);
       }

       $manager->persist($ad);
       $manager->flush();

       //mise a jour slug avec l'id créé pour rendre unique le slug
          $slug2 = $ad->getSlug().'_'.$ad->getId();
          $ad->setSlug($slug2);

         $manager->persist($ad);
          $manager->flush();


   }



}


   $manager->flush();
 }
}
```

**php bin/console doctrine:fixtures:load**

### 4) MODIFICATION DE LA PAGE ANNONCE:

On veut faire apparaitre l'auteur d'une annonce. Dans show.html.twig :

```
<div class="col-4">

            <div class="row">
               <div class="col-3">
                  <img src="{{ ad.author.picture }}" >
               </div>
               <div class="col-9">
                  Auteur: {{ ad.author.firstName }} {{ ad.author.lastName }}
                  <span class="badge badge-primary">{{ ad.author.ads | length }} annonces</span>
               </div>

            </div>

            <div class="row">
              {{ ad.author.description | raw }}
            </div>

         </div>
```

### 5) ENCODAGE DES MOTS DE PASSE:

On utilise le composant **security** (ensemble de fonctionnalités qui gèrent la sécurité dans Symfony)

La configuration se situe dans **config > package > security.yaml**

Il faut mettre en place les **encodeurs**. Ils permettent de hasher les mots de passe des utilisateurs

```
security:
   encoders:
      App\Entity\User:
         algorithm: auto
```

On peut encoder plusieurs entity ( plusieurs encodeurs)

On va utiliser également la class UserPasswordEncoderInterface
https://symfony.com/doc/4.4/security.html#c-encoding-passwords

Dans la fixture, il va falloir créer un constructeur pour passer UserPasswordEncoderInterface et injecter l'encodeur. (injection de dépendance)

```
            private $encoder;
               public function __construct(UserPasswordEncoderInterface $encoder){

                  $this->encoder=$encoder;

               }
```

Et on aura:

```
$encoded = $this->encoder->encodePassword($user, "pass");
            $user->setHash($encoded);
```

Si maintenant , on lance la fixture, il va y avoir une erreur

```
Argument 1 passed to Symfony\Component\Security\Core\Encoder\UserPasswordEncoder::encodePasswor
d() must be an instance of Symfony\Component\Security\Core\User\UserInterface, instance of App\
Entity\User given, called in C:\laragon\www\formation-symfony4-encours\src\DataFixtures\AppFixt
ures.php on line 35
```

, on a besoin également que le **User** que l'on passe soit:

- **une entité**
- **implémenté d'une interface. (obligé d'avoir certaines méthodes essentielles: plan ) ( clic droit sur UserInterface ->Goto definition)**

https://api.symfony.com/3.4/Symfony/Component/Security/Core/User/UserInterface.html

Maintenant il faut implémenter toutes les fonctions obligatoires de l'interface ( but de l'interface )

## Methods

(Role|string)[]
getRoles()
Returns the roles granted to the user.

string
getPassword()
Returns the password used to authenticate the user.

string|null
getSalt()
Returns the salt that was originally used to encode the password.

string
getUsername()
Returns the username used to authenticate the user.

eraseCredentials()
Removes sensitive data from the user.

Il va falloir rajouter les fonctions manquantes: getPassword(), getUsername(), getSalt() , getRoles(), eraseCredentials
https://api.symfony.com/3.4/Symfony/Component/Security/Core/User/UserInterface.html
On retourne dans l'entité **User** et on rajoute:

```
….
use Symfony\Component\Security\Core\User\UserInterface;
…
class User implements UserInterface
{….
```

Et  on créé les fonctions qui manquent:

```
public function getPassword() {
    return $this->hash;
    }

// Identifiant avec lequel on va se connecter
    public function getUsername() {
    return $this->email;
    }

// Retourne une chaine de caractères à concaténer au mot de passe, si on a besoin.
    public function getSalt() {

    }

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

    }

// Supprimer les informations sensibles de l'entité.
   public function eraseCredentials() {

    }
```

On relance la fixture et on voit que maintenant le password est encodé au niveau de la base de donnée.

## XXIII) FORMULAIRE DE CONNEXION:

### 1) CREATION D'UNE AUTHENTIFICATION:

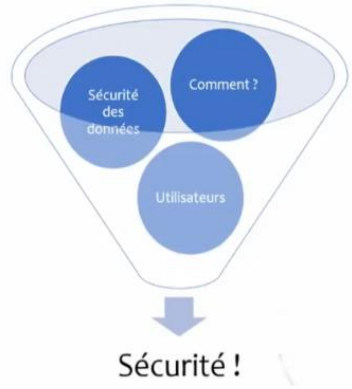
Comment faire en sorte que les gens puissent s'incrire et se connecter pour voir des choses et des actions qui ne sont pas autorisées à tout le monde.
Pour cela, on utilise le composant **security** (ensemble de fonctionnalités qui gèrent la sécurité dans Symfony)

#### a) Comment protéger une application ?:

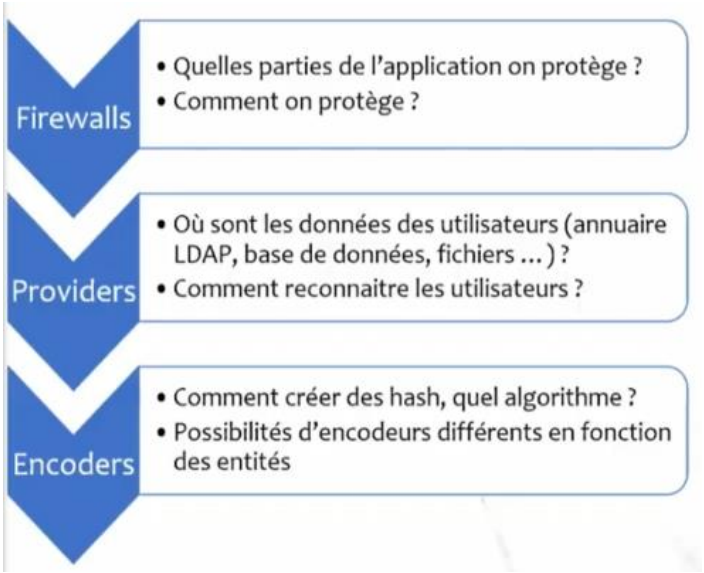
Quelles questions doit-on se poser ?

- Comment ? (un formulaire de login, token … )
- Comment gérer la sécurité des données ? (mot de passe..)
- On se trouve les utilisateurs ? (dans la base de données, viennent des annuaires, dans un fichier texte.. )

On va retrouver ces questions dans le fichier de configuration du composant (package) security




**LE COMPOSANT SECURITY DE SYMFONY**




#### b) Configuration de security:

La configuration se situe dans **config > package > security.yaml**
On retrouve un firewall pour le développement qui match avec des adresses, notamment pour laisser accès à la toolbar
Et pas de sécurité dessus:

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
```

On peut créer autant de firewall que l'on veut

Pour l'instant **main** match sur tout le reste de l'application et on a le droit d'y arriver en tant que anonyme

## 2) CREATION DU CONTROLLER ET DU FORMULAIRE:

Il faut expliquer à symfony. Ou sont les utilisateurs (providers) ?
On va utiliser les providers de données ( permet de dire comment fournir les données utilisateurs )
https://symfony.com/doc/current/security/user_provider.html
https://symfony.com/doc/3.3/security/entity_provider.html

On va dans la configuration: **config > package > security.yaml**

```
providers:
        in_memory: { memory: ~ }
        in_database: # nom arbitraire
            entity:
                class: App\Entity\User # entité qui sert à voir ou sont les utilisateurs
                property: email # propriété de l'utilisateur qui permet de l'authentifier
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: lazy #Le mode LAZY est celui par défaut. Les données sont chargées uniquement si nécessaires
```

On créé le controleur:
**php bin/console make:controller AccountController**

```
class AccountController extends AbstractController
{
    /**
     * @Route("/login", name="account_login")
     */
    public function login()
    {
        return $this->render('account/login.html.twig');
    }
}
```

On n'a pas besoin de créer un formulaire via la console
On va faire un formulaire manuellement
Lors d'une connexion Symfony s'attend à recevoir des champs de noms **_username** et **_password**

On va spécifier que la fonction login est "spéciale", elle sert à faire le login des utilisateurs
https://symfony.com/doc/4.4/security/form_login.html

Action du formulaire ( ou sont envoyés les données ?)
On envoi à la fonction login() sauf que Symfony va intercepter les informations ( on ne les traite pas nous même )

On configure donc le composant de sécurité ( firewalls) pour qu'il tienne compte du formulaire
**config > package > security.yaml**

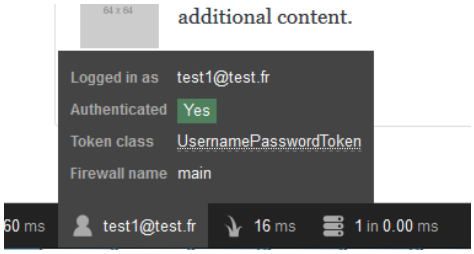
```
        main:
            anonymous: lazy

            # on se sert du provider in_database
            provider: in_database

            # on précise que l'on se connecte via un formulaire de login
            form_login:
                #endroit ou se situe le formulaire (nom d'une route)
```

```
        login_path: account_login

        #l'endroit a appeler pour vérifier les infos de login
        check_path: account_login
```

**login.html.twig**

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

{% block body %}


<h1>Connexion sur le site</h1>

<div class="container">
<form action="{{ path('account_login') }}" method="post">

    <div class="form-group">
        <input placeholder="Adresse email" required type="text" name="_username" class="form-control">
    </div>

    <div class="form-group">
        <input placeholder="mot de passe" required type="password" name="_password" class="form-control">
    </div>

    <div class="form-group">
        <button type="submit" class="btn btn-success">Connexion !</button>
    </div>

</form>
 </div>

{% endblock %}
```

Et on teste:



### 3) DECONNEXION DE L'UTILISATEUR:

De la même façon, Symfony va gérer la déconnexion via le composant de sécurité
https://symfony.com/doc/4.4/security.html#logging-out

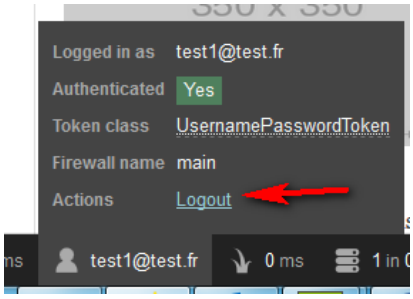
On créé une fonction vide dans **AccountController.php** :

```
/**
 * @Route("/deconnexion", name="account_logout ")
 */
public function logout(){

}
```

Le composant de sécurité va se charger de la déconnexion , il faut juste le préciser dans
**config > package > security.yaml**

```
        logout:
            path: account_logout # nom de la route qui permet de se déconnecter ?

            # nom de la route où l'on doit aller après s'être déconnecté ?
            target: account_login
```

Puis on réactualise la page /logout
Ou en faisant




### 4) REPERER LES ERREURS D'AUTHENTIFICATION:

Symfony fournit des utilitaires qui concerne l'authentification **( objet AuthenticationUtils )**
https://symfony.com/doc/current/security/form_login_setup.html
On modifie la fonction login()

```
/**
 * @Route("/login", name="account_login")
 */
public function login(AuthenticationUtils $utils)
{

        // fonction qui permet de récupérer la dernière erreur
    $error = $utils-> getLastAuthenticationError();

     //dump($error);

    return $this->render('account/login.html.twig',[
        // on renvoit un boolean
        'hasError' => $error

    ]);
}
```

Et dans **login.html.twig**

```
{% if (hasError) %}
    <div class="alert alert-danger alert-dismissible fade show" role="alert">
        Mauvais identifiant ou mot de passe
        <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
```

```
{% endif %}
```

## XXIV) FORMULAIRE D'INSCRIPTION:

### 1) CREATION DU FORMULAIRE:

On créé une class formulaire

**php bin/console make:form RegistrationType**

User

On modifie dans le fichier **src/Form/RegistrationType,**

```
->add('firstName')
        ->add('lastName')
        ->add('email')
        ->add('picture')
        ->add('hash')
        ->add('introduction')
        ->add('description')
        ->add('slug')
```

En

```
        ->add('firstName')
        ->add('lastName')
        ->add('email',EmailType::class)
        ->add('picture',UrlType::class)
        ->add('hash',PasswordType::class)
        ->add('confirm_password',PasswordType::class)
        ->add('introduction')
        ->add('description')
```

Par contre faites attention, Si on rajoute un nouveau champ, **il faut aussi qu'il existe** dans l'entité **User**

```
public $confirm_password;
```

On créé une nouvelle fonction register() dans **AccountController.php**

```
/**
    * @Route("/register", name="account_register")
    */
   public function register(){

   $user=new User();
   $form = $this->createForm(RegistrationType::class, $user ); // on relie les champs du formulaire aux champs de l'utilisateur


 return $this->render('account/registration.html.twig', [
         'formUser' => $form -> createView(), // envoi du form à twig
       ]);

   }
```

et un fichier **registration.html.twig**

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

{% form_theme formUser 'bootstrap_4_layout.html.twig' %}

{% block body %}

   <div class="container">
      <h1>Inscription sur le site</h1>
```

```
        {{ form_start(formUser) }}

        {{ form_row(formUser.firstName, {'label':'Prénom','attr':{'placeholder':"Votre Prénom"}}) }}
        {{ form_row(formUser.lastName, {'label':'Nom','attr':{'placeholder':"Votre Nom"}}) }}
        {{ form_row(formUser.email, {'label':'Email','attr':{'placeholder':"Votre email"}}) }}
        {{ form_row(formUser.picture, {'label':'Image de profil','attr':{'placeholder':"Votre Image de profil"}}) }}

        {{ form_row(formUser.hash, {'label':'Mot de passe','attr':{'placeholder':"Votre mot de passe"}}) }}
        {{ form_row(formUser.confirm_password, {'label':'Confirmation Mot de passe','attr':{'placeholder':"Confirmation du mot
de passe"}}) }}

        {{ form_row(formUser.introduction, {'label':'Introduction','attr':{'placeholder':"Votre introduction"}}) }}
        {{ form_row(formUser.description, {'label':'Description','attr':{'placeholder':"Votre description"}}) }}


        <button type="submit" class="btn btn-success">Inscription !
        </button>

        {{ form_end(formUser) }}
    </div>

{% endblock %}
```

**2) <u>ENREGISTRER LES DONNEES DU FORMULAIRE:</u>**

Dans la fonction **register()**

```
/**
     * @Route("/register", name="account_register")
     */
    public function register(Request $request, EntityManagerInterface $manager,UserPasswordEncoderInterface $encoder){

    $user=new User();
    $form = $this->createForm(RegistrationType::class, $user ); // on relie les champs du formulaire aux champs de l'utilisateur

        $form ->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid())
                {
                    $password=$user->getHash();
                    $encoded = $encoder->encodePassword($user, $password);
                    $user->setHash($encoded);

                $slugify = new Slugify();
                $slug=$slugify->slugify("{$user->getFirstName()}-{$user->getLastName()}");
                $user->setSlug($slug);

                    $manager->persist($user); // on fait persister dans le temps
                    $manager->flush(); // envoi la requête => ecriture dans la base

                    $this->addFlash(
                'success',
                'Vous avez bien été enregistrée !'
                    );

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

                }


    return $this->render('account/registration.html.twig', [
            'formUser' => $form -> createView(), // envoi du form à twig
        ]);
```

```
}
```

### 3) CREATION DES VALIDATIONS:

Il faut maintenant rajouter des validations https://symfony.com/doc/current/validation.html :

#### a) hash et confirm_password doivent être identiques:

On retourne dans l'entité **User** et on rajoute le namespace et les contraintes:
use Symfony\Component\Validator\Constraints as Assert;

```
….
   /**
    * @ORM\Column(type="string", length=255)
    * @Assert\Length(
    *      min = 8,
    *      minMessage="Votre mot de passe doit faire un minimun de 8 caractères",
    *)
    */
   private $hash;


   /**
   * @Assert\EqualTo(propertyPath="hash",message="Les deux mots de passe ne sont pas identiques")
    */
public $confirm_password;
```

#### b) L'email doit être unique:

https://symfony.com/doc/current/reference/constraints/UniqueEntity.html
On rajoute les contraintes dans l'entité **User**

```
…
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
  * @UniqueEntity(fields={"email"},
   *     message="cet email est déjà utilisé",
   *)
 */

class User implements UserInterface
{
….
```

#### c) Validation des différents champs:

```
   /**
    * @ORM\Column(type="string", length=255)
    *@Assert\NotBlank(message="Vous devez renseigner votre prénom")
    */
   private $firstName;

   /**
    * @ORM\Column(type="string", length=255)
     *@Assert\NotBlank(message="Vous devez renseigner votre nom")
    */
   private $lastName;

   /**
    * @ORM\Column(type="string", length=255)
     *@Assert\Email(message="veuillez renseigner un email valide !")
    */
   private $email;

   /**
```

```
    * @ORM\Column(type="string", length=255, nullable=true)
   *@Assert\Url(message="veuillez renseigner un url valide !")
    */
   private $picture;
```

### 4) MISE A JOUR DE LA BARRE DE NAVIGATION:

On peut utiliser la grosse variable d'environnement **app** qui contient énormément de données
https://symfony.com/doc/current/templating/app_variable.html

On met **header.html.twig** à jour

```
<ul class="navbar-nav ml-auto">
            {% if (app.user) %}
               <li class="nav-item active">
                  <a class="nav-link" href="{{ path('account_logout') }}">Déconnexion</a>
               </li>
               <li class="nav-item active">
                  <a class="nav-link" href="#">Mon compte</a>
               </li>
               {% else %}
               <li class="nav-item active">
                  <a class="nav-link" href="{{ path('account_register') }}">Inscription</a>
               </li>
               <li class="nav-item active">
                  <a class="nav-link" href="{{ path('account_login') }}">Connexion</a>
               </li>

               {% endif %}

            </ul>
```

## XXV) FORMULAIRE D'EDITION DU PROFIL:

On créé un nouveau formulaire:
**php bin/console make:form AccountType**

User

On supprime dans AccountType.php:

```
            ->add('hash')
            ->add('slug')
```

On créé une nouvelle fonction **profile()** dans **AccountController.php**

```
    /**
     * affiche et traite le formulaire de modification de profil
      * @Route("/account/profile", name="account_profile")
      */
    public function profile(Request $request, EntityManagerInterface $manager)
    {

// recupérer l'utilisateur qui est connecté
       $user= $this->getUser();

    $form = $this->createForm(AccountType::class, $user ); // on relie les champs du formulaire aux champs de l'utilisateur

    $form ->handleRequest($request);


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


                        $manager->persist($user); // on fait persister dans le temps
                        $manager->flush(); // envoi la requête => ecriture dans la base

                        $this->addFlash(
                          'success',
                          'le profil a bien été modifié !'
                        );

                    //return $this->redirectToRoute('account_login');

                    }



                        return $this->render('account/profile.html.twig' ,[
            'form' => $form -> createView(), // envoi du form à twig

        ]);
                    }
```

Et **profile.html.twig**

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

{% form_theme form 'bootstrap_4_layout.html.twig' %}


    {% block title %}Modification du profil utilisateur{% endblock %}

{% block body %}
```

```
<div class="container">
    <h1>Modification du profil</h1>

        {{ form_start(form) }}

    {{ form_row(form.firstName, {'label':'Prénom','attr':{'placeholder':"Votre Prénom"}}) }}
    {{ form_row(form.lastName, {'label':'Nom','attr':{'placeholder':"Votre Nom"}}) }}
    {{ form_row(form.email, {'label':'Email','attr':{'placeholder':"Votre email"}}) }}
    {{ form_row(form.picture, {'label':'Image de profil','attr':{'placeholder':"Votre Image de profil"}}) }}

    {{ form_row(form.introduction, {'label':'Introduction','attr':{'placeholder':"Votre introduction"}}) }}
    {{ form_row(form.description, {'label':'Description','attr':{'placeholder':"Votre description"}}) }}


    <button type="submit" class="btn btn-success">Enregistrer les modifications
    </button>

    {{ form_end(form) }}

</div>
</div>

{% endblock %}
```

Si on est connecté, les champs sont donc pré-remplis et les validations fonctionnent.

## XXVI) FORMULAIRE DE MODIFICATION DU MOT DE PASSE:

Je voudrais un vrai formulaire, mais je n'ai pas d'entité qui permette de gérer cela.
On va créer une "**fausse entité**" pour utiliser les validations: *( on ne veut pas que cette entité soit reflétée dans la base de donnée, on va supprimer toutes les annotations ORM) On se sert de symfony pour avoir une belle classe PHP*

**php bin/console make:entity PasswordUpdate**

oldPassword string 255 no
newPassword string 255 no
confirmPassword string 255 no

**!!!!!!!!!! ATTENTION ON NE FAIT PAS DE MIGRATION !!!!!!!**

On va dans **PasswordUpdate.php** et on supprime tous les ORM
Et on ajoute les contraintes
..

