SYMFONY

SYMFONY

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

Table des matières

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

Liste des compétences du module :

Etre capable de créer des pages Web Dynamique,

Etre capable de mettre en place un système d'API,

Etre capable de connecter une application serveur à une base de données côté Back-end,

Etre capable de gérer des requêtes HTTP d'interaction côté Back-end.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

1 Présentation du Framework :

1.1 Présentation Symfony :

Symfony est un ensemble de composants **PHP**, un **Framework** d'application Web, une philosophie et une communauté.

Un **Framework** permet aux développeurs de gagner du temps en réutilisant des modules génériques pour se concentrer sur d'autres domaines. Il va nous aider à développer mieux et plus vite !

Un **Framework** vous apporte la certitude que l'