```
use Symfony\Component\Validator\Constraints as Assert;
….
/**
* @Assert\Length(
*      min = 8,
*      minMessage="Votre mot de passe doit faire un minimun de 8 caractères",
*)
*/
    private $newPassword;

/**
* @Assert\EqualTo(propertyPath="newPassword",message="Les deux mots de passe ne sont pas identiques")
*/
    private $confirmPassword;
```

On créé un nouveau formulaire:
**php bin/console make:form PasswordUpdateType**

⇨ On ne choisit pas d'entité car il n'y en a pas …

Dans **PasswordUpdateType.php**
On remplace:
```
->add('field_name')
```

Par
```
 ->add('oldPassword',PasswordType::class)
  ->add('newPassword',PasswordType::class)
  ->add('confirmPassword',PasswordType::class)
```

On créé une nouvelle fonction **updatePassword()** dans **AccountController.php**
```
      /**
   * permet de modifier le mot de passe
    * @Route("/account/password-update", name="account_password")
    */
      public function updatePassword(Request $request, EntityManagerInterface $manager,UserPasswordEncoderInterface $encoder)
      {

// recupérer l'utilisateur qui est connecté
       $user=$this->getUser();

       $passwordUpdate= new PasswordUpdate();

   $form = $this->createForm(PasswordUpdateType::class, $passwordUpdate ); // on relie les champs du formulaire aux champs de l'utilisateur

   $form ->handleRequest($request);


   if ($form->isSubmitted() && $form->isValid())
   {
                // verifie que le oldpassword est le même que celui de l'user
    if (!password_verify($passwordUpdate->getOldPassword(), $user->getHash())){

      $this->addFlash(
        'danger',
        'L\ancien mot de passe est incorrect !'
      );

     } else{
      $newPassword = $passwordUpdate->getNewPassword();
      $encoded = $encoder->encodePassword($user, $newPassword);
      $user->setHash($encoded);

                $manager->persist($user); // on fait persister dans le temps
                $manager->flush(); // envoi la requête => ecriture dans la base

                $this->addFlash(
                  'success',
                  'Le mot de passe a été modifié !'
                );

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

              }
```

```
        }

        return $this->render('account/password.html.twig' ,[
    'form' => $form -> createView(), // envoi du form à twig

    ]);
        }
```

Et **password.html.twig**

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

{% form_theme form 'bootstrap_4_layout.html.twig' %}


  {% block title %}Modification du mot de passe{% endblock %}

{% block body %}

  <div class="container">
    <h1>Modification du mot de passe</h1>

      {{ form_start(form) }}

    {{ form_row(form.oldPassword, {'label':'Ancien Mot de passe','attr':{'placeholder':"Ancien mot de passe"}}) }}
    {{ form_row(form.newPassword, {'label':'Nouveau Mot de passe','attr':{'placeholder':"Nouveau mot de passe"}}) }}
    {{ form_row(form.confirmPassword, {'label':'Confirmation Mot de passe','attr':{'placeholder':"Confirmation mot de
passe"}}) }}

    <button type="submit" class="btn btn-success">Enregistrer les modifications
    </button>

    {{ form_end(form) }}

  </div>
  </div>

{% endblock %}
```

## XXVII) PAGE PROFIL D'UN UTILISATEUR:

### 1) CREATION DU CONTROLLER:

On créé un nouveau controller:
**php bin/console make:controller UserController**

**UserController.php**

```
class UserController extends AbstractController
{
    /**
     * @Route("/user/{slug}", name="user_show")
     */
    public function index(User $user)
    {


        dump($user);
        return $this->render('user/index.html.twig', [
            'user' => $user,
        ]);
    }
}
```

**index.html.twig**

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

{% block title %}{{ user.firstName }} {{ user.lastName }}{% endblock %}

    {% block body %}
        <div class="container">
            <h1>Profil de {{ user.firstName }} {{ user.lastName }}</h1>

            <div class="row">
                <div class="col-3 text-center">
                    <img src="{{ user.picture }}" class="mb-3"><br/>
                    <span class="badge badge-primary">{{ user.ads | length }} annonces</span>

                </div>
                <div class="col-9">
                    {{ user.description}}
                </div>

            </div>

            <h2>Les annonces de de {{ user.firstName }} {{ user.lastName }}</h2>

                {% if (user.ads | length) > 0 %}

                    <div class="row">
                        {% for ad in user.ads %}
                            {#{{ dump(ad) }}#}

            Annonces de l'utilisateur

    {% endfor %}
                </div>

                {% else %}
                    <div class="row">
                        <div class="alert alert-warning">Vous n'avez pas d'annonces</div>
                    </div>
```

```
        {% endif %}

    </div>

{% endblock %}
```

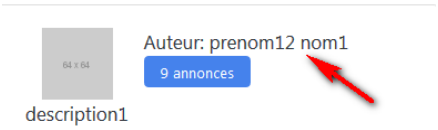
### 2) ANNONCES DE L'UTILISATEUR:

Il faut récupérer le contenu du *for* de ad/index.html.twig

Mais il faut éviter de dupliquer du code. On va donc mettre ce code dans un fichier à part **ad/_ad.html.twig** que l'on va récupérer ensuite dans **ad/index.html.twig** et **user/index.html.twig** avec {% include ('ad/_ad.html.twig') %}

### 3) LIENS VERS LA PAGE PROFIL DE L'UTILISATEUR:

Dans une annonce **ad/show.html.twig** , on rajoute le lien vers l'utilisateur




```
<a href="{{ path('user_show',{'slug':ad.author.slug}) }}">
    Auteur: {{ ad.author.firstName }} {{ ad.author.lastName }}
</a>
```

### 4) DONNER ACCES A SON COMPTE POUR L'UTILISATEUR:

On va créer une nouvelle route dans AccountController.php

```
    /**
 * permet d'afficher le profil de l'utilisateur connecté
 * @Route("/account/", name="account_index")
 */
public function myAccount(){


    return $this->render('user/index.html.twig' ,[
        'user' => $this->getUser(), // utilisateur connecté
    ]);

}
```

### 5) LIENS DE GESTION DU COMPTE UTILISATEUR:

Au niveau de l'affichage du profil, on veut rajouter des liens vers modifier le mot de passe ou modifier le profil

On va dans **user/index.html.twig**

```
<div class="col-9">
                {% if user is same as(app.user) %}
                    <div class="mt-3">
                        <a href="{{ path('account_profile') }}" class="btn btn-primary">Modifier mes
informations</a>
                        <a href="{{ path('account_password') }}" class="btn btn-primary">Modifier mot de
passe</a>
                    </div>
                {% endif %}

                    {{ user.description}}
                </div>
```

### 6) AJOUT D'UN DROPDOWN :

On va utiliser le code bootstrap suivant: https://getbootstrap.com/docs/4.0/components/dropdowns/

```
<div class="dropdown">
  <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Dropdown button
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
```

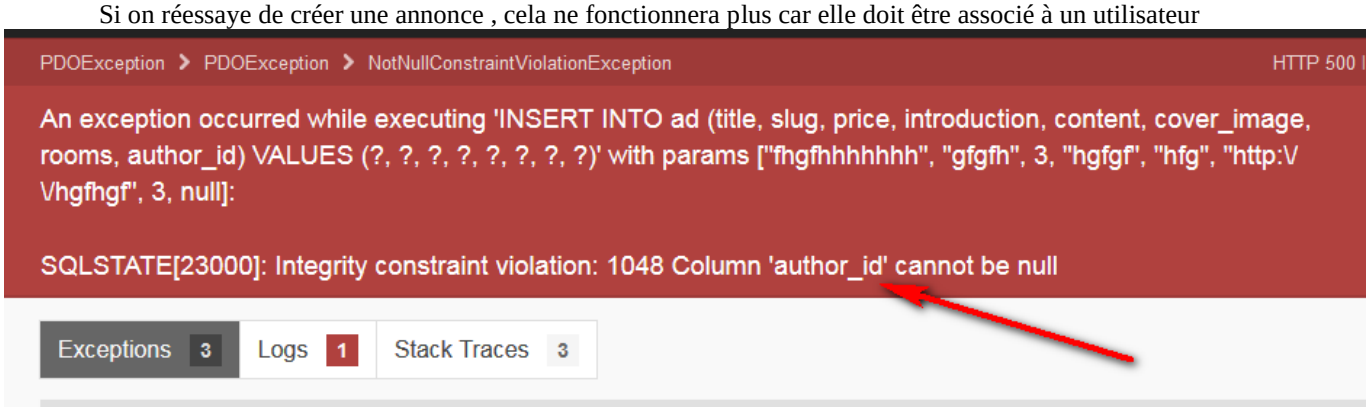
```
            <a class="dropdown-item" href="#">Action</a>
            <a class="dropdown-item" href="#">Another action</a>
            <a class="dropdown-item" href="#">Something else here</a>
          </div>
        </div>
```

On remplace *Mon compte* dans le **nav** qui est dans **header.html.twig**

```
    <div class="dropdown">
                <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                    {{ app.user.firstName }} {{ app.user.lastName }}
                </button>
                <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
                    <a class="dropdown-item" href="{{ path('account_index') }}">Mon compte</a>
                    <a class="dropdown-item" href="{{ path('account_profile') }}">Modifier mon profil</a>
                    <a class="dropdown-item" href="{{ path('ads_create') }}">Ajouter une annonce</a>
                    <a class="dropdown-item" href="{{ path('account_password') }}">Modifier mon mot de passe</a>
                    <a class="dropdown-item" href="{{ path('account_logout') }}">Déconnexion</a>
                </div>
            </div>
```

### 7) ATTRIBUER UNE NOUVELLE ANNONCE A UN UTILISATEUR:

Si on réessaye de créer une annonce , cela ne fonctionnera plus car elle doit être associé à un utilisateur




Donc on retourne dans le **AdController.php** et on rajoute

```
…..
   // si le form est soumis et valide (par rapport aux règles en place)
      if ($form->isSubmitted() && $form->isValid() ){

                $ad->setAuthor($this->getUser());
……
```

## XXVIII) ROLES DES UTILISATEURS (LES AUTORISATIONS):

### 1) CREATION DE L'ENTITE ROLE:

Qui a le droit d'aller où ? Qui peut faire quoi sur l'application ?
On créé une Entité pour gérer les rôles

**php bin/console make:entity Role**

title string 255 no
users relation User ManyToMany yes userRoles

**php bin/console make:migration**

**php bin/console doctrine:migrations:migrate**

### 2) MODIFICATION DE LA FIXTURE:

…..

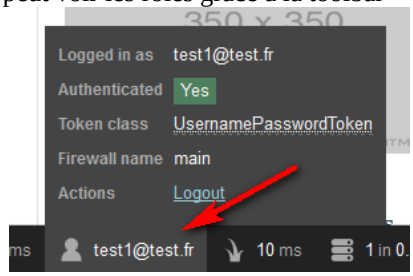
```
  public function load(ObjectManager $manager)
  {

      $adminRole= new Role();
      $adminRole ->setTitle('ROLE_ADMIN');
      $manager->persist($adminRole);

      $adminUser = new User();
      $adminUser ->setFirstName('Eric')
                 ->setLastName('Devolder')
                 ->setEmail('eric.devolder@ac-nice.fr')
                 ->setHash($this->encoder->encodePassword($adminUser, "password"))
                 ->setPicture('https://via.placeholder.com/64')
                 ->setIntroduction('Introduction de Eric')
                 ->setDescription('Description de Eric')
                 ->setSlug('eric-devolder')
                 ->addUserRole($adminRole);
      $manager->persist($adminUser);
```

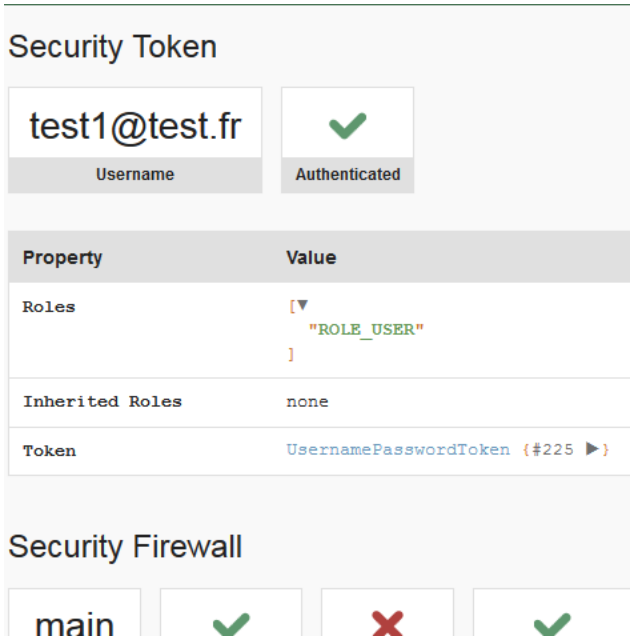
…….

**php bin/console doctrine:fixtures:load**

### 3) FOURNIR LES ROLES DES UTILISATEURS AU COMPOSANT DE SECURITE:

On peut voir les rôles grâce à la toolbar






Les rôles vient de la fonction **getRole()** de l'entité **User** qui est appelé par le composant de sécurité lors de la connexion d'un utilisateur.

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

  }
```

Donc à l'heure actuelle tous les utilisateurs ont le rôle ROLE_USER

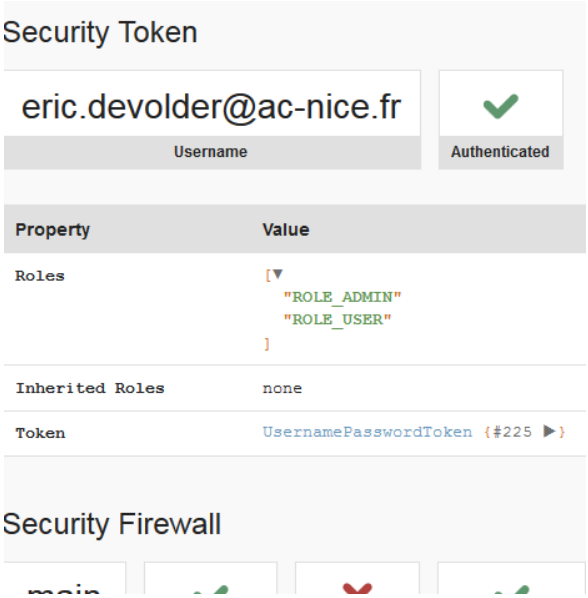
Il va falloir faire en sorte de récupérer aussi le rôle ROLE_ADMIN . On va donc modifier la fonction getRole()

$userRole est une ArrayCollection()  ( une espèce de super tableau, plus puissant, sur-couche d'un tableau )
https://www.doctrine-project.org/api/collections/latest/Doctrine/Common/Collections/ArrayCollection.html

```
    public function getRoles()  {

        foreach ($this->userRoles as $role)
          {
            //dump($role);
            //dump($role->getTitle());
          $roles[]=$role->getTitle();
          }

        /*
        // autre méthode avec map
        //Il y a une fonction map()  ( transforme les éléments du tableau en d'autres grâce à une fonction)
        //Et on transforme l'ArrayCollection en un tableau classique

        $roles = $this->userRoles->map(function($role){
            return $role->getTitle();
        })->toArray();*/

        $roles[]='ROLE_USER';

        //dump($roles);
        return $roles;
```

```
}
```

Si on se connecte maintenant avec l'admin, on aura:




### 4) SECURISER LE ADCONTROLLER AVEC LES ANNOTATION @ISGRANTED() ET @SECURITY():

https://symfony.com/doc/master/bundles/SensioFrameworkExtraBundle/annotations/security.html
C'est annotations serviront à sécuriser l'application. @SECURITY est un peu plus flexible que @ISGRANTED()
On les utilisera pour dire que telle action est uniquement accessible pour tel rôle.

On va sécuriser la page création d'une annonce pour être sur que c'est quelqu'un de connecté qui puisse y accéder.

- @IsGranted() : permet de s'assurer que l'utilisateur possède un certain rôle
- @security est plus flexible grâce aux expressions. https://symfony.com/doc/current/security/expressions.html

https://symfony.com/doc/master/bundles/SensioFrameworkExtraBundle/annotations/security.html

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
....
/**
 * @Route("/ads/new", name="ads_create")
 *@IsGranted("ROLE_USER")
 */
 public function create(Request $request,EntityManagerInterface $manager){

    $ad = new Ad();
....
```

Et si maintenant , on essaye d'accéder à la page de création d'une annonce alors que l'on est pas connecté, on est redirigé automatiquement vers la page de connexion **login**

Pour le cas d'édition d'une annonce, on veut que non seulement que l'on **soit connecté** mais aussi que l'on **soit créateur de l'annonce.**
On va utiliser @security

```
…
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
* permet d'afficher le formulaire d'édition
 * @Route("/ads/{slug}/edit", name="ads_edit")
```

```
*@Security("is_granted('ROLE_USER') and user === ad.getAuthor() " ,  message ="Cette annonce ne vous appartient pas")
*/
public function edit(Request $request,ObjectManager $manager,Ad $ad){

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

….
Et si maintenant , on essaye d'accéder à la page d'édition alors que l'on n'est pas le créateur.


### 5) AJOUT D'UN LIEN EDITION DE L'ANNONCE AU SEIN D'UNE ANNONCE:

Cela se joue an sein du code twig (est-ce que l'on fait apparaitre un bouton ou pas ? )
On va dans **show.html.twig**

```
{% if app.user and app.user == ad.author %}
        <a href="{{ path('ads_edit', {'slug':ad.slug}) }}" class="btn btn-primary mt-3 mb-3">Modifier l'annonce</a>
{% endif %}
```

De la même façon, on peut rajouter ce code au niveau des blocs de présentation des annonces dans le fichier **_ad.html.twig**

### 6) PERMETTRE AUX UTILISATEURS DE SUPPRIMER LEURS ANNONCES:

On va dans le controller **AdController.php** pour créer la fonction **delete()**

```
/**
* permet de supprimer une annonce
 * @Route("/ads/{slug}/delete", name="ads_delete")
 *@Security("is_granted('ROLE_USER') and user === ad.getAuthor() " ,  message ="Vous n'avez pas le droit d'accéder à cette ressource")
 *
 */
 public function delete(Ad $ad, EntityManagerInterface $manager){

$manager->remove($ad);
$manager->flush(); // envoyer l'info à la bdd

  $this->addFlash(
                'success',
                'L\'annonce '.$ad->getTitle().' a bien été supprimée !'
            );

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

On va maintenant dans **show.html.twig**

```
<a href="{{ path('ads_delete', {'slug':ad.slug}) }}" class="btn btn-danger mt-3 mb-3" onclick="return confirm('Etes-vous sur de vouloir supprimer l\'annonce ?')">Supprimer l'annonce</a>
```

Si on teste , on remarque que l'annonce et les images associées ont bien été supprimés

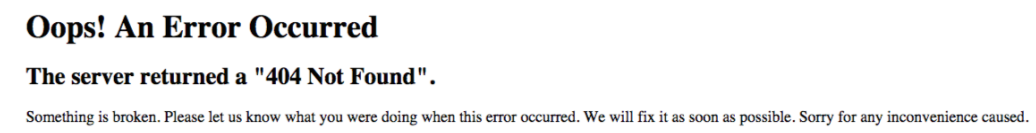
## XXIX) CUSTOMISER LES PAGES D'ERREURS:

### 1) INTRODUCTION:

https://symfony.com/doc/current/controller/error_pages.html

Dans l'environnement de développement , les erreurs sont interceptées et affichés par twig ( page rouge … )


En production , on aurait cette page.



## 2) PERSONNALISER LA PAGE 404 ET 403:

Les affichages qui montrent les différentes erreurs sont issus du bundle qui s'appelle le twig bundle. Si on veut mettre en place des affichages qui concerne ce bundle, il faut préciser certaines choses au niveau des templates.
https://symfony.com/doc/current/bundles/override.html
On va créer des dossiers /templates/**bundles/TwigBundle/Exception**
Il servira à faire des affichages personnalisés pour des bundles que nous n'avons pas développés mais qui sont installés au sein de symfony
On va expliquer à symfony que l'on va remplacer les templates qui concernent le TwingBundle
On créé dans le dossier Exception un fichier **error404.html.twig**

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

{% block title %}Page non trouvée{% endblock %}

    {% block body %}

        <div class="container">
            <p>LA PAGE N'EXISTE PAS
            </p>
            <a href="{{ path('homepage') }}" class="btn btn-primary">Retour à l'accueil</a>
        </div>
    {% endblock %}
```

Pour tester si la page fonctionne, il faut taper http://127.0.0.1:8000/_error/404

~~Ou alors passer en environnement de production dans **.env**~~

```
###> symfony/framework-bundle ###
APP_ENV=prod
```

~~Et taper une url qui n'existe pas => **à voir**~~

De la même façon, on peut créer **error403.html.twig**

## XXX) LES RESERVATIONS:

### 1) CREATION DE L'ENTITE BOOKING:

**php bin/console make:entity Booking**

booker relation User ManyToOne no yes bookings no => liaison entre reservation et utilisateur
ad relation Ad ManyToOne no yes bookings no => une reservation se fait par rapport à une annonce
startDate datetime no
endDate datetime no
createdAt datetime no
amount float no
comment text yes

**php bin/console make:migration**

**php bin/console doctrine:migrations:migrate**

### 2) AJOUT DE FAUSSES DONNEES:

Par exemple pour chaque annonce , nous voulons de 0 à 10 réservations
On rajoute dans la fixture

```
// gestion des réservations
    for ($j=0; $j <mt_rand(0,10) ; $j++) {
        $booking= new Booking();
        $booking->setCreatedAt(new \DateTime())
        -> setStartDate(new \DateTime("+ 5 days"))
        -> setEndDate(new \DateTime("+ 15 days"))
        ->setAmount($ad->getPrice()*10)
        ->setComment("Commentaire a la réservation $j")
        ->setAd($ad)
        ->setBooker($user);

        $manager->persist($booking);
        }
```

Et on lance
**php bin/console doctrine:fixtures:load**

### 3) FORMULAIRE DE RESERVATION:

**php bin/console make:form BookingType**

avec l'entité Booking

Dans le formulaire , je supprime (ce n'est pas à la personne de le renseigner):

```
    ->add('createdAt')
    ->add('amount')
    ->add('booker')
    ->add('ad')
```

Et on modifie:

```
$builder
  ->add('startDate',DateType::class,['widget'=>'single_text'])
   ->add('endDate',DateType::class,['widget'=>'single_text'])
   ->add('comment',TextareaType::class)
```

Rem: widget est une option prévue par Datetype

**php bin/console make:controller BookingController**

On modifie **BookingController.php**

```
namespace App\Controller;

use App\Entity\Ad;
use App\Entity\Booking;
use App\Form\BookingType;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class BookingController extends AbstractController
{
    /**
     * @Route("/ads/{slug}/book", name="booking_create")
      *@IsGranted("ROLE_USER")  // accessible qu'aux personnes connectés
     */
    public function book(Ad $ad)
    {

        $booking=new Booking();

        $form = $this->createForm(BookingType::class,$booking);

        return $this->render('booking/book.html.twig', [
            'ad' => $ad,
            'form' => $form->createView()
        ]);
    }
}
```

On modifie **book.html.twig**

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

{% form_theme form with ['bootstrap_4_layout.html.twig', "ad/_collection.html.twig"] %}


    {% block title %}Réserver l'annonce {{ ad.title }}{% endblock %}

        {% block body %}

            <div class="container">
                <h1> Réserver l'annonce {{ ad.title }}</h1>

                <p>Vous êtes sur le point de réserver l'annonce de {{ ad.author.firstName }} {{ ad.author.lastName }}</p>

                {{ form_start(form) }}
                <div class="row">
                    <div class="col">
                        {{ form_row(form.startDate, {'label':'Date de début','attr':{'placeholder':"date de début"}}) }}
                    </div>
                    <div class="col">
                        {{ form_row(form.endDate, {'label':'Date de fin','attr':{'placeholder':"Date de fin"}}) }}
                    </div>
                </div>
                <div class="row">
                    <div class="col">
                        {{ form_row(form.comment, {'label':'Commentaire','attr':{'placeholder':"Ecrire un commentaire"}}) }}
                    </div>
                </div>

                <h4>Montant du séjour: .....</h4>
```

```
            <span>0 nuit à {{ ad.price }} / nuit</span>

            <br/>
            <button type="submit" class="btn btn-success">Faire la réservation</button>

            {{ form_end(form) }}

        </div>

    {% endblock %}
```

**4) AJOUT DU LIEN RESERVATION SUR L'ANNONCE:**

Dans **show.html.twig**

```
<a href="{{ path('booking_create',{'slug':ad.slug}) }}" class="btn btn-primary mt-3 mb-3">Réserver!</a>
```

**5) ENREGISTRER UNE RESERVATION :**

On retourne dans **BookingController.php**

```
public function book(Ad $ad,Request $request,EntityManagerInterface $manager)
{

    $booking=new Booking();

    $form = $this->createForm(BookingType::class,$booking);


   $form ->handleRequest($request);

    // si le form est soumis et valide (par rapport aux règles en place)
    if ($form->isSubmitted() && $form->isValid() ){

        $user=$this->getUser(); // utilisateur connecté

        $booking->setBooker($user);
        $booking->setAd($ad);
        $booking->setCreatedAt(new \DateTime());

        // intervalle de date
        $diff = date_diff($booking->getStartDate(),$booking->getEndDate());
        //dump($diff);
        $booking->setAmount($ad->getPrice()*$diff->days);
        dump($booking);
        $manager->persist($booking);
        $manager->flush();

        return $this->redirectToroute('booking_show',['id'=>$booking->getId()]);

    }

    return $this->render('booking/book.html.twig', [
        'ad' => $ad,
        'form' => $form->createView()
    ]);
}



/**
 * @Route("/booking/{id}", name="booking_show")
 */
public function show(Booking $booking)
{
```

```
    return $this->render('booking/show.html.twig', [
            'booking' => $booking,
        ]);

    }

}
```

Création de la page de view de réservation ( show.html.twig):

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

{# nbre de jours de réservation #}
{% set difference = date(booking.endDate).diff(date(booking.startDate)) %}
{% set NumberDays = difference.days %}

    {% block title %}Réservation n°{{ booking.id }}{% endblock %}

        {% block body %}

            <div class="container">
                <h1>Votre réservation n° {{ booking.id }}</h1>
                <div class="row">
                    <div class="col">
                        <div class="alert alert-light">
                            <p>Détails</p>
                            <ul class="list-group">
                              <li class="list-group-item">Numéro: {{ booking.id }}</li>
                              <li class="list-group-item">Date d'arrivée: {{ booking.startDate |date("d/m/Y") }}</li>
                              <li class="list-group-item">Date de départ: {{ booking.endDate |date("d/m/Y") }}</li>
                              <li class="list-group-item">Nombre de nuit: {{ NumberDays }}</li>
                              <li class="list-group-item">Montant total: {{ booking.amount }}</li>
                              <li class="list-group-item">Commentaire: {{ booking.comment | default('Aucun commentaire') }}</li>
                            </ul>

                        </div>

                        <h2>Propriétaire</h2>
                        <div class="row">
                            <div class="col">
                                <img src="{{ booking.ad.author.picture }}"/>
                            </div>
                            <div class="col">
                                <h4><a href="{{ path('user_show',{'slug':booking.ad.author.slug}) }}">{{ booking.ad.author.firstName }}
{{ booking.ad.author.lastName }}</a></h4>
                                <span class="badge badge-primary">{{ booking.ad.author.ads|length }} annonces</span>

                            </div>

                        </div>
                            {{ booking.ad.author.description | raw }}
                        </div>


                        <div class="col">
                            <div class="alert alert-light">
                                <h2>Votre hébergement </h2>
                                    <h4><a href="{{ path('ads_show',{'slug':booking.ad.slug, 'id':booking.ad.id }) }}">{{ booking.ad.title
}}</a></h4>
                                    <img src="{{ booking.ad.coverImage }}" alt="image de {{ booking.ad.title }}" class="img-fluid" />
                                    {{ booking.ad.content | raw }}
                            </div>
```

```
            </div>
        </div>
    </div>

{% endblock %}
```

On veut que cette page s'affiche aussi bien lors d'une réservation effectuée avec succès que lors d'une simple consultation.
Donc on voudrait mettre une alert si on a une url du type **/booking/199?withAlert=true** pour pouvoir récupérer éventuellement le paramètre GET
On peut accéder à la requête via twig avec la variable **app.request** en tapant **app.request.query.get**
On aura :

```
{% if app.request.query.get('withAlert') %}
        <div class="alert alert-success">
            <h4>Bravo , la réservation a été effectuée avec success</h4>
        </div>

    {% endif %}
```

On va part contre modifier la fonction book de **BookingController.php**

```
return $this->redirectToroute('booking_show',['id'=>$booking->getId(),
                                'withAlert'=> true]);
```

Vue que l'id est déjà présent dans `* @Route("/booking/{id}", name="booking_show")`
withAlert se mettra automatiquement en GET

### 6) VALIDATION DE LA RESERVATION :

Dans l'entité **Booking.php ,** on rajoute use Symfony\Component\Validator\Constraints as Assert;
On veut valider les dates

```
/**
 * @ORM\Column(type="datetime")
 * @Assert\Date(message="Attention, la date d'arrivée doit être au bon format !")
 */
private $startDate;

/**
 * @ORM\Column(type="datetime")
* @Assert\Date(message="Attention, la date de départ doit être au bon format !")
 */
private $endDate;
```

On veut faire en sorte également que le commentaire ne soit pas obligatoire.
Dans **BookingType.php**

```
->add('comment',TextareaType::class,['required' => false])
```

### 7) DISPONIBILITE D'UNE ANNONCE LORS D'UNE RESERVATION:

On veut éviter que l'appartement soit loué par deux personnes dont les dates se chevauchent.

- Il faut connaitre les dates qui sont impossibles pour l'annonce
- Il faut comparer les dates choisies avec les dates impossibles
- répond true ou false

Dans le controller **BookingController.php ,** on va récupérer toutes les réservations de l'annonce considérée avec BookingRepository

```
  /**
   * @Route("/ads/{slug}/book", name="booking_create")
    *@IsGranted("ROLE_USER")  // accessible qu'aux personnes connectés
   */
 public function book(Ad $ad,Request $request,EntityManagerInterface $manager,BookingRepository $repo)
  {

     //***************** début liste dates non disponibles pour la réservation **********************************
          // initialisation du tableau contenant les dates non dispo
            $notAvailableDays=[];

            // il faut connaitre les dates qui sont impossibles pour l'annonce
            // on recupère toutes les réservations déjà existantes de l'annonce
            $repo = $repo->findBy(array('ad'=>$ad->getId()));
            //dump($repo);

            // on boucle sur toutes les réservations déjà faites
            foreach ($repo as $item) {

              //on crée un tableau de timestamp en utilisant la fonction range() ( intervalle entre deux éléments avec un certain
step)
            $tous_les_timestamp=range($item->getStartDate()->getTimestamp(),$item->getEndDate()-
>getTimestamp(),24*60*60);

            // on fusionne tous les tableaux range
              $notAvailableDays=array_merge($notAvailableDays,$tous_les_timestamp);
            }

            //supprimer doublons
            $notAvailableDays=array_unique($notAvailableDays);

            // réinitialiser les clés
            $notAvailableDays=array_values($notAvailableDays);
            //dump($notAvailableDays);


    //***************** fin liste dates non disponibles pour la réservation **********************************

      $booking=new Booking();

      $form = $this->createForm(BookingType::class,$booking);


    $form ->handleRequest($request);

      // si le form est soumis et valide (par rapport aux règles en place)
      if ($form->isSubmitted() && $form->isValid() ){

            $user=$this->getUser(); // utilisateur connecté

            $booking->setBooker($user);
            $booking->setAd($ad);
            $booking->setCreatedAt(new \DateTime());

            //************* debut calcul du prix de la réservation ******************
            // intervalle de date
            $diff = date_diff($booking->getStartDate(),$booking->getEndDate());
            //dump($diff);
            $booking->setAmount($ad->getPrice()*$diff->days);
            //dump($booking);
            //************* fin calcul du prix de la réservation ******************
```

```
            // l'intervalle de date entre les deux dates choisies de la réservation
        $tous_les_timestamp_choisies=range($booking->getStartDate()->getTimestamp(),$booking->getEndDate()->getTimestamp(),24*60*60);
        //dump($tous_les_timestamp_choisies);

        //initialisation de datesOK
        $datesOK=true;

        // on boucle sur tous les timestamp des dates choisies
            foreach ($tous_les_timestamp_choisies as $value) {

             // on regarde si un timestamp choisi est dans le tableau $notAvailableDays
             if (array_search($value,$notAvailableDays)) {$datesOK=false;break;}

            }


          // si les dates ne sont pas disponibles , message d'erreur
            if (!$datesOK){

             $this->addFlash(
                'warning',
                "Les dates choisies ne sont pas disponibles"
             );


            } else {

            //sinon enregistrement et redirection

            $manager->persist($booking);
            $manager->flush();

            return $this->redirectToroute('booking_show',['id'=>$booking->getId(),
                                          'withAlert'=> true
                                      ]);

            }




    }

    return $this->render('booking/book.html.twig', [
        'ad' => $ad,
        'form' => $form->createView(),
        'notAvailableDays'=> $notAvailableDays
    ]);
  }
```

**<u>REMARQUE:</u> pour déterminer** $notAvailableDays , on pourrait plutôt créer une fonction **getNotAvailabledays()** dans l'entité Ad (puisque dans book() , on a accès à Ad ). On y ferait alors appel dans book() de cette façon:
$notAvailableDays = $ad->getNotAvailableDays();

Avec dans App\Entity\Ad;:

```
            public function getNotAvailableDays(){

                // initialisation du tableau contenant les dates non dispo
                $notAvailableDays=[];

                // il faut connaitre les dates qui sont impossibles pour l'annonce
```

```
    // on recupère toutes les réservation de l'annonce
    foreach ($this->bookings as $booking) {

      //on crée un tableau en utilisant la fonction range() ( intervalle entre deux éléments avec un certain step)
    $tous_les_timestamp=range($booking->getStartDate()->getTimestamp(),$booking->getEndDate()->getTimestamp(),24*60*60);

    // on fusionne tous les tableaux range
     $notAvailableDays=array_merge($notAvailableDays,$tous_les_timestamp);
    }

    //supprimer doublons
    $notAvailableDays=array_unique($notAvailableDays);

    // réinitialiser les clés
    $notAvailableDays=array_values($notAvailableDays);
    dump($notAvailableDays);


    return $notAvailableDays;

 }
```

## 8) UTILISATION D'UN CALENDRIER JAVASCRIPT:

### a) Datepicker uxsolutions:

On ne connait les dates disponibles que lors de la validation du formulaire ( côté back). Il serait bien de voir déjà sur le calendrier du formulaire les dates disponibles.
On va utiliser **bootstrap datepicker uxsolutions**
https://github.com/uxsolutions/bootstrap-datepicker

https://bootstrap-datepicker.readthedocs.io/en/stable/

***Usage***
Call the datepicker via javascript:
$('.datepicker').datepicker();

https://bootstrap-datepicker.readthedocs.io/en/stable/options.html#datesdisabled

On utilise https://cdnjs.com/libraries/bootstrap-datepicker

⇨ https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/css/bootstrap-datepicker.min.css
⇨ https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/js/bootstrap-datepicker.min.js

On va dans book.html.twig et on surcharge le bloc stylesheets et javascript

```
{% block stylesheets %}
<link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/css/bootstrap-datepicker.min.css">
{% endblock %}


{% block javascripts %}
```

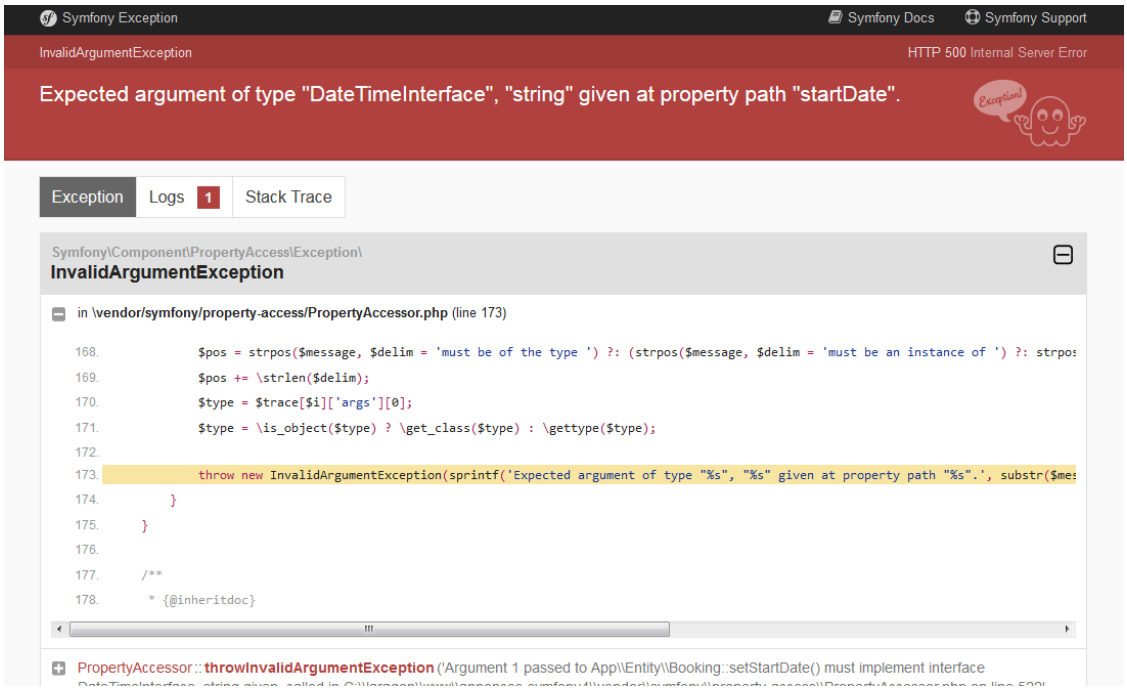
```
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/js/bootstrap-datepicker.min.js"></script>

<script>

   $(document).ready(function(){

// on détermine #booking_startDate et #booking_endDate en explorant le code des inputs
      $('#booking_startDate , #booking_endDate').datepicker({
                        format: 'dd/mm/yyyy',
                          datesDisabled: [
                                    {% for day in notAvailableDays %}
                                          "{{ day | date('d/m/Y') }}",
                                     {% endfor %}

                          ],
                          startDate: new Date()
                                });

   })

</script>
```

Il faut également spécifier au niveau du formulaire **BookingType.php** que l'on a maintenant de simple champ text (sinon on a deux calendriers)

```
      $builder
          ->add('startDate',TextType::class)
          ->add('endDate',TextType::class)
```

### b) Datatransformers:

Les **datatransformers** transforment des données au moment où on les passe à un formulaire ou au moment où le formulaire nous les passe.
Mais il y a un problème maintenant au niveau du format de la date que symfony attend. Il attend un dateTime, or on lui fournit un string




On doit transformer les dates du formulaire. On doit passer par les **datatransformers qui permettent de transformer les donnée d'un formulaire.**

**Les données sont transformées pour se conformer à ce qu'elles devraient être.**
https://symfony.com/doc/current/form/data_transformers.html

https://symfony.com/doc/current/form/data_transformers.html#creating-the-transformer
On créé un dossier **Form/DataTransformer** avec un fichier **FrenchToDateTimeTransformer.php**

Si on regarde l'interface
https://github.com/symfony/symfony/blob/2.7/src/Symfony/Component/Form/DataTransformerInterface.php
On voit que la classe doit implémenter deux fonctions ( transform et reverseTransform)

- **transform doit transformer les données que l'on envoit au formulaire (DateTime -> string)**
- **reverseTransform doit transformer les données que l'on reçoit du formulaire (string -> DateTime)**

**https://www.php.net/manual/fr/datetime.format.php**

```
<?php


namespace App\Form\DataTransformer;

use Symfony\Component\Form\DataTransformerInterface;
use Symfony\Component\Form\Exception\TransformationFailedException;

class FrenchToDateTimeTransformer implements DataTransformerInterface {

// lorsque l'on passe des données au formulaire
      public function transform($date){


          if ($date === null){

          return '';

          }

      return $date->format('d/m/Y');

      }

 // On reçoit des données du formulaire
public function reverseTransform($frenchDate){

// frenchDate = 17/03/2019

/*
      if ($frenchDate === null){
         // exception
            throw new TransformationFailedException("Vous devez fournir une date !");


      }*/
 // https://www.php.net/manual/fr/datetime.createfromformat.php

      $date = \DateTime::createFromFormat('d/m/Y',$frenchDate);

//met l'heure à 0:0:0 car sinon problème de timestamp qui prends en compte l'heure par rapport à $notAvailableDays qui est avec une heure nulle
      $date->setTime(0,0);


/*
      if ($date === false){
```

```
            // exception
                throw new TransformationFailedException("Le format de la date n'est pas le bon !");

        }*/

    return $date;
}


}
```

On pourra dire à symfony. Attention sur le champ date , il faudra faire passer la transformation

Au sein de **BookingType.php** , on va avoir besoin du transformer ( on va faire de l'injection de dépendance via la constructeur )
https://symfony.com/doc/current/form/data_transformers.html#using-the-transformer

```
    private $transformer;

 // injection de dépendance
    public function __construct(FrenchToDateTimeTransformer $transformer){

        $this->transformer = $transformer;

    }
```

Et au sein de buildForm , on rajoute:

```
        $builder->get('startDate')->addModelTransformer($this->transformer);
        $builder->get('endDate')->addModelTransformer($this->transformer);
```

### c) Validation des dates:

Maintenant à cause des champs texte du formulaire , on peut mettre ce que l'on veut
On va utiliser la contrainte GreaterThan https://symfony.com/doc/current/reference/constraints/GreaterThan.html
Dans l'entité Booking.php on met:

```
    /**
     * @ORM\Column(type="datetime")
     * @Assert\Date(message="Attention, la date d'arrivée doit être au bon format !")
     * @Assert\GreaterThan("today", message="la date d'arrivée doit être ultérieur à aujourd'hui !")
     */
    private $startDate;

    /**
     * @ORM\Column(type="datetime")
    * @Assert\Date(message="Attention, la date de départ doit être au bon format !")
    * @Assert\GreaterThan(propertyPath="startDate", message="La date de départ doit être supérieur à la date d'arrivée")
     */
    private $endDate;
```

## 9) CALCULER LE NOMBRE DE NUITS ET LE MONTANT DE LA RESERVATION EN JAVASCRIPT:

On va dans la partie javascript du fichier twig **book.html.twig**
On crée une fonction:

```
    function calculateAmount(){
            // on chope les dates
            // Mais on est au format francais, donc il faut que javascript comprenne
            //On va utiliser la fonction replace avec une expression régulière et une capture avec les parenthèses
            // 17/03/2019 => ([0-9]{2})\/([0-9]{2})\/([0-9]{4})

            const endDate = new Date($('#booking_endDate').val().replace(/([0-9]{2})\/([0-9]{2})\/([0-9]{4})/,'$3-$2-$1'));
```

```
        const startDate = new Date($('#booking_startDate').val().replace(/([0-9]{2})\/([0-9]{2})\/([0-9]{4})/,'$3-$2-$1'));

        if (startDate && endDate && startDate < endDate){

            const DAY_TIME = 24 * 60 * 60 *1000; // nbre de miliseconde dans une journée
            // timestamp ( en javascript => milliseconde)
            const interval = endDate.getTime() - startDate.getTime();
            const days = interval / DAY_TIME;
            const amount = days * {{ ad.price }};

            $('#days').text(days);
            $('#amount').text(amount);

        }

    }
```

Et on rajoute un évènement au niveau de $(document).ready
```
$('#booking_startDate , #booking_endDate').on('change',calculateAmount);
```

Et on modifie:

```
    <h4>Montant du séjour: .....</h4>
⇨   <h4>Montant du séjour: <span id="amount">…</span>  </h4>

    <span>0 nuit à {{ ad.price }} / nuit</span>
⇨   <span id="days">0</span>  nuit à {{ ad.price }} / nuit
```

**10) <u>CREER UNE PAGE QUI AFFICHE LES RESERVATIONS D'UN UTILISATEUR:</u>**

On va dans **AccountController.php** et on va créer une nouvelle fonction **bookings()** et un nouveau template **account/bookings.html.twig**

Dans **AccountController.php**

```
/**
*Permet d'afficher la liste des informations faites par l'utilisateur
* @Route("/account/bookings", name="account_bookings")
*/
    public function bookings(){


        return $this->render('account/bookings.html.twig');

    }
```

Dans **bookings.html.twig  (on utilize la grosse variable app )**

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


    {% block title %}Vos Réservations{% endblock %}

    {% block body %}
        <div class="container">
            <h1>Mes réservations</h1>

            <div class="alert alert-info">
                <p>Retrouvez toutes vos réservations</p>

            </div>
```

```
        {% for booking in app.user.bookings %}

            <div class="row">
                <div class="col">
                    <img src="{{ booking.ad.coverImage }}" alt="Photo de l'annonce {{ booking.ad.title }}" class="img-fluid">
                </div>

                <div class="col">
                    <h4>{{ booking.ad.title }}</h4>

                    <p>

                        Réservation n° {{ booking.id }}<br>
                        du {{ booking.startDate|date('d/m/Y') }} au {{ booking.endDate|date('d/m/Y') }}
                        ( {{ booking.amount }} € )
                    </p>

                    <a href="{{ path('booking_show',{'id':booking.id}) }}">Plus d'informations</a>
                </div>

            </div>

        {% endfor %}


    </div>

{% endblock %}
```

Et on rajoute dans le menu déroulant de l'utilisateur un lien vers ses réservations (**header.html.twig**)

```
<a class="dropdown-item" href="{{ path('account_bookings') }}">Mes réservation</a>
```

## XXXI) AVIS DES VISITEURS CONCERNANT UNE RESERVATION:

On veut donner la possibilité de laisser un commentaire à la fin de la réservation

### 1) ENTITE COMMENTAIRE:

**php bin/console make:entity Comment**

createdAt datetime no
rating integer no
content texte no
ad  relation Ad ManyToOne no yes comments yes
author relation User ManyToOne no yes comments yes

**php bin/console make:migration**
**php bin/console doctrine:migrations:migrate**

### 2) MODIFICATION DE LA FIXTURE:

On va agir au niveau d'une réservation

```
    // gestion des commentaires des fin de réservations
    if (mt_rand(0,1)) {
        $comment=new Comment();
        $comment ->setCreatedAt(new \DateTime())
                -> setRating(mt_rand(0,5))
                ->setContent("Commentaire fin de réservation N°$j")
                ->setAd($ad)
                ->setAuthor($user);

        $manager->persist($comment);
    }
```

**php bin/console doctrine:fixtures:load**

### 3) AFFICHAGE DES COMMENTAIRES SUR LA PAGE D'UNE ANNONCE:

On veut afficher les commentaires sur la page de l'annonce correspondante
On va dans **ad/show.html.twig** en dessous du caroussel

```
{% if ad.comments|length > 0 %}
    <h2>Commentaires des voyageurs</h2>

    {% for comment in ad.comments %}
      <div class="row alert-light mb-3">
        <div class="col">
          <strong>{{ comment.author.firstName }} {{ comment.author.lastName }}</strong><br>
          <blockquote>{{ comment.content }}</blockquote>
          <strong>Note: </strong>{{ comment.rating }}
        </div>
      </div>
    {% endfor %}

  {% else %}
    <h2>il n'y a pas de commentaires</h2>
  {% endif %}
```

### 4) AFFICHER LES NOTES SOUS FORME D'ETOILES:

On veut afficher les notes sous forme d'étoiles. On va utiliser FontAwesome ( donne la possibilité de créer des icones)
https://fontawesome.com/ => https://fontawesome.com/start
On copie le CDN dans **base.html.twig**

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous">
```

On voit comment afficher des étoiles:
https://fontawesome.com/icons/star?style=solid
https://fontawesome.com/icons/star?style=regular

On va donc modifier la page: **ad/show.html.twig**

```
    {% if ad.comments|length > 0 %}
    <h2>Commentaires des voyageurs</h2>

    {% for comment in ad.comments %}
      <div class="row alert-light mb-3">
        <div class="col">
          <strong>{{ comment.author.firstName }} {{ comment.author.lastName }}</strong><br>
          <blockquote>{{ comment.content }}</blockquote>
          <strong>Note: </strong>{{ comment.rating }}

          {% for i in 1..5 %}
            {% if i <= comment.rating %}
              <i class="fas fa-star"></i>
            {% else %}
              <i class="far fa-star"></i>
            {% endif %}
          {% endfor %}

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

  {% else %}
    <h2>il n'y a pas de commentaires</h2>
  {% endif %}
```

Rem: On peut créer éventuellement son propre fichier d'icônes https://icomoon.io/app/#/select (on aura alors une taille plus réduite) et le minifier ensuite: http://www.minifycss.com/css-compressor/ ou https://cssminifier.com/

### 5) CALCULER ET AFFICHER LA NOTE MOYENNE D'UNE ANNONCE:

On veut calculer la moyenne générale d'une annonce( moyenne de toutes les notes données)
On va dans le template **ad/show.html.twig** et on ajoute pour tester.

```
<div class="alert alert-info">
  <div class="row">
    <div class="col">
      Note Globale sur {{ ad.comments|length }} commentaires
    </div>
    <div class="col">
        <i class="fas fa-star"></i>
        <i class="far fa-star"></i>
    </div>
  </div>
</div>
```

Pour calculer la moyenne, on va se rendre dans l'entité annonce **Ad.php** et on va créer une fonction

```
// calcul de la moyenne des notes d'une annonce
 public function getAvgRatings(){
     // calculer la somme des notations
        $somme=0;
        //dump($this->comments);
        foreach ($this->comments as $value) {
          // dump($value);
           $somme = $somme + $value->getRating();
        }

        // calcul de la moyenne
      if (count($this->comments) > 0)  return $somme/count($this->comments);
      return 0;

 }
```

On peut donc modifier **ad/show.html.twig**

```
<div class="alert alert-info">
            <div class="row">
              <div class="col">
                 Note Globale sur {{ ad.comments|length }} commentaires
              </div>
              <div class="col">
                    {% for i in 1..5 %}
                      {% if i <= ad.getAvgRatings %}
                        <i class="fas fa-star"></i>
                      {% else %}
                        <i class="far fa-star"></i>
                      {% endif %}
                    {% endfor %}
              </div>
            </div>
          </div>
```

Et ajouter également la note dans l'entête de l'annonce et sur toutes les annonces ( ce qui fait du code dupliqué)
On va donc le factoriser. On met le code

```
                    {% for i in 1..5 %}
                      {% if i <= rating %}
                        <i class="fas fa-star"></i>
                      {% else %}
                        <i class="far fa-star"></i>
                      {% endif %}
```

```
{% endfor %}
```

Dans un fichier **partials/rating.html.twig**
Et on y fera appel avec un include, par exemple:

```
{% include 'partials/rating.html.twig' with {'rating': ad.getAvgRatings} %}
```

### 6) AFFICHAGE DU FORMULAIRE DE COMMENTAIRE SUR UNE RESERVATION:

On doit pouvoir laisser le commentaire à la fin de la réservation

On créé le formulaire: **php bin/console make:form CommentType**
Sur l'entité Comment
On ne garde que:

```
->add('rating',IntegerType::class,['attr'=>
                                    ['min' => 0, 'max' => 5]
                                ])
->add('content')
```

On va dans le controller **BookingController.php** dans la fonction **show()**

```
/**
 * @Route("/booking/{id}", name="booking_show")
 */
public function show(Booking $booking,Request $request, EntityManagerInterface $manager)
{
  $comment = new Comment();

  $form= $this->createForm(CommentType::class ,$comment);

$form ->handleRequest($request);

if ($form->isSubmitted() && $form->isValid())
 {
    $comment->setAd($booking->getAd());
    $comment->setCreatedAt(new \DateTime());
    $comment->setAuthor($booking->getBooker());

    //dump($booking->getBooker());
    //dump($this->getUser());
    $manager->persist($comment);
    $manager->flush();


     $this->addFlash(
              'success',
              'Le commentaire a bien été enregistrée !'
          );
 }

 return $this->render('booking/show.html.twig', [
       'booking' => $booking,
       'form' => $form->createView()
    ]);

 }
```

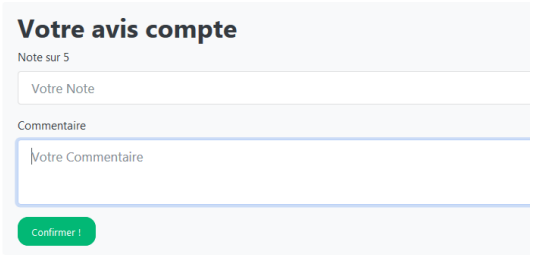
Il faut modifier le template **booking/show.html.twig**

```
<div class="row">
            <div class="col">
              <div class="alert alert-light">
```

```
            <h2>Votre avis compte</h2>
            {% if date() > date(booking.endDate) %}

                {{ form_start(form) }}

                {{ form_row(form.rating, {'label':'Note sur 5','attr':{'placeholder':"Votre Note"}}) }}
                {{ form_row(form.content, {'label':'Commentaire','attr':{'placeholder':"Votre Commentaire"}}) }}

                <button type="submit" class="btn btn-success">Confirmer !
                </button>

                {{ form_end(form) }}

            {% else %}
                <p>Revenez à la fin de votre voyage pour pouvoir noter votre commentaire</p>
            {% endif %}
        </div>
    </div>
</div>
```

**7) <u>S'ASSURER QU'UN VISITEUR NE COMMENTE QU'UNE SEULE FOIS UNE ANNONCE:</u>**

Une fois que le visiteur a commenté l'annonce, on ne veut plus voir le formulaire mais voir un résumé du commentaire




On va dans l'entité annonce Ad.php et on va créer une fonction:

```
// permet de récupérer le commentaire d'un auteur par rapport à une annonce
public function getCommentFromAuthor(User $author){
    foreach ($this->comments as $comment ) {

        // dump($author);

        if ($comment->getAuthor() === $author) return $comment;

        return null;

    }
```

Et on met à jour **booking/show.html.twig**

```
{#
    {{ dump(booking.ad.getCommentFromAuthor(booking.booker)) }}

    {{ dump(booking.ad.getCommentFromAuthor(app.user)) }}
    #}

    <h2>Votre avis compte</h2>
    {% if date() > date(booking.endDate) %}
```

```
                {% if (booking.ad.getCommentFromAuthor(booking.booker) is  null )  %}
                    {{ form_start(form) }}

                    {{ form_row(form.rating, {'label':'Note sur 5','attr':{'placeholder':"Votre Note"}}) }}
                    {{ form_row(form.content, {'label':'Commentaire','attr':{'placeholder':"Votre Commentaire"}}) }}

                    <button type="submit" class="btn btn-success">Confirmer !
                    </button>

                    {{ form_end(form) }}
                {% else %}
                    <blockquote>
                        Commentaire:  {{ booking.ad.getCommentFromAuthor(booking.booker).content }}
                      <br>
                        Note:  {% include 'partials/rating.html.twig' with {'rating':
booking.ad.getCommentFromAuthor(booking.booker).rating} %}

                {% endif  %}

            {% else %}
              <p>Revenez à la fin de votre voyage pour pouvoir noter votre commentaire</p>
            {% endif %}
```

## XXXII) ADMINISTRATION DU SITE:

### 1) CONTROLEUR DES ANNONCES:

On va en gros avoir autant de contrôleurs pour la partie administration que pour la partir front.
On commence par créer un contrôleur d'administration pour les annonces.

**php bin/console make:controller AdminAdController**

created: src/Controller/AdminAdController.php
created: templates/admin_ad/index.html.twig

Il a été créé **admin_ad/index.html.twig** , on préfère plutôt avoir **admin/ad/index.html.twig**
On modifie donc.. ( On devra le faire à chaque fois)
On modifie également le controleur.

```
<?php

namespace App\Controller;

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

class AdminAdController extends AbstractController
{
    /**
     * @Route("/admin/ads", name="admin_ads_index")
     */
    public function index(AdRepository $repo)
    {
        return $this->render('admin/ad/index.html.twig', [
            'ads' => $repo -> findAll(),
        ]);
    }
}
```

Et également **admin/ad/index.html.twig** en utilisant les tables bootstrap
https://getbootstrap.com/docs/4.0/content/tables/

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

{% block title %}Administration des annonces{% endblock %}

{% block body %}

    <div class="container-fluid">
        <h1>Gestion des annonces</h1>

            <table class="table">
              <thead>
                <tr class="text-center">
                  <th scope="col">#</th>
                  <th scope="col">Id</th>
                  <th scope="col">Titre</th>
                  <th scope="col">Auteur</th>
                  <th scope="col">Réservations</th>
                  <th scope="col">Note</th>
                  <th scope="col">Actions</th>
                </tr>
              </thead>
              <tbody>

                {% for ad in ads %}
                <tr class="text-center">
                  <th scope="row">{{ loop.index0 }}</th>
                  <td>{{ad.id}}</td>
                  <td>{{ad.title}}</td>
                  <td>{{ad.author.firstName}} {{ad.author.lastName}}</td>
                  <td>{{ad.bookings | length}}</td>
```

```
            <td>{{ad.getAvgRatings|number_format(2, '.', ',')}}</td>
            <td>
               <a href="#" class="btn btn-primary"><i class="fas fa-edit"></i></a>
              <a href="#" class="btn btn-danger"><i class="fas fa-trash-alt"></i></a>

            </td>
          </tr>
          {% endfor %}

        </tbody>
      </table>




  </div>


{% endblock %}
```

**2) <u>HABILLAGE DIFFERENT POUR L'ADMINISTRATION:</u>**

On va créer dans le dossier: /admin

- Un fichier base.html.twig
- Un dossier partials
  - Un fichier /partials/header.html.twig
  - Un fichier /partials/footer.html.twig

On prend le contenu du base.html.twig ainsi que le header et footer du front et on modifie les différents liens

**3) <u>MISE A JOUR BARRE DE NAVIGATION:</u>**

Mise à jour de la barre dans /admin//partials/header.html.twig

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <a class="navbar-brand" href="{{ path('homepage') }}">Site Annonces</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor03" aria-
controls="navbarColor03" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarColor03">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="#">DashBoard<span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item ">
        <a class="nav-link" href="{{ path('admin_ads_index') }}">Annonces</a>
      </li>
      <li class="nav-item ">
        <a class="nav-link" href="#">Réservations</a>
      </li>
      <li class="nav-item ">
        <a class="nav-link" href="#">Commentaires</a>
      </li>
      <li class="nav-item ">
        <a class="nav-link" href="#">Utilisateurs</a>
      </li>

    </ul>
    <ul class="navbar-nav ml-auto">

      <li class="nav-item ">
        <a class="nav-link" href="{{ path('homepage') }}" target="_blank">Voir le site</a>
```

```
        </li>

        {% if (app.user) %}
          <li class="nav-item active">

            <div class="dropdown">
              <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                {{ app.user.firstName }} {{ app.user.lastName }}
              </button>
              <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
                <a class="dropdown-item" href="{{ path(' admin_account_logout') }}">Déconnexion</a>
              </div>
            </div>

          </li>

        {% endif %}

      </ul>

    </div>
</nav>

      {#
      {{ dump(app) }}
      {{ dump(app.flashes) }}
      #}

      {% for label, messages in app.flashes %}

        <div class="alert alert-{{ label }} alert-dismissible fade show" role="alert">

          {% for message in messages %}
            <p>{{ message | raw }}</p>
          {% endfor %}

          <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>

      {% endfor %}
```

**4) <u>PROTEGER L'ACCES A TOUTE L'ADMINISTRATION:</u>**

On va dans config/packages/security.yaml et on va pouvoir sécuriser l'administration en une seul ligne
On va gérer les ACL (permet de faire une gestion plus générale des accès en fonction des rôles)
Dans acces_control, On décommente:

```
- { path: ^/admin, roles: ROLE_ADMIN }
```

Tout ce qui commence par /admin nécessite le role_admin

Si maintenant on actualise l'url /admin/ads , on va être redirigé vers le formulaire de login . Mais on va créer un autre formulaire de login pour l'admin.

**a) <u>Page de connexion spécifique pour l'admin ( et deconnexion):</u>**

On va créer un nouveau controleur

**php bin/console make:controller AdminAccountController**

On va organiser différemment . On ne veut pas de /admin_account/index.html.twig. On va plutôt créer un dossier account dans le dossier admin et on déplace index.html.twig que l'on renomme login.html.twig
On modifie ensuite le controller en créant la fonction **login et logout**

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class AdminAccountController extends AbstractController
{
    /**
     * @Route("/admin/login", name="admin_account_login")
     */
    public function login(AuthenticationUtils $utils)
    {

        // fonction qui permet de récupérer la dernière erreur
        $error = $utils-> getLastAuthenticationError();


        return $this->render('admin/account/login.html.twig', [
            'hasError' => $error
        ]);
    }



    /**
     * @Route("/admin/logout", name="admin_account_logout")
     */
    public function logout(){

    }
}
```

Mais pour l'instant ,on ne peut pas aller sur admin/login pour se connecter car le dossier /admin n'est accessible qu'au role admin
On est donc obligé dans les access control de security.yaml de spécifier quelque-chose de plus spécifique.

```
access_control:
        - { path: ^/admin/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin, roles: ROLE_ADMIN }
```

IS_AUTHENTICATED_ANONYMOUSLY représente un utilisateur non connecté
https://symfony.com/doc/current/security.html

**b) Firewall particulier pour l'administration:**

Le problème maintenant est que si l'on va sur /admin/ads , on est redirigé sur /login et non sur /admin/login
Il faut dire à symfony, si on va sur une page admin redirige moi sur /admin/login
On va utiliser les **Firewall** (délimitent les emplacements du site et la façon de les sécuriser)

Avec le pattern, on va utiliser une expression régulière qui permettra de délimiter la partie à sécuriser.

https://symfony.com/doc/current/security/form_login.html

On va créer un firewall **admin** entre le dev et le main car il est plus spécifique que le main

```
        admin:
            pattern: ^/admin
            anonymous: true #on peut y accéder de manière anonyme à cause du access_control { path: ^/admin/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }

            provider: in_database

            form_login:
                login_path: admin_account_login
                check_path: admin_account_login

                #Redirection après succès de connexion
                default_target_path: admin_ads_index



            logout:
                path: admin_account_logout # nom de la route qui permet de se déconnecter ?

                # nom de la route où l'on doit aller après s'être déconnecté ?
                target: homepage
```

Et maintenant si on va sur /admin/ads , on est bien redirigé sur /admin/login

### 5) FORMULAIRE DE CONNEXION A L'ADMINISTRATION:

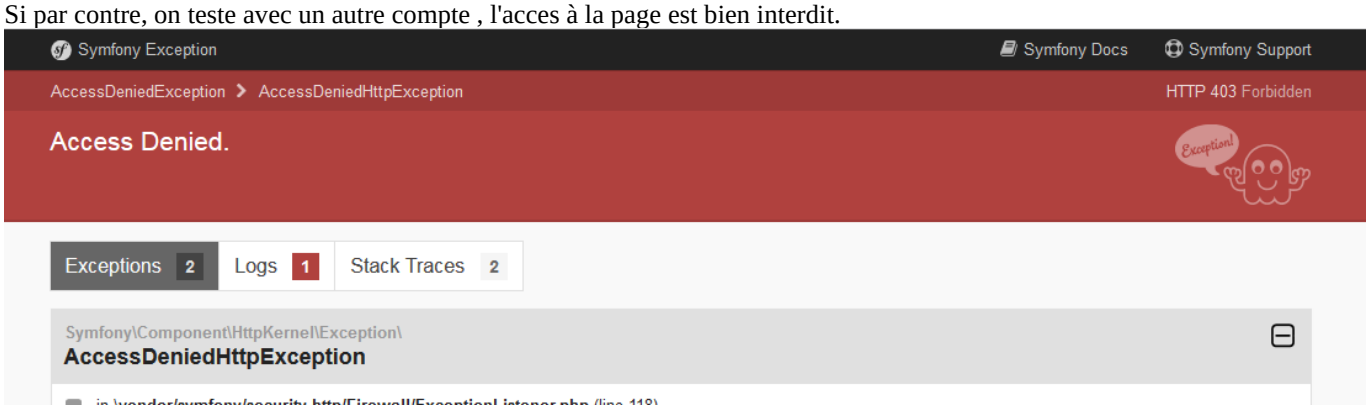
On va créer la page admin/account/login.html.twig

On créé une page classique html5

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous">

</head>
<body>

    <h1>Connexion à l'administration</h1>

    <div class="container">

        {% if (hasError) %}
            <div class="alert alert-danger alert-dismissible fade show" role="alert">
                Mauvais identifiant ou mot de passe
                <button type="button" class="close" data-dismiss="alert" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
        {% endif %}

        <form  method="post">

            <div class="form-group">
```

```
            <input placeholder="Adresse email" required type="text" name="_username" class="form-control">
        </div>

        <div class="form-group">
            <input placeholder="mot de passe" required type="password" name="_password" class="form-control">
        </div>

        <div class="form-group">
            <button type="submit" class="btn btn-success">Connexion !</button>
        </div>

    </form>
</div>


</body>
</html>
```

Si on va sur /admin/login et que l'on teste avec le compte administrateur , on est bien redirigé sur /admin/ads.
Si par contre, on teste avec un autre compte , l'acces à la page est bien interdit.




### 6) FORMULAIRE D'EDITION D'UNE ANNONCE:

Sur ce genre de site, on veut faire de la modération ( supprimer une annonce, modifier une annonce)
On a déjà un formulaire de modification ( Form/AnnonceType.php)
On va donc créér dans le controller **AdminAdController.php** une nouvelle fonction qui va exploiter AnnonceType.php

```
    /**
 *permet d'afficher le formulaire d'édition
     * @Route("/admin/ads/{id}/edit", name="admin_ads_edit")
     */
    public function edit(Request $request,EntityManagerInterface $manager,Ad $ad){


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



// debut Résolution du problème du bug du champ slug éventuellement vide lors du handleRequest (il extrait les données POST de la requête précédente, les traite et exécute toute validation (vérifie l'intégrité des données attendues par rapport aux données reçues))
//dump($request); // contenu de la request
//dump($request->request->get('annonce'));
//dump($request->request->get('annonce')['slug']);

//https://symfony.com/doc/current/components/http_foundation.html
        // Si le slug est vide , on le remplace par le titre formaté
            $slugify = new Slugify();
            $data = $request->request->get('annonce');
            //dump($data);
```

```
        if (empty($data['slug'])) $data['slug']=$slugify->slugify($request->request->get('annonce')['title']);
          //dump($data);
           //dump($data['slug']);
           $request->request->set('annonce',$data);
           //dump($data);
// fin Résolution du problème du bug du champ slug éventuellement vide



        $form ->handleRequest($request);


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


                //génération du slug automatiquement si pas soumis (voir fixture)
                  if (!$ad->getSlug()) {
                      $slugify = new Slugify();
                      $slug=$slugify->slugify($ad->getTitle());
                      $ad->setSlug($slug);
                      }

                  //dump($ad->getImages());exit;
                  foreach ($ad->getImages() as  $image) {
                      $image->setAd($ad);
                      $manager->persist($image);
                  }


                  $manager->persist($ad); // previent doctrine que l'on veut sauver
                  $manager->flush(); // envoi la requête à la base de donnée

              $this->addFlash(
                  'success',
                  'L\'annonce '.$ad->getTitle().' a bien été modifiée !'
              );

                  // on retourne sur la page de l'annonce
            //return $this->redirectToRoute('ads_show',['slug' => $ad->getSlug()]);


        }




    return $this->render('admin/ad/edit.html.twig', [
            'form' => $form -> createView(),
            'ad' => $ad,
        ]);
    }
```

On créé également le fichier **admin/ad/edit.html.twig**
On veut dans ce fichier pouvoir modifier l'annonce (colonne de gauche) mais aussi faire des rappels (voir le nombre de réservation, commentaires .. dans la colonne de droite)

Donc pour commencer on met le contenu du fichier **templates/ad/edit.html.twig** qui apparaitra dans la colonne de gauche.
On modifie le block _annonce_images_entry_row pour afficher les images et non les urls.

Rem: faire un **dump (form)** dans le bloc pour voir comment accéder à l'image et à la légende

On supprime le bouton ajouter une image

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

{% form_theme form with ['bootstrap_4_layout.html.twig', _self] %}


   {% block title %}Edition de l'annonce {% endblock %}

      {% block body %}

         <div class="container-fluid">


            <div class="row mt-3">
               <div class="col">
                  <div class="card card-body">
                     <h4 class="card-header">Corriger {{ ad.title }} ?</h4>

                     {{ form_start(form) }}

                     {{ form_row(form.title, {'label':'Titre','attr':{'placeholder':"Titre de l'annonce",'title':'Titre avec un minimun de 10 caractères et au maximum 50 caractères'}}) }}
                     {{ form_row(form.slug, {'label':'Chaine URL','attr':{'placeholder':"Adresse Web Automatique"}}) }}
                     {{ form_row(form.coverImage, {'label':"URL de l'image principale",'attr':{'placeholder':"Adresse d'une image"}}) }}
                     {{ form_row(form.introduction, {'label':'Introduction','attr':{'placeholder':"Description globale"}}) }}
                     {{ form_row(form.content, {'label':'Description détaillée','attr':{'placeholder':"Description détaillée"}}) }}
                     {{ form_row(form.rooms, {'label':'Nombre de chambres','attr':{'placeholder':"Contenu de l'article"}}) }}
                     {{ form_row(form.price, {'label':'Prix par nuit','attr':{'placeholder':"Indiquez le prix par nuit"}}) }}

                     {# champ annonce_images qui sera sur-chargé #}
                     {{ form_row(form.images) }}

                     <button type="submit" class="btn btn-success">Enregistrer les modifications</button>

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


               <div class="col">
                  <div class="card card-body ">

                     <h4 class="card-header">Les réservations</h4>

                     <table class="table">
                        <thead>
                           <tr class="text-center">
                              <th scope="col">Id</th>
                              <th scope="col">Voyageur</th>
                              <th scope="col">Date de réservation</th>
                              <th scope="col"></th>

                           </tr>
                        </thead>
                        <tbody>

                           {% for booking in ad.bookings %}
```

```
                    <tr class="text-center">

                        <td>{{booking.id}}</td>
                        <td>{{booking.booker.firstName}} {{booking.booker.lastName}}</td>
                        <td>{{booking.createdAt | date("d/m/Y")}}</td>
                        <td> <a href="#" class="btn btn-primary"><i class="fas fa-edit"></i></a></td>
                    </tr>
                {% endfor %}

            </tbody>
        </table>

        <h4 class="card-header mt-3">Les commentaires</h4>
        <table class="table">
            <thead>
                <tr class="text-center">
                    <th scope="col">Id</th>
                    <th scope="col">Voyageur</th>
                    <th scope="col">Note</th>
                    <th scope="col">Commentaire</th>
                    <th scope="col"></th>

                </tr>
            </thead>
            <tbody>

                {% for comment in ad.comments %}
                    <tr class="text-center">

                        <td>{{comment.id}}</td>
                        <td>{{comment.author.firstName}} {{comment.author.lastName}}</td>
                        <td>{{comment.rating}}</td>
                        <td>{{comment.content}}</td>
                        <td> <a href="#" class="btn btn-primary"><i class="fas fa-edit"></i></a></td>
                    </tr>
                {% endfor %}

            </tbody>
        </table>

    </div>
</div>

</div>

</div>
{% endblock %}

{# permet de modifier la sortie de form_row sur le champ annonce_images   #}
{# entry représente un élément de la collection ( ici sous-formulaire)  #}
{# Ici la variable form représente une entry  #}
{% block _annonce_images_entry_row %}
    {#{{ dump() }}#}
    {#{{dump(form)}}#}
    <div class="row mt-2" id="{{ id }}">
        <div class="col-3">
            <img src="{{form.children.url.vars.value}}" class="img-fluid" alt="{{form.children.caption.vars.value}}"/>
        </div>
        <div class="col">
            {{ form_row(form.caption, {'label':false,'attr':{'placeholder':"description de l'image"}}) }}
        </div>
        <div class="col-2">
            <button type="button"  class="btn btn-danger del_image" data-bloc="{{ id }}" >X</button>
        </div>
```

```
        </div>

    {% endblock %}

    {% block javascripts %}

        <script type="text/javascript">
    //variable globale qui donne le nombre de form images
    var counter = {{ form.images|length }};

    $('#add_image').click(function(){

        // nbrs de groupe de champs créés afin de connaitre l'indice à mettre dans le prototype
        counter = counter + 1;
        console.log(counter);

        // détermination du nombre de ligne
        //const index = $('#annonce_images .row').length;
        //console.log(index);

        // je récupère le prototype des entrées
        var tmpl = $('#annonce_images').data('prototype');
        //console.log(tmpl);

        // on remplace __name__ plusieurs fois
        tmpl = tmpl.replace(/__name__/g, counter);

        //On ajoute tmpl à la fin de la div annonce_images
        $('#annonce_images').append(tmpl);

        // on gère le bouton supprimer
        deleteBloc();

    });


    function deleteBloc(){

        $('.del_image').click(function(){
            var bloc = $(this).data('bloc');
            //console.log(bloc);

            $('#'+ bloc).remove();


        });

    }

    // on gère le bouton supprimer lors de l'arriver sur le formulaire (s'il y a déjà des images)
        deleteBloc();


        </script>

        {% endblock %}
```

On modifie également **admin/ad/index.html.twig** en rajoutant l'id sur le bouton edit

```
<a href="{{ path('admin_ads_edit', {'id':ad.id}) }}" class="btn btn-primary"><i class="fas fa-edit"></i></a>
```

### 7) PERMETTRE A L'ADMINISTRATEUR DE SUPPRIMER UNE ANNONCE

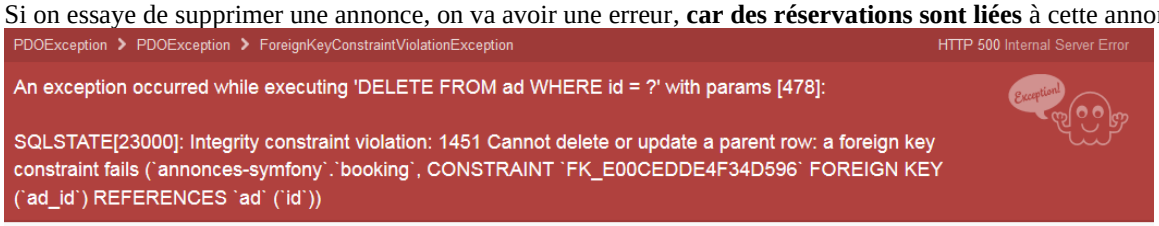
On créé une nouvelle fonction delete() dans AdminAdController.php

```
/**
*permet de supprimer une annonce
 * @Route("/admin/ads/{id}/delete", name="admin_ads_delete")
 */
  public function delete(EntityManagerInterface $manager,Ad $ad){


      $manager->remove($ad);
      $manager->flush(); // envoyer l'info à la bdd

 $this->addFlash(
                'success',
                'L\'annonce '.$ad->getTitle().' a bien été supprimée !'
            );
      return $this->redirectToroute('admin_ads_index');


   }
```

Et on modifie **admin/ad/index.html.twig**

```
<a href="{{ path('admin_ads_delete', {'id':ad.id}) }}" onclick="return confirm('Etes-vous certain de vouloir supprimer l\'annonce ?')" class="btn btn-danger"><i class="fas fa-trash-alt"></i></a>
```

Si on essaye de supprimer une annonce, on va avoir une erreur, **car des réservations sont liées** à cette annonce




On va donc empêcher de supprimer une annonce si des réservations sont en cours.
On modifie donc la fonction delete().

```
/**
*permet de supprimer une annonce
 * @Route("/admin/ads/{id}/delete", name="admin_ads_delete")
 */
  public function delete(ObjectManager $manager,Ad $ad){

    if (count($ad->getBookings())==0)
        {   $manager->remove($ad);
            $manager->flush(); // envoyer l'info à la bdd

          $this->addFlash(
                'success',
                'L\'annonce '.$ad->getTitle().' a bien été supprimée !'
            );
        }
    else {
            $this->addFlash(
                'warning',
                'L\'annonce '.$ad->getTitle().' ne peut pas être supprimée car des réservations sont en cours !'
            );

    }

    return $this->redirectToroute('admin_ads_index');
  }
```

### 8) AFFICHEZ LA LISTE DES COMMENTAIRES:

On créé un nouveau controller **AdminCommentController.php**

**php bin/console make:controller AdminCommentController**

```php
<?php

namespace App\Controller;

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

class AdminCommentController extends AbstractController
{
    /**
     * @Route("/admin/comments", name="admin_comment_index")
     */
    public function index(CommentRepository $repo)
    {

        $comments=$repo->findAll();

        return $this->render('admin/comment/index.html.twig', [
            'comments' => $comments,
        ]);
    }
}
```

Et le fichier **admin/comment/index.html.twig**

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

{% block title %}Administration des commentaires{% endblock %}

{% block body %}

    <div class="container-fluid">
        <h1>Gestion des commentaires</h1>

            <table class="table">
              <thead>
                <tr class="text-center">
                  <th scope="col">Id</th>
                  <th scope="col">Date</th>
                  <th scope="col">Auteur</th>
                  <th scope="col">Commentaire</th>
                  <th scope="col">Note</th>
                  <th scope="col">Annonce</th>
                  <th scope="col">Actions</th>
                </tr>
              </thead>
              <tbody>

                {% for comment in comments %}
                <tr class="text-center">
                  <td>{{comment.id}}</td>
                  <td>{{comment.createdAt | date('d/m/Y')}}</td>
                  <td><img src="{{comment.author.picture }}" alt="">{{comment.author.firstName}}
{{comment.author.lastName}}</td>
                  <td>{{comment.content}}</td>
                  <td>{{comment.rating|number_format(2, '.', ',')}}</td>
                  <td>{{comment.ad.title}}</td>
                  <td>
                     <a href="#" class="btn btn-primary"><i class="fas fa-edit"></i></a>
                    <a href="#" onclick="return confirm('Etes-vous certain de vouloir supprimer l\'annonce ?')" class="btn btn-
danger"><i class="fas fa-trash-alt"></i></a>

                  </td>
```

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

          </tbody>
        </table>

    </div>


{% endblock %}
```

### 9) FORMULAIRE D'EDITION D'UN COMMENTAIRE:

On pourrait utiliser le formulaire CommentType . Mais il est mieux d'en créer un nouveau pour bien séparer l'utilisateur de l'admin ou bien si on veut rajouter éventuellement des champs supplémentaires coté admin…

On créé AdminCommentType.php

**php bin/console make:form AdminCommentType**

Comment

Et on ne garde que add('content')

```
$builder
        //->add('createdAt')
        //->add('rating')
        ->add('content')
        //->add('ad')
        //->add('author')
```

On créé maintenant une nouvelle fonction **edit() dans AdminCommentController.php**

```
/**
* permet de modifier un commentaire
 * @Route("/admin/comments/{id}/edit", name="admin_comment_edit")
 */
public function edit(Comment $comment,Request $request,EntityManagerInterface $manager)
{

    $form=$this->createForm(AdminCommentType::class,$comment);


     $form ->handleRequest($request);


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

            $manager->persist($comment); // previent doctrine que l'on veut sauver
          $manager->flush(); // envoi la requête à la base de donnée

        $this->addFlash(
          'success',
          'Le commentaire d\'id '.$comment->getId().' a bien été modifiée !'
        );

        //
      return $this->redirectToRoute('admin_comment_index');

    }

    return $this->render('admin/comment/edit.html.twig', [
```

```
            'form' => $form->createView(),
            'comment'=>$comment,
        ]);
    }
```

Et on créé le fichier **admin/comment/edit.html.twig**

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

{% form_theme form with ['bootstrap_4_layout.html.twig'] %}


    {% block title %}Modification du commentaire n°: {{comment.id}} {% endblock %}

        {% block body %}

            <div class="container-fluid">


                <div class="row mt-3">
                    <div class="col">
                        <div class="card card-body">
                            <h4 class="card-header">Modifier le commentaire n°: {{comment.id}} ?</h4>


                            <div class="alert alert-info">
                            <p><strong>- Annonce lié:</strong> {{comment.ad.title  }}</p>
                            <p><strong>- Auteur:</strong> {{ comment.author.firstname }} {{ comment.author.lastName }}</p>
                            <p><strong>- Date de création:</strong> {{ comment.createdAt | date('d/m/Y') }}</p>
                            <p><strong>- Note donnée:</strong> {{ comment.rating }}</p>

                            </div>

                            {{ form_start(form) }}

                            {{ form_row(form.content, {'label':'contenu du Commentaire'}) }}

                            <button type="submit" class="btn btn-success">Enregistrer les modifications</button>

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


        {% endblock %}
```

Et on met à jour le bouton édition dans **admin/comment/index.html.twig**

```
<a href="{{ path('admin_comment_edit',{'id':comment.id})}}" class="btn btn-primary"><i class="fas fa-edit"></i></a>
```

**10) <u>PERMETTRE A L'ADMINISTRATEUR DE SUPPRIMER UN COMMENTAIRE:</u>**

On créé une nouvelle fonction delete()  dans AdminCommentController.php

```
    /**
```

```
     * permet de supprimer un commentaire
     * @Route("/admin/comments/{id}/delete", name="admin_comment_delete")
     */
    public function delete(Comment $comment,EntityManagerInterface $manager)
    {

                    $manager->remove($comment); // previent doctrine que l'on veut effacer
                $manager->flush(); // envoi la requête à la base de donnée

            $this->addFlash(
                'success',
                'Le commentaire a bien été supprimé !'
            );

            //
        return $this->redirectToRoute('admin_comment_index');

    }
```

Et on met à jour le bouton de suppression dans **admin/comment/index.html.twig**

```
<a href="{{ path('admin_comment_delete',{'id':comment.id})}}" onclick="return confirm('Etes-vous certain de vouloir supprimer le commentaire ?')" class="btn btn-danger"><i class="fas fa-trash-alt"></i></a>
```

On met également à jour le menu de navigation

```
    <li class="nav-item ">
            <a class="nav-link" href="{{ path('admin_comment_index') }}">Commentaires</a>
        </li>
```

**11) <u>AFFICHEZ LA LISTE DES RESERVATIONS:</u>**

On créé un nouveau controller **AdminBookingController**

**php bin/console make:controller AdminBookingController**

et on modifie la fonction index

```
    /**
     * @Route("/admin/bookings", name="admin_booking_index")
     */
    public function index(BookingRepository $repo)
    {


        return $this->render('admin/booking/index.html.twig', [
            'bookings' => $repo -> findAll(),
        ]);
    }
```

Et on met a jour le fichier **admin/booking/index.html.twig**

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

{% block title %}Administration des réservations{% endblock %}

{% block body %}

    <div class="container-fluid">
        <h1>Gestion des réservations</h1>

            <table class="table">
              <thead>
                <tr class="text-center">
```

```
            <th scope="col">Id</th>
            <th scope="col">Date</th>
            <th scope="col">Visiteur</th>
            <th scope="col">Annonce</th>
            <th scope="col">Durée</th>
            <th scope="col">Montant</th>
            <th scope="col">Actions</th>
          </tr>
        </thead>
        <tbody>

          {% for booking in bookings %}

            {% set difference= date(booking.endDate).diff(date(booking.startDate)) %}

          <tr class="text-center">
            <td>{{booking.id}}</td>
            <td>{{booking.createdAt|date('d/m/Y')}}</td>
            <td><img src="{{booking.booker.picture }}" alt="">{{booking.booker.firstName}}
{{booking.booker.lastName}}</td>
            <td>{{booking.ad.title}}</td>
            <td>{{difference.days}}</td>
            <td>{{booking.amount}} €</td>
            <td>
              <a href="#" class="btn btn-primary"><i class="fas fa-edit"></i></a>
             <a href="#" onclick="return confirm('Etes-vous certain de vouloir supprimer la réservation ?')" class="btn btn-
danger"><i class="fas fa-trash-alt"></i></a>

            </td>
          </tr>
          {% endfor %}

        </tbody>
      </table>

  </div>

{% endblock %}
```

**12) <u>FORMULAIRE DE GESTION DES RESERVATIONS:</u>**

On créé un nouveau formulaire

**php bin/console make:form AdminBookingType**

Booking

On supprime la date de création et le montant que l'on calculera automatiquement

```
    $builder
        ->add('startDate')
        ->add('endDate')
        //->add('createdAt')
        //->add('amount')
        ->add('comment')
        ->add('booker')
        ->add('ad')
    ;
  }
```

On créé une nouvelle fonction **edit() dans AdminBookingController.php**

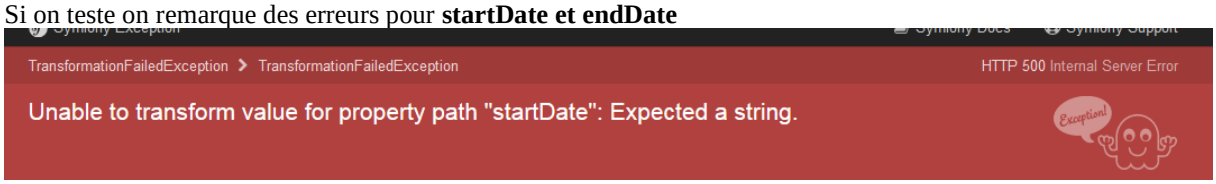
```
 /**
```

```
    *Permet d'éditer une réservation
     * @Route("/admin/bookings/{id}/edit", name="admin_booking_edit")
     */
    public function edit(Booking $booking)
    {

        $form=$this->createForm(AdminBookingType::class,$booking);


        return $this->render('admin/booking/edit.html.twig', [
            'form' => $form -> createView(),
            'booking' => $booking
        ]);
    }
```

Et le fichier **admin/booking/edit.html.twig**

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

{% form_theme form with ['bootstrap_4_layout.html.twig'] %}


    {% block title %}Modifier la réservations N°:{{booking.id}} {% endblock %}

        {% block body %}

            <div class="container-fluid">


                <div class="row mt-3">
                    <div class="col">
                        <div class="card card-body">
                            <h4 class="card-header">Modifier la réservations N°:{{booking.id}}</h4>

                            {{ form_start(form) }}

                            {{ form_widget(form)}}

                                    <button type="submit" class="btn btn-success">Enregistrer les modifications</button>
                            {{ form_end(form) }}
                        </div>
                    </div>
                </div>
            </div>


        {% endblock %}
```

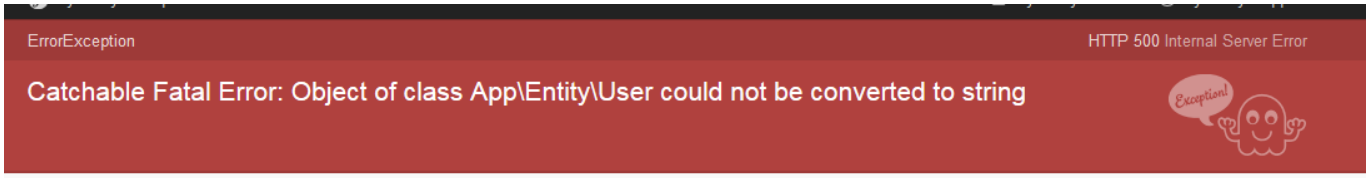
Si on teste on remarque des erreurs pour **startDate et endDate**




On donne donc le type DateType avec éventuellement l'option widget au niveau de **AdminBookingType**

```
        ->add('startDate',DateType::class,['widget'=>'single_text'])
         ->add('endDate',DateType::class,['widget'=>'single_text'])
```

On a ensuite une erreur à cause ->add('booker') et ->add('ad') qui sont des entités




https://symfony.com/doc/current/reference/forms/types.html

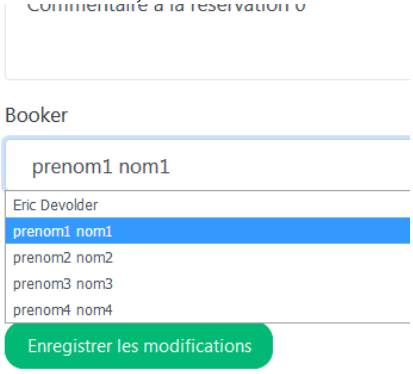
Choice Fields ¶

- ChoiceType
- EntityType

On va choisir EntityType

```
$builder
        ->add('startDate',DateType::class,['widget'=>'single_text'])
        ->add('endDate',DateType::class,['widget'=>'single_text'])
        //->add('createdAt')
        //->add('amount')
        ->add('comment')
       ->add('booker',EntityType::class,[
              'class'=>User::class,
              'choice_label' => function($user)
                 {return $user->getFirstName().' '.$user->getLastName();},
        ])
        ->add('ad',EntityType::class,[
           'class'=> Ad::class,
           'choice_label' => 'title',
        ])
    ;
```

Si maintenant, on teste:




On peut maintenant modifier la fonction **edit()** pour recalculer le nouveau montant de la réservation si modification.

```
/**
*Permet d'éditer une réservation
 * @Route("/admin/bookings/{id}/edit", name="admin_booking_edit")
 */
public function edit(Booking $booking,Request $request,EntityManagerInterface $manager)
{

    $form=$this->createForm(AdminBookingType::class,$booking);


   $form ->handleRequest($request); // analyse la request

    // si le form est soumis et valide (par rapport aux règles en place)
    if ($form->isSubmitted() && $form->isValid() ){

            // intervalle de date de réservation
```

```
            $diff = date_diff($booking->getStartDate(),$booking->getEndDate());
            //dump($diff);
            //dump($booking->getAd()->getPrice());
            // calcul nouveau prix
            $booking->setAmount($booking->getAd()->getPrice()*$diff->days);


            //sinon enregistrement et redirection

            $manager->persist($booking);
            $manager->flush();

                $this->addFlash(
                  'success',
                  "La réservation N°: {$booking->getId()} a été modifié"
                );

            return $this->redirectToroute('admin_booking_index');


        }

    return $this->render('admin/booking/edit.html.twig', [
        'form' => $form -> createView(),
        'booking' => $booking
    ]);
}
```

**Remarque:**
On peut également remettre en fonction datepicker (**pas une obligation**) pour les champs dates
Il faudra donc modifier **AdminBookingType.php**

```
private $transformer;


public function __construct(FrenchToDateTimeTransformer $transformer){

    $this->transformer = $transformer;

}


public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('startDate',TextType::class)
        ->add('endDate',TextType::class)
        //->add('createdAt')
        //->add('amount')
        ->add('comment')
       ->add('booker',EntityType::class,[
              'class'=>User::class,
              'choice_label' => function($user)
                 {return $user->getFirstName().' '.$user->getLastName();},
        ])
        ->add('ad',EntityType::class,[
            'class'=> Ad::class,
            'choice_label' => 'title',
        ])
```

```
        ;

$builder->get('startDate')->addModelTransformer($this->transformer);
$builder->get('endDate')->addModelTransformer($this->transformer);
```

Et **AdminBookingController.php**

```
    /**
    *Permet d'éditer une réservation
     * @Route("/admin/bookings/{id}/edit", name="admin_booking_edit")
     */
    public function edit(Booking $booking,Request $request,EntityManagerInterface $manager,BookingRepository $repo)
    {


 //***************** début liste dates non disponibles pour la réservation **********************************
            // initialisation du tableau contenant les dates non dispo
              $notAvailableDays=[];

              // il faut connaitre les dates qui sont impossibles pour l'annonce
              // on recupère toutes les réservations déjà existantes de l'annonce
              $repo = $repo->findBy(array('ad'=>$booking->getAd()->getId()));
              //dump($repo);

              // on boucle sur toutes les réservations déjà faites
              foreach ($repo as $item) {

                //on crée un tableau de timestamp en utilisant la fonction range() ( intervalle entre deux éléments avec un certain
step)
              $tous_les_timestamp=range($item->getStartDate()->getTimestamp(),$item->getEndDate()-
>getTimestamp(),24*60*60);

              // on fusionne tous les tableaux range
                $notAvailableDays=array_merge($notAvailableDays,$tous_les_timestamp);
              }

              //supprimer doublons
              $notAvailableDays=array_unique($notAvailableDays);

              // réinitialiser les clés
              $notAvailableDays=array_values($notAvailableDays);
              //dump($notAvailableDays);


      //***************** fin liste dates non disponibles pour la réservation **********************************
……….
```

Il faudrait bien sûr rajouter le test sur les dates dispos ou non.

**Et /admin/booking/edit.html.twig**

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

{% form_theme form with ['bootstrap_4_layout.html.twig'] %}


    {% block title %}Modifier la réservations N°:{{booking.id}} {% endblock %}


{% block stylesheets %}
```

```
<link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/css/bootstrap-datepicker.min.css">

{% endblock %}

    {% block body %}

        <div class="container-fluid">


            <div class="row mt-3">
                <div class="col">
                    <div class="card card-body">
                        <h4 class="card-header">Modifier la réservations N°:{{booking.id}}</h4>

                        {{ form_start(form) }}

                            <div class="row">
                                <div class="col">
                                    {{ form_row(form.startDate, {'label':'Date de début','attr':{'placeholder':"date de début"}}) }}
                                </div>
                                <div class="col">
                                    {{ form_row(form.endDate, {'label':'Date de fin','attr':{'placeholder':"Date de fin"}}) }}
                                </div>
                            </div>
                                <div class="row">
                                    <div class="col">
                                        {{ form_row(form.comment, {'label':'Commentaire','attr':{'placeholder':"Ecrire un commentaire"}}) }}
                                    </div>
                                </div>

                                <div class="row">
                                    <div class="col">
                                        {{ form_row(form.booker,{'label':'Visiteur'}) }}
                                    </div>
                                </div>

                                <div class="row">
                                    <div class="col">
                                        {{ form_row(form.ad,{'label':'Annonce'}) }}
                                    </div>
                                </div>

                                <button type="submit" class="btn btn-success">Enregistrer les modifications</button>
                        {{ form_end(form) }}
                    </div>
                </div>
            </div>
        </div>


    {% endblock %}



{% block javascripts %}
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/js/bootstrap-datepicker.min.js"></script>

<script>

  $(document).ready(function(){
```

```
// on détermine #admin_booking_startDate et #admin_booking_endDate en explorant le code des inputs
    $('#admin_booking_startDate , #admin_booking_endDate').datepicker({
                format: 'dd/mm/yyyy',
                  datesDisabled: [
                          {% for day in notAvailableDays %}
                              "{{ day | date('d/m/Y') }}",
                          {% endfor %}

                  ],
                  startDate: new Date()
                      });

  });




</script>

{% endblock %}
```

### 13) NOTION DE GROUPES DE VALIDATION:

Pour une même entité, on peut avoir des validations différentes. ( groupe de validations)
Exemple: validations différentes pour les formulaires frontend et backend
https://symfony.com/doc/current/validation/groups.html
On peut mettre les validations dans des groupes ( je veux appeler la validation avec tel groupe …)
Il existe deux groupes par défauts (voir doc):

- Groupe **Default** (toutes les contraintes qui n'ont aucun groupe identifié )
- Groupe User (nom de la classs )
- Groupe registration

Exemple: (validation valable que pour le groupe front )

```
    * @Assert\GreaterThan("today", message="la date d'arrivée doit être ultérieur à aujourd'hui !" ,groups={"front"})
    */
   private $startDate;
```

A la création du formulaire, on pourra préciser les groupes de validation que l'on voudra exécuter avec l'option validation_groups . https://symfony.com/doc/current/form/validation_groups.html

```
$form=$this->createForm(AdminBookingType::class,$booking,[
        'validation_groups' => ["Default","front"],
      ]);
```

### 14) PERMETTRE A L'ADMINISTRATEUR DE SUPPRIMER UNE RESERVATION:

On créé une fonction **delete()** dans **AdminBookingController.php**

```
    /**
   *Permet de supprimer une réservation
    * @Route("/admin/bookings/{id}/delete", name="admin_booking_delete")
    */
   public function delete(Booking $booking, EntityManagerInterface $manager)
   {

           $manager->remove($booking);
           $manager->flush();

           $this->addFlash(
                       'success',
                       'La réservation a bien été supprimée !'
                   );

           return $this->redirectToroute('admin_booking_index');
}
```

On modifie les liens action dans la gestion des réservations: (index.html.twig)

```
                <a href="{{path('admin_booking_edit',{'id':booking.id})}}" class="btn btn-primary"><i class="fas fa-edit"></i></a>
                <a href="{{path('admin_booking_delete',{'id':booking.id})}}" onclick="return confirm('Etes-vous certain de vouloir supprimer la réservation ?')" class="btn btn-danger"><i class="fas fa-trash-alt"></i></a>
```

Et le lien Réservation de la barre de navigation

```
    <li class="nav-item ">
          <a class="nav-link" href="{{ path('admin_booking_index') }}">Réservations</a>
       </li>
```

## XXXIII) METHODES DES REPOSITORIES POUR RECUPERER LES DONNEES:

### 1) LES REPOSITORIES ( ENTREPOTS DE DONNEES):

Rappels: Ils nous permettent de faire des sélections avec plusieurs méthodes.
https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/working-with-objects.html#querying

- Méthode **find:** permet de retrouver un enregistrement par son identifiant (find($id) )
  Ex: **$ad = $repo->find(332);**

- Méthode **findBy():** permet de retrouver plusieurs enregistrements grâce à des critères ( retourne une liste d'entités). Elle peut prendre 4 arguments (Critères, Orders, Limit, Offset (début))
  Ex:
  **$ad = $repo->findBy([], [], 5, 0);** (5 annonces à partir du début)

  ```
  $ad = $repo->findBy(
          array('author' => '44'), // Critere
          array('rooms' => 'desc'),       // Tri
          2,                      // Limite = nombre d'enregistrements que l'on remonte
          0                       // Offset = à partir de où on part
          );
  ```

- Méthode **findOneBy():** permet de trouver un enregistrement grâce à des critères de recherche sous forme d'un tableau. (ne retourne qu'une seule entité)
  Ex: 
  ```
  $ad = $repo->findOneBy([
          'id' => 332,
          'title' => "test annonce"
  ]);
  ```

On pourra faire des requêtes plus complexes avec le langage **DQL: Doctrine Query Language**

### 2) UTILISATION DU FINDBY() POUR PAGINER:

#### a) Utilisation du findby() pour paginer:

On retourne sur le thème choisi sur https://bootswatch.com/ et on récupère ce qui concerne la pagination et on l'installe dans le template twig des annonces. (admin/ad/index.html.twig)
https://getbootstrap.com/docs/4.0/utilities/flex/ => centrer div pagination

```
……
        </tbody>
    </table>



    <div class="d-flex justify-content-center">
      <ul class="pagination">
        <li class="page-item disabled">
          <a class="page-link" href="#">&laquo;</a>
        </li>
        <li class="page-item active">
          <a class="page-link" href="#">1</a>
        </li>
        <li class="page-item">
          <a class="page-link" href="#">2</a>
        </li>
        <li class="page-item">
          <a class="page-link" href="#">3</a>
        </li>
        <li class="page-item">
          <a class="page-link" href="#">4</a>
        </li>
        <li class="page-item">
          <a class="page-link" href="#">5</a>
        </li>
        <li class="page-item">
          <a class="page-link" href="#">&raquo;</a>
```

```
            </li>
        </ul>
    </div>


</div>

{% endblock %}
```

On veut afficher que **10** enregistrements par page
On va changer la route en rajoutant un paramètre {page} et on met une page par défaut  $page = 1

```
/**
 * @Route("/admin/ads/{page}", name="admin_ads_index")
 */
public function index(AdRepository $repo, $page= 1)
{

  $limit=10; // on veut dix enregistrements

  $start = $page * $limit - $limit;
  // 1 * 10 -10 = 0
  // 2 * 10 -10 = 10

    return $this->render('admin/ad/index.html.twig', [
      'ads' => $repo -> findBy([],[],$limit,$start),
    ]);
}
```

Attention cependant la variable page n'est pas contrainte. En effet "un petit malin" pourrait mettre http://127.0.0.1:8000/admin/ads/bonjour et la tout "plante".


On peut mettre des contraintes sur la route: https://symfony.com/doc/current/routing.html#adding-wildcard-requirements

```
/**
 * @Route("/admin/ads/{page}", name="admin_ads_index", requirements={"page":"[0-9]{1,}"})
 */
```

Regex: [0-9]{1,} => un chiffre une fois ou plus

### b) Rendu HTML de la pagination:

Il nous faut agir maintenant sur le rendu HTML de façon dynamique.

- Il faut donc déjà connaitre le nombre d'enregistrements au total: $total=count($repo->findAll());
- Le nombre de pages total (arrondi à l'entier supérieur):  $pages= ceil($total/$limit);

On va passer à twig les variables $pages et $page

```
return $this->render('admin/ad/index.html.twig', [
        'ads' => $repo -> findBy([],[],$limit,$start),
        'pages' => $pages,
        'page' => $page
    ]);
```

Et dans twig , on modifie le code pagination

```
<div class="d-flex justify-content-center">
        <ul class="pagination">
          <li class="page-item {% if page==1 %}disabled{% endif %}">
            <a class="page-link" href="{{path('admin_ads_index',{'page':page-1})}}">&laquo;</a>
          </li>

        {% for i in 1..pages %}
          <li class="page-item {% if page==i %}active{% endif %}">
            <a class="page-link" href="{{path('admin_ads_index',{'page':i})}}">{{i}}</a>
          </li>
        {% endfor %}

          <li class="page-item {% if page==pages %}disabled{% endif %}">
            <a class="page-link" href="{{path('admin_ads_index',{'page':page+1})}}">&raquo;</a>
          </li>
        </ul>
      </div>
```

**c) <u>Notion de Service pour la pagination ( code réutilisable):</u>**

On a un souci. Si l'on veut mettre des paginations partout. Il va falloir dupliquer du code (dans les controllers et templates) . C'est une mauvaise pratique.
On introduit donc la notion de service ( extraire et mettre en commun une logique particulière)
Un service permet de centraliser du code.
On créé un dossier **src/Service** et un fichier **PaginationService.php**
Ex de test:

```
<?php

namespace App\Service;


class PaginationService{

        private $testService='service de pagination';

          public function getTestService(){

            return $this->testService;
                }
}

?>
```

Et maintenant dans un controller, on peut y avoir acces
Ex dans **AdminAdController.php**

```
……..
public function index(AdRepository $repo, $page= 1, PaginationService $pagination)
  {

    dump($pagination ->getTestService());
    die();

……..
```

```
AdminAdController.php on line 23:
"service de pagination"
```

Pour **PaginationService.php**
On créé une variable $limit avec son setter et getter

```
  private $limit=10; // nbrs d'enregistrements


     public function SetLimit($limit){

          $this->limit=$limit;
         return $this;

     }


     public function getLimit(){

     return $this->limit;

     }
```

Il faut maintenant une fonction qui va aller chercher les données des pages. Elle devra faire appel au repository.

```
     public function getData($page,$repo){
         //1) calculer l'offset ( start)
         $start = $page * $this->limit - $this->limit;

         //2) demander au repository de trouver les éléments
         $data=$repo -> findBy([],[],$this->limit,$start);

         //3) renvoyer les éléments en question
         return $data;

     }
```

Il faut aussi une fonction pour le nombre de pages.

```
     public function getPages($repo){

         $total=count($repo->findAll()); // nbrs d'enregistrement au total

          $pages= ceil($total/$this->limit); // nbrs de page total arrondi à l'entier supérieur

          return $pages;

     }
```

**AdminAdController.php** devient maintenant:

```
  /**
   * @Route("/admin/ads/{page}", name="admin_ads_index", requirements={"page":"[0-9]{1,}"})
   */
  public function index(AdRepository $repo, $page= 1, PaginationService $pagination)
  {

      return $this->render('admin/ad/index.html.twig', [
         // 'ads' => $repo -> findBy([],[],$limit,$start),
          'ads' => $pagination->getData($page,$repo),
          //'pages' => $pages,
          'pages' => $pagination->getPages($repo),
          'page' => $page
      ]);
  }
```

Il est judicieux également de faire des includes concernant la pagination twig

## XXXIV) RECUPERER SES ENTITES AVEC DOCTRINE2

### 1) MISE EN PLACE D'UN DASHBOARD:

On créé un nouveau controller:

**php bin/console make:controller AdminDashboardController**

```
<?php

namespace App\Controller;

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

class AdminDashboardController extends AbstractController
{
    /**
     * @Route("/admin", name="admin_dashboard")
     */
    public function index()
    {
        return $this->render('admin/dashboard/index.html.twig');
    }
}
```

Et **index.html.twig**

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

{% block title %}Tableau de bord{% endblock %}

    {% block body %}

        <div class="container-fluid">
            <h1>Tableau de bord</h1>
            <div class="row">
                <div class="col">
                    <div class="card card-body bg-primary text-white text-center">
                        100 utilisateurs

                    </div>
                </div>
                <div class="col">
                    <div class="card card-body bg-warning text-white text-center">
                        50  Annonces

                    </div>
                </div>
                <div class="col">
                    <div class="card card-body bg-success text-white text-center">
                        100 réservations

                    </div>
                </div>
                <div class="col">
                    <div class="card card-body bg-primary text-white text-center">
                        250 Avis

                    </div>
                </div>

            </div>

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

## 2) LE DOCTRINE QUERY LANGUAGE (DQL):

Depuis un repository, il existe deux moyens de récupérer les entités : en utilisant du **DQL** et en utilisant le **QueryBuilder.**

Le **DQL** n'est rien d'autre que du SQL adapté à la vision par objets que Doctrine utilise. Il s'agit donc de faire ce qu'on a l'habitude de faire, des requêtes textuelles comme celle-ci par exemple :

**SELECT a FROM App\Entity\Ad a**

Pour tester rapidement vos requêtes DQL:
**php bin/console doctrine:query:dql "select a from App\Entity\Ad a"**

https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/dql-doctrine-query-language.html
Tout d'abord, vous voyez que l'on n'utilise pas de table. On a dit qu'on pensait objet et non plus base de données ! Il faut donc utiliser dans les FROM et les JOIN le nom des entités.
Le namespace complet de l'entité. De plus, il faut toujours donner un alias à l'entité, ici on a mis **« a ».** On met souvent la première lettre de l'entité, même si ce n'est absolument pas obligatoire.
On sélectionne simplement l'alias, ici **« a »,** ce qui sélectionne en fait tous les attributs d'une annonce. L'équivalent d'une étoile (*) en SQL donc.

Il est tout de même possible de ne sélectionner qu'une partie d'un objet, en faisant **« a.title** » par exemple. Mais vous ne recevez alors qu'un tableau contenant les attributs sélectionnés, et non un objet. Vous ne pouvez donc pas modifier/supprimer/etc. l'objet, puisque c'est un tableau.

**SELECT a.title FROM App\Entity\Ad a**

Pour créer une requête en utilisant du DQL, il faut utiliser la méthode **createQuery()** de l'EntityManager :
Dans un controller , on peut avoir accès à un $manager . Et le manager de doctrine permet de traiter des requêtes DQL

```
/**
 * @Route("/admin", name="admin_dashboard")
 */
public function index(EntityManagerInterface $manager)
{

    //$users= $manager->createQuery("SELECT u FROM App\Entity\User u");
    //$users = $users->getResult();

$users= $manager->createQuery("SELECT COUNT(u) FROM App\Entity\User u")->getSingleScalarResult();
//dump($users);
$ads= $manager->createQuery("SELECT COUNT(a) FROM App\Entity\Ad a")->getSingleScalarResult();
$bookings= $manager->createQuery("SELECT COUNT(b) FROM App\Entity\Booking b")->getSingleScalarResult();
$comments= $manager->createQuery("SELECT COUNT(c) FROM App\Entity\Comment c")->getSingleScalarResult();

    return $this->render('admin/dashboard/index.html.twig',[
        'users'=>$users,
        'ads'=>$ads,
        'bookings'=>$bookings,
        'comments'=>$comments

    ]);
}
```

**getResult():** Exécute la requête et retourne un tableau contenant les résultats sous forme d'objets. Vous récupérez ainsi une liste des objets, sur lequels vous pouvez faire des opérations, des modifications, etc.

**getScalarResult():** Exécute la requête et retourne un tableau contenant les résultats sous forme de valeurs. dans ce tableau, un résultat est une valeur, non un tableau de valeurs (getArrayResult) ou un objet de valeurs (getResult). Cette méthode est donc utilisée lorsque vous ne sélectionnez qu'une seule valeur dans la requête, par exemple :SELECT COUNT(*) FROM ...

**getSingleScalarResult():** Exécute la requête et retourne une seule valeur. Cette méthode est très utilisée également pour des requêtes du type SELECT COUNT(*) FROM , qui ne retournent qu'une seule ligne de résultat, et une seule valeur dans cette ligne.

On veut maintenant récupérer les annonces qui ont été le mieux notés et celles qui ont été le moins bien notés.

```
$bestAds= $manager->createQuery(
        'SELECT AVG(c.rating) as note, a.title, a.id, u.firstName, u.lastName, u.picture
        FROM App\Entity\Comment c
        JOIN c.ad a
        JOIN a.author u
        GROUP BY a
        ORDER BY note DESC'
)->setMaxresults(5)
->getResult();

$worstAds= $manager->createQuery(
        'SELECT AVG(c.rating) as note, a.title, a.id, u.firstName, u.lastName, u.picture
        FROM App\Entity\Comment c
        JOIN c.ad a
        JOIN a.author u
        GROUP BY a
        ORDER BY note ASC'
)->setMaxresults(5)
->getResult();
```

Plutôt que de faire tout cela dans le controller. Il serait plus judicieux de créer un service qui servirait à faire des statistiques.

Et on modifie bien sur index.html.twig

### 3) LE QUERYBUILDER (utilitaire pour créer des requêtes DQL):

Le QueryBuilder est un moyen plus nouveau. Comme son nom l'indique, il sert à construire une requête, par étape.
Un des avantages est qu'il est possible de construire la requête en plusieurs fois.
le QueryBuilder permet de construire une Query, mais il n'est pas une Query !

On va se servir des **repositories.**
Lorsqu'on récupère un QueryBuilder depuis un repository, c'est que l'on veut faire une requête sur l'entité gérée par ce repository.
Donc si l'on pouvait définir la partie **SELECT a.title FROM App\Entity\Ad a** sans trop d'effort, cela serait bien pratique, car ce qui est intéressant, c'est le reste de la requête. Heureusement, le repository contient également une méthode **createQueryBuilder($alias)** qui utilise la méthode de l'EntityManager, mais en définissant pour nous le **SELECT et le FROM**
L'alias en argument de la méthode est le raccourci que l'on donne à l'entité du repository. On utilise souvent la première lettre du nom de l'entité, dans notre exemple de l'annonce cela serait donc un **« a ».**

Exemple:
Nous allons recréer la méthode **findAll()** dans notre repository **Ad**
**myFindAll()** retourne exactement le même résultat qu'un **findAll(),**
On va dans **AdRepository.php**

```
    public function myFindAll()
{

  // on fait SELECT a FROM App\Entity\Ad a avec la méthode raccourci
  $queryBuilder= $this->createQueryBuilder('a');

  // On récupère la Query à partir du QueryBuilder
  $query = $queryBuilder->getQuery();

  // On récupère les résultats à partir de la Query
  $results = $query->getResult();

  // On retourne ces résultats
  return $results;

}
```

Si maintenant dans AdController.php , on remplace **findAll** par **myFindAll** , on retrouve la même chose

Un des avantages avec le queryBuilder est qu'il est possible de construire la requête en plusieurs fois ( on rajoute des conditions)

Le QueryBuilder dispose de plusieurs méthodes afin de construire notre requête. Il y a une ou plusieurs méthodes par partie de requête : le WHERE, le ORDER BY, le FROM, etc.

- **where() :** On définit un paramètre dans la requête avec : **nom_du_parametre**
- **setParameter() :** attribue une valeur à ce paramètre avec **setParameter('nom_du_parametre', $valeur).**

Ex: on recrée la méthode **find($id)**

```
    public function myFindId($id)
{


  $queryBuilder= $this->createQueryBuilder('a')
                ->where('a.id = :id')
                ->setParameter('id',$id);

  // On récupère la Query à partir du QueryBuilder
  $query = $queryBuilder->getQuery();

  // On récupère les résultats à partir de la Query
  $results = $query->getResult();

  // On retourne ces résultats
  return $results;

 }
```

- **andWhere():**
- **orderBy()**
- **setMaxResults();** nbrs limite de résultats

**Ex: On veut récupérer toutes les annonces ayant un prix < 100 € et un nombre de chambres > 3 , ordonné par prix croissant**

```
    public function myFindAdPrice($price,$rooms)
{


  $queryBuilder= $this->createQueryBuilder('a')
                ->where('a.price <= :price')
                ->setParameter('price',$price)
                ->andWhere('a.rooms >= :rooms')
                ->setParameter('rooms',$rooms)
                ->orderBy('a.price','ASC');

  // On récupère la Query à partir du QueryBuilder
  $query = $queryBuilder->getQuery();

  // On récupère les résultats à partir de la Query
  $results = $query->getResult();

  // On retourne ces résultats
  return $results;

 }
```

On aurait pu mettre tout dans le même where() à la place du andWhere()

```
->where('a.price <= :price AND a.rooms >= :rooms')
->setParameters(array('price'=> $price,'rooms'=>$rooms))  => ATTENTION setParameters pas setParameter
```

**Pour faire une jointure :**
On veut récupérer les entités liées.
Comment récupérer tout d'un coup avec une jointure ?

- **leftJoin() :** récupérer toutes les données, même celles qui n'ont pas de correspondance.
- **join() :** équivalent d'un INNER JOIN . ex: **join('a.author', 'author') .** Le premier argument de la méthode est l'attribut de l'entité principale sur lequel faire la jointure. Le deuxième argument de la méthode est l'alias de l'entité jointe.
- **addSelect() : ex: addSelect('author').** On sélectionne également l'entité jointe.
- **groupBy() :** Il faut utiliser **groupBy** en même temps qu'une fonction d'agrégat ( SUM, AVG , COUNT … )
- **having():** filtrer les données regroupées. HAVING est un peu l'équivalent de WHERE, mais il agit sur les données une fois qu'elles ont été regroupées.

**Ex: On veut récupérer toutes les annonces écrites par un auteur (email)**

```
    public function myFindAdEmail($email)
{


  $queryBuilder= $this->createQueryBuilder('a')
                ->join('a.author','author')
                ->addSelect('author')
                ->where('author.email = :email')
                ->setParameter('email',$email);


  // On récupère la Query à partir du QueryBuilder
  $query = $queryBuilder->getQuery();

  // On récupère les résultats à partir de la Query
  $results = $query->getResult();

  // On retourne ces résultats
  return $results;

}
```

**Ex: On veut récupérer toutes les annonces écrites par un auteur (email) et une moyenne de note >3**

```
    public function myFindAdRating($email,$note)
{


  $queryBuilder= $this->createQueryBuilder('a')
                ->join('a.author','author')
                ->addSelect('author')
                ->where('author.email = :email')
                ->setParameter('email',$email)
                ->join('a.comments','comments')
                ->addSelect('AVG(comments.rating)')
                ->groupBy('a')
                ->having('AVG(comments.rating) > :note')
                ->setParameter('note',$note);


  // On récupère la Query à partir du QueryBuilder
  $query = $queryBuilder->getQuery();
```

```
    // On récupère les résultats à partir de la Query
    $results = $query->getResult();

    // On retourne ces résultats
    return $results;

}
```

## XXXV) HOMEPAGE DU SITE:

### 1) LES MEILLEURES ANNONCES:

On veut récupérer les annonces les mieux notées.

On va dans **AdRepository.php** et on crée la fonction **findBestAds()**

```
    public function findBestAds($limit)
    {

        $queryBuilder= $this->createQueryBuilder('a')
            ->join('a.comments','comments')
            ->addSelect('AVG(comments.rating) as avgRatings')
            ->groupBy('a')
            ->setMaxResults($limit)
            ->orderBy('avgRatings', 'DESC');


        // On récupère la Query à partir du QueryBuilder
        $query = $queryBuilder->getQuery();

        // On récupère les résultats à partir de la Query
        $results = $query->getResult();

        // On retourne ces résultats
        return $results;


    }
```

Et on modifie le HomeController

```
    /**
     * @Route("/", name="homepage")
     */
    public function index(AdRepository $repo)
    {

        $ads=$repo->findBestAds(3);
        //dump($ads);


        return $this->render('home/index.html.twig', [
            'ads'=>$ads
        ]);
    }
```

Ainsi que le fichier index.html.twig

….

```
            <h2>Nos Appartements stars</h2>

            <div class="row mt-3 mb-3">

{% for ad in ads %}

            {% include 'ad/_ad.html.twig' with {'ad':ad.0}  %}

{% endfor %}



            </div>
```

…..

**2) LES PROPRIETAIRES STARS:**

On veut récupérer les propriétaire ayant les annonces les mieux notées.

On va dans **UserRepository.php** et on crée la fonction **findBestUsers()**

```
    public function findBestUsers($limit=2)
{

        $queryBuilder= $this->createQueryBuilder('u')
            ->join('u.ads','ads')
            ->join('ads.comments','comments')
            ->addSelect('AVG(comments.rating) as avgRatings')
            ->groupBy('u')
            ->setMaxResults($limit)
            ->orderBy('avgRatings', 'DESC');

                // On récupère la Query à partir du QueryBuilder
  $query = $queryBuilder->getQuery();

  // On récupère les résultats à partir de la Query
  $results = $query->getResult();

  // On retourne ces résultats
  return $results;

  }
```

Et on modifie le **HomeController**

```
  /**
   * @Route("/", name="homepage")
   */
  public function index(AdRepository $repo,UserRepository $repo2)
  {

     $ads=$repo->findBestAds(3);
     //dump($ads);

      $users=$repo2->findBestUsers();
     //dump($users);

     return $this->render('home/index.html.twig', [
        'ads'=>$ads,
        'users'=>$users
     ]);
  }
```

Ainsi que le fichier **index.html.twig**

```
…….
<h2>Nos Propriétaires stars</h2>
        <div class="row mt-3 mb-4">

            {% for user in users %}
              <div class="col-6">
                <div class="card ">
                  <div class="card-header">
                    {{user.0.firstName}} {{user.0.lastName}}
                  </div>
                  <div class="card-body">
                    <div class="card-text">
                      <img src="{{user.0.picture}}" class="float-left mr-3">
                      <p>{{user.0.introduction}}</p>
                    </div>
                    <div class="text-right">
                      <a href="{{path('user_show',{'slug':user.0.slug})}}" class="btn btn-primary">En Savoir plus</a>
                    </div>
                  </div>
                </div>
              </div>
            {% endfor %}


        </div>
……….
```

**ANNEXE:**

## 1) VERSIONNING DE CODE AVEC GIT:

Cet outil permet de faire du versioning et de garder une trace des modifications apportées à un projet.


**Historique des modifications:** ensemble des modifications apportées à un projet. A tout moment on pourra revenir en arrière
**Résolution des conflits par la fusion:** travail à plusieurs ( si deux personnes travaillent en même temps sur un même fichier
**Possibilité d'avoir des branches:** Si on veut essayer de nouveaux tests et que l'on ne veut pas faire planter le projet, on va créer une sorte de timeline parallèle sur laquelle on pourra faire des essais . Si cela fonctionne , on pourra ramener les modifications sur le projet principal

### a) Premier Commit:

**git init** : initialise le versionning (création du dépôt GIT)

Pour voir les commandes de configuration**: git help config**

**git config –global user.email test@tes.fr** : permet de vous identifier lorsque vous faites du versionning (qui a travaillé au niveau de l'historique)
**git config –global user.name "eric" :** permet de vous identifier lorsque vous faites du versionning (qui a travaillé au niveau de l'historique)
**git config –list** : état de la configuration

**git status** : état du projet

Pour ignorer des fichier , on créé un fichier **.gitignore** avec à l'intérieur ce que l'on veut ignorer
*.tmp
tmp/*

**git log --oneline** : liste les derniers commit effectués dans le projet

**git diff** : prend tous les fichiers du dernier commit et montre les modifications

---
**Création du premier commit:**
**git add .** : on rajoute tout ce que l'on vient de faire à GIT
**git commit -m "Premier commit, installation"** : confirme les changements

### b) Revenir en arrière:

**git log --pretty=oneline** (voir tous les commits)

```
λ  git log --pretty=oneline
ada2500cd5861c9c9fa433439c7b3c1ca546891a (HEAD -> master) suppression contenu index
7636c0f629730d9830a7258b166babc01f81941a commit 2
731d2f9c65e0e99561b977a4c9f646d40e3c2f5d premier commit

D:\GRETA\laragon\www\projet (master)
λ
```

**git log --pretty=oneline -p index.html** ( voir les modifs de index.html)

**Revenir au commit 2**

**git checkout 7636c0f629730d9830a7258b166babc01f81941a** ( voir l'état du dépôt au moment du commit 2)

Mais on n'a pas perdu pour autant toutes les autres modifications. En fait on revient juste dans le temps comme un spectateur, on regarde comment c'était avant.

On ne pourra en aucun cas sauvegarder les modifications (versionning) à partir de cette "position".

Si on fait un commit à ce moment-là (dans le passé), il ne sera bien sauvegardé, mais il ne sera nulle part. (il ne sera relié à rien)

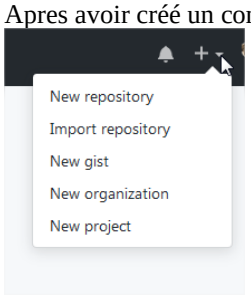
Si on tape **git checkout 7636c0f629730d9830a7258b166babc01f81941a index.html ,** cela prendra la fichier index tel qu'il était à ce moment-là.

Revenir sur le master : **git checkout master**

**git reset 7636c0f629730d9830a7258b166babc01f81941a --hard** (revient au commit 2) en effaçant tous les suivants.

### c) Compte GitHub (permet de mettre en ligne le code versionner):

Apres avoir créé un compte, on créé un nouveau dépôt (new repository)




Une fois les données renseignées, on aura la procédure :

**git remote add origin https://github.com/seabird2fr/formation.git**

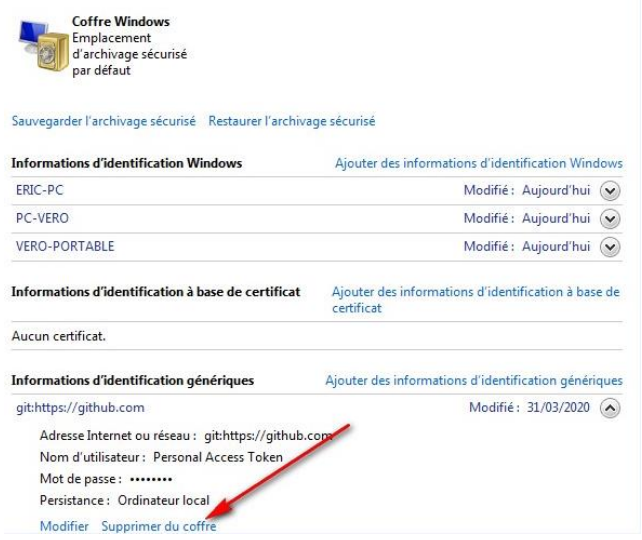
**git push -u origin master**

ce qui transférera tout sur la plate-forme github

**Rem:**

Si il y a une erreur de ce type par exemple: remote: *Permission to seabird2fr/annoncegreta.git denied*

Aller dans Aller dans le *gestionnaire d'identification* de windows



**Remarque:**
Si on regarde sur Github la liste des fichiers, on voit que le dossier **vendor** (contient toutes les dépendances que composer a installé) est absent. Il n'y a pas non plus le fichier .env ( le fichier est personnel)

**Ajouter un proxy pour git:**
git config --global http.proxy http://10.40.170.12:8181 (par exemple)

**Supprimer un proxy pour git:**
git config --global --unset http.proxy

## 2) HEBERGEMENT MISE EN LIGNE AVEC ACCES SSH:

Tout d'abord, il faut installer **composer** dans notre dossier de travail (on a besoin de composer pour installer notre application)
https://getcomposer.org/download/

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'48e3236262b34d30969dca3c37281b3b4bbe3221bda826ac6a9a62d6444cdb0dcd0615698a5cbe587c3f0fe57a54d8f5') { echo
'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Une fois l'installation effectuée, si on fait **ls**, on voit un fichier **composer.phar**
Pour le lancer au lieu de taper **composer**, on tapera:
$ **php composer.phar**

Ensuite, on va sur le dépôt git:
On clique sur **Clone ou download** et on copie le lien
**$ git clone https://github.com/seabird2fr/formation.git**
(si on a pas la commande GIT: **apt-get install git** )

Par ex, on pourrait avoir un structure telle que la suivante:
composer.phar **formation www**

On rentre dans la dossier **formation** pour installer toutes les dépendances grâce à composer
**…/formation $ php ../composer.phar update**

Une fois ceci fait , on modifie le fichier **.env** (base de donnée)
Si la base existe déjà, on lance les migrations: **$ php bin/console doctrine:migrations:migrate**
Et éventuellement les fixtures: **$ php bin/console doctrine:fixtures:load**

Et enfin il faut faire pointer le nom de domaine vers le dossier **formation/public**
Ou alors on peut créer un lien symbolique (**lien entre www et formation/public** )
**ln -s formation/public www** (il faut bien sur supprimer le dossier www avant)

et on passe dans **.env: APP_ENV=prod**

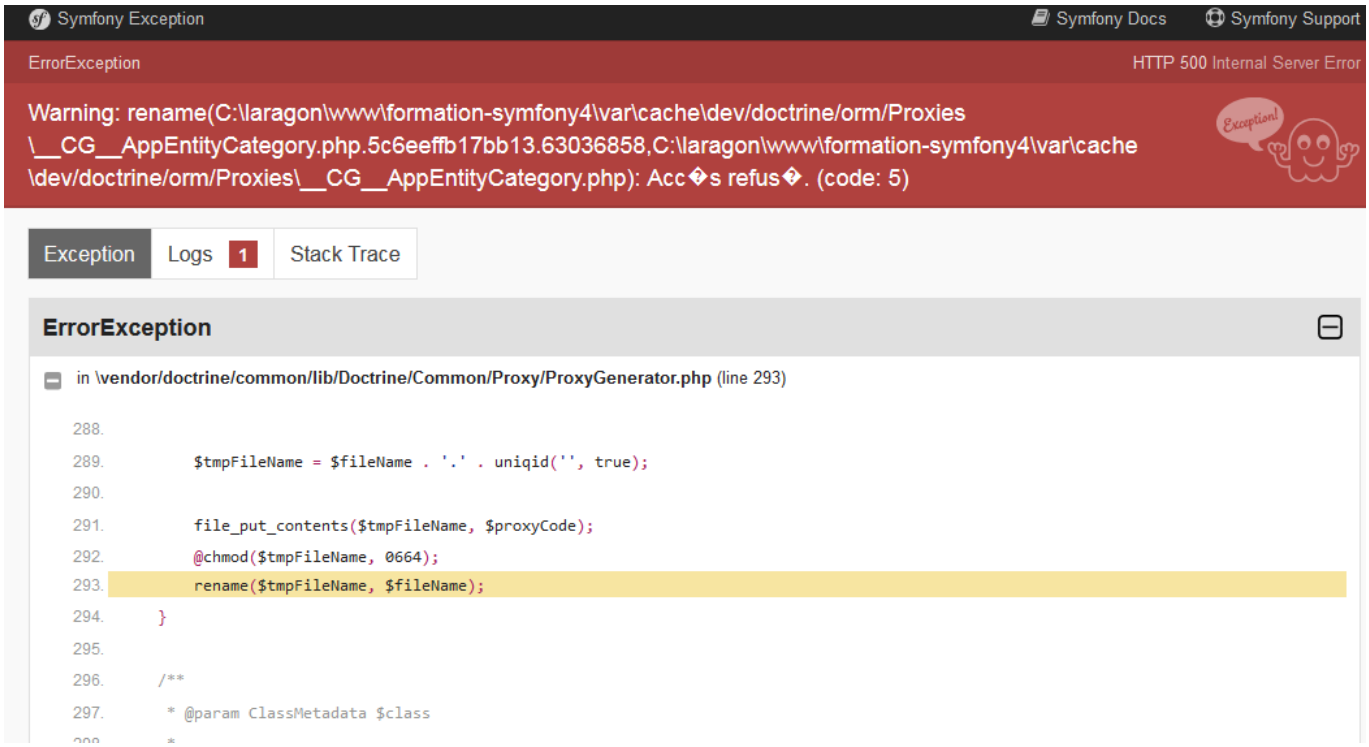
**Mise à jour:**
On peut faire en local et envoyer sur git
Puis en ssh: $ **git pull**

### 3) HEBERGEMENT MISE EN LIGNE SANS ACCES SSH:

On copie l'ensemble des fichiers par ftp
On modifie le fichier **.env**
Dans phpmyadmin on importe la base sql (que l'on a exporté en désactivant la vérification des clés étrangères )

Très compliqué par contre pour faire les mise à jour.

### 4) PARFOIS PROBLEME LORS NAVIGATION




Mettre **auto_generate_proxy_classes** à **false**. Dans config >packages >doctine.yaml

### 5) LE CACHE:

Il permet d'améliorer les performance de framework en enregistrant le résultat d'opération lourdes en évitant de les refaire.
Si le système da cache pose problème, on peut le vider manuellement via la ligne de commande
***php bin\console cache:clear***

### 6) MIGRATION:

Si il y a un problème lors d'une réinstallation et de l'application des migrations:

- Effacer toutes les migrations
- php bin/console doctrine:migrations:diff
- php bin/console doctrine:migrations:execute --up 20190429111218 (exemple)

### 7) DUMP:

#### a) Dump coté php:

https://symfony.com/doc/current/components/var_dumper.html

Lorsque l'on fait un dump.
Le + indique une propriété publique
Le – indique une propriété privée
Le # indique une propriété protégée (protected)

```
PropertyExample {#14 ▼
  +publicProperty: "The `+` prefix denotes public properties,"
  #protectedProperty: "`#` protected ones and `-` private ones."
  -privateProperty: "Hovering a property shows a reminder."
}
```

- **Exemple avec $form:**

```
dump($form);
```

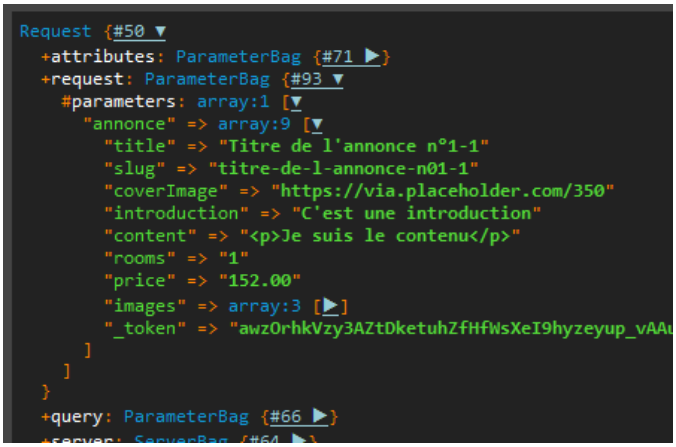
```
Form {#1385 ▼
  -config: FormBuilder {#1405 ▶}
  -parent: null
  -children: OrderedHashMap {#1395 ▶}
  -errors: []
  -submitted: false
  -clickedButton: null
  -modelData: Ad {#1026 ▶}
  -normData: Ad {#1026 ▶}
  -viewData: Ad {#1026 ▼
    -id: 478
    -title: "Titre de l'annonce n°1-1"
    -slug: "titre-de-l-annonce-n01-1"
    -price: 152.0
    -introduction: "C'est une introduction"
    -content: "<p>Je suis le contenu</p>"
    -coverImage: "https://via.placeholder.com/350"
    -rooms: 1
    -images: PersistentCollection {#1060 ▶}
    -author: User {#1099 ▶}
    -bookings: PersistentCollection {#1098 ▶}
    -comments: PersistentCollection {#1096 ▶}
  }
  -extraData: []
  -transformationFailure: null
  -defaultDataSet: true
  -lockSetData: false
}
```

Si on veut accéder à viewdata (propriété privée)
dump($form->getData());

```
Ad {#1026 ▼
  -id: 478
  -title: "Titre de l'annonce n°1-1"
  -slug: "titre-de-l-annonce-n01-1"
  -price: 152.0
  -introduction: "C'est une introduction"
  -content: "<p>Je suis le contenu</p>"
  -coverImage: "https://via.placeholder.com/350"
  -rooms: 1
  -images: PersistentCollection {#1060 ▶}
  -author: User {#1099 ▶}
  -bookings: PersistentCollection {#1098 ▶}
  -comments: PersistentCollection {#1096 ▶}
}
```

Et si on veut accéder au title qui est une propriété privée. ( Ad est un objet )
dump($form->getData()->getTitle());

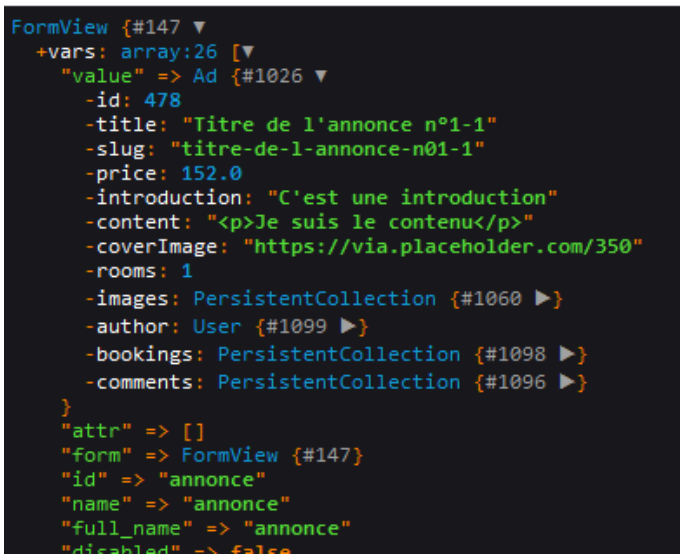
- **Exemple avec $request**
  dump($request);




Si on veut accéder à title
dump($request->request->get('annonce')['title']);
(On ne peut pas faire dump($request->request->get('annonce')->title); car annonce n'est pas un objet mais un tableau )

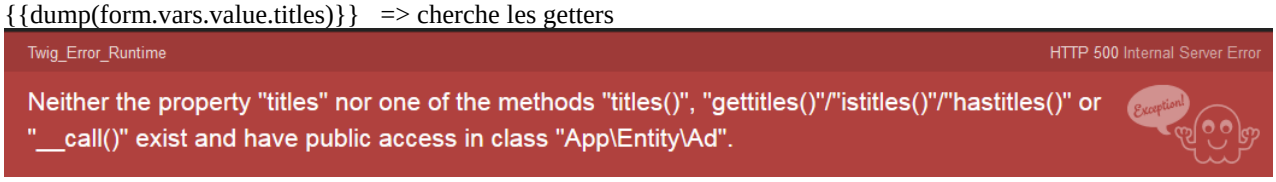
### b) Dump côté twig

- **Exemple 1:** {{dump(form)}}

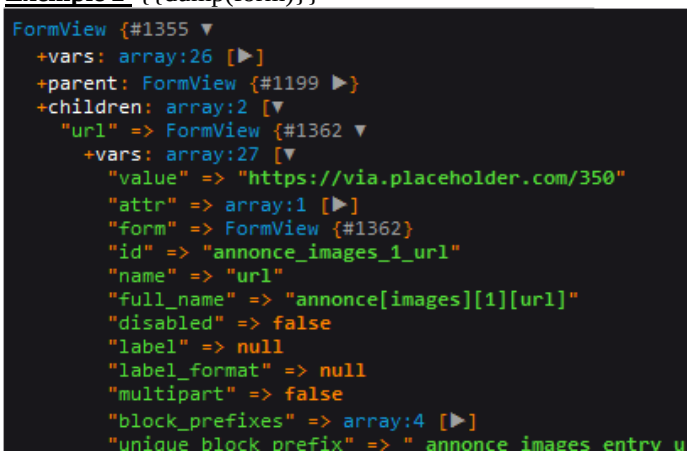



Si on veut accéder à title.
**{{dump(form.vars.value.getTitle())}}**
**Ou**
**{{dump(form.vars.value.title)}}** => twig cherche automatiquement les getters

Si on met une propriété inexistante ( titles)
{{dump(form.vars.value.titles)}} => cherche les getters



- **Exemple 2** {{dump(form)}}




Si on veut accéder à value

**{{dump(form.children.url.vars.value)}}**

**8) Problème proxy composer:**

Exemple dans le terminal:
set http_proxy = 10.40.170.12:8181
set https_proxy = 10.40.170.12:8181

pour supprimer le proxy:
set http_proxy=
set https_proxy=

9) **Problème installation symphony** ( dans le cas d'utilisation d'un proxy par exemple):
*[Seld\JsonLint\ParsingException]*
*"https://flex.symfony.com/p/symfony,flex,iv1.4.4,1562912965" does not contain valid JSON*
*Parse error on line 1:*
=> Désactiver extension Curl de php et éventuellement interbase

## UPLOAD DE PHOTOS A LA PLACE DES URLS:

**CREATION DE L'ENTITE IMAGEUPLOAD (relation entre entités):**
https://symfony.com/doc/current/controller/upload_file.html
Elle va servir à représenter l'image que l'on rajoute par upload dans l'annonce
Une annonce peut avoir une liaison avec plusieurs images et une image peut avoir une seule annonce

**php bin/console make:entity ImageUpload**

name string 255 no
url string 255 no
ad relation Ad ManyToOne no yes imageUploads yes

**php bin/console make:migration**

**php bin/console doctrine:migrations:migrate**

### 1) CREATION DU DOSSIER UPLOADS ET DU PARAMETRE DANS SERVICES.YAML:

- On créé le dossier uploads dans public/

- On créé dans config/services.yaml le paramètre qui est utilisé dans le contrôleur pour spécifier le répertoire dans lequel les images doivent être stockées:

```
parameters:
    directory_files: '%kernel.project_dir%/public/uploads'
```

### 2) CREATION DU FORMULAIRE:

- On créé manuellement une variable dans l'entité Ad:
  Par exemple:

```
public $file;   // en public pour aller plus vite, sinon il faut créer les getters et setters en private
```

- Dans le formulaire **AnnonceType.php** on va rajouter le champ **file**

```
->add('file',FileType::class, [
        'label' => false,
        'required'=>false,
        //'mapped'=>false, // en mettant mapped false, on n'a pas besoin de créer une propriété dans l'entité ImageUpload (dans notre cas, on a besoin de récupérer les données)
        'multiple'=>true,
        'attr' => ['placeholder' => 'Choose file'],


    ])
```

- Dans **new.html.twig** et **edit.html.twig** on rajoute:

```
{{ form_row(form.file) }}
```

### 3) TRAITEMENT DU FORMULAIRE PAR LE CONTROLLER ADCONTROLLER.PHP

Dans les fonction **create** et **edit**

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

        //dump($ad->getImageUploads());
      //dump($ad->file);



      foreach($ad->file as $file)
      {
     // dump($file);
      //dump($file->getClientOriginalName());

      // suppression de l'extension du fichier original
        $position_point = strpos($file->getClientOriginalName(), '.');
        $nom_original=substr($file->getClientOriginalName(), 0, $position_point);

          // encodage du fichier
        $filename = md5(uniqid()) . '.' .$file->guessExtension();
            //dump($filename);

        $upload = new ImageUpload();
        $upload->setAd($ad);
        $upload->setName($nom_original);
       $upload->setUrl('/uploads/'.$filename);
        $manager->persist($upload);

        $file->move(
            $this->getParameter('directory_files'), $filename
        );

      }

....
```

### 4) VALIDATION DU CHAMP FILE:

https://symfony.com/doc/4.4/reference/constraints/File.html
https://symfony.com/doc/current/reference/constraints/All.html
Dans l'entité Ad.php

```
    /**
     * @Assert\All({
      * @Assert\File(
     *     maxSize = "1024k",
     *     mimeTypes = {"image/jpeg"},
     *     mimeTypesMessage = "Entrer un jpg ou jpeg"
     * )
     * })
     */
public $file; // en public pour aller plus vite, sinon il faut créer les getters et setters
```

**5) <u>Visuel SHOW:</u>**

On va fusionner les deux tableaux ad.images et ad.imageUploads

```
{% set images = ad.images|merge(ad.imageUploads) %}
```

Et on remplace ad.images par la variable images

ATTENTION, il n'y a pas de légende ( caption) pour les photos uploadées.

Donc on peut rajouter par exemple un ternaire:

```
<p>{{ image.caption is not defined ? '' : image.caption }}</p>
```

**6) <u>TRAITEMENT DES IMAGES UPLOADEES DANS EDIT.HTML.TWIG ET LE CONTROLLER:</u>**

- On crée une variable dans l'entité: **Ad.php**

```
public $tableau_id; // tableau des id des images uploadées
```

que l'on rajoute dans le formulaire AnnonceType.php

```
->add('tableau_id',HiddenType::class,[
    'required'=>false,

    ])
```

- Dans edit.html.twig

```
{% for image in ad.imageUploads %}
    <img id="{{image.id}}" src="{{image.url}}" style="width:200px">
{% endfor %}
```

On crée également un champ caché qui recevra les id des images effacées

On créé également une fonction javascript:

```
$("img").click(function(event){
        //alert(event.target.id);
        $('#' + event.target.id).remove();

        var precedent = $('#annonce_tableau_id').val();

        $('#annonce_tableau_id').val(precedent + ',' + event.target.id);

});
```

- Dans la fonction edit du controller AdController.php.

```
// suppression des images uploadée si besoin
  //dump($ad->tableau_id);
  $tabid = $ad->tableau_id;
  $tabid = preg_replace("#^,#", "", $tabid);

  // on extrait les id sous forme d'un tableau
  $tabid=explode(',',$tabid);
```

```
foreach ($tabid as $id) {
   foreach ($ad->getImageUploads() as $image) {
      if ($id==$image->getId())
      {
         $manager->remove($image);
         $manager->flush();

               //on supprime le fichier
     unlink($_SERVER['DOCUMENT_ROOT'] . $image->getUrl());

      }
   };

}
```

### 7) EFFACEMENT DES FICHIERS LORS D'UNE SUPPRESSION D'ANNONCE :

On va utiliser le cycle de vie des entités.
On va dans src/Entity/ad.php et on déclare au -dessus de la classe que l'on va soumettre notre entité à des évènements de cycle de vie.

```
use Doctrine\ORM\Mapping\HasLifecycleCallbacks;

/**
 * @ORM\Entity(repositoryClass="App\Repository\AdRepository")
 * @ORM\HasLifecycleCallbacks
 */
class Ad
{
   /**
....

/**
 * permet de supprimer toutes les images du dossiers uploads associées à une entité que l'on supprime
* @ORM\PreRemove
*/
public function deleteUploadFiles(){


   foreach ($this->getImageUploads() as $image)
           {
             //dump($image);
                 //on supprime les fichiers
        unlink($_SERVER['DOCUMENT_ROOT'] . $image->getUrl());

           }

}
```

### 8) NOTION DE SERVICE ( CODE REUTILISABLE):

On remarque que l'on va avoir le même code dans la fonction **create** et la fonction **edit** du controller **AdController.php**
On introduit donc la notion de service ( extraire et mettre en commun une logique particulière)
Un service permet de centraliser du code.
On créé un dossier **src/Services** et un fichier **ImagesUploadService.php**
On fait un extends de AbstractController pour avoir la fonction getParameter à disposition.

```
<?php
```

```
namespace App\Services;

use App\Entity\ImageUpload;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;


class ImagesUploadService extends AbstractController
{
    public function upload($ad, $manager){

                                                    foreach ($ad->file as $file) {

                                        // suppression de l'extension du fichier original
                                $position_point = strpos($file->getClientOriginalName(), '.');
                                $nom_original=substr($file->getClientOriginalName(), 0, $position_point);

                                $Filename = md5(uniqid()).'.'.$file->guessExtension();

                                $upload=new ImageUpload();
                                        $upload->setName($nom_original)
                                        ->setUrl('/uploads/'.$Filename)
                                        ->setAd($ad);
                                        $manager->persist($upload);

                                        $file->move(
                                        $this->getParameter('directory_files'),
                                        $Filename);
                                                }
    }
}
```

Maintenant , il faut modifier AdController.php pour avoir acces à ImagesUploadService

Fonction create

```
/**
     * @Route("/ads/new", name="ads_create")
     */
    public function create(EntityManagerInterface $manager,Request $request, ImagesUploadService $upload)
    {
…..
                if ($form ->isSubmitted() && $form->isValid())
                    {


                        $upload->upload($ad,$manager);

….
```

Fonction edit:

```
    /**
    * @Route("/ads/{slug}/edit", name="ads_edit")
    */
    public function edit(EntityManagerInterface $manager,Request $request,Ad $ad,ImagesUploadService $upload)
    {

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

            $upload->upload($ad,$manager);

…..
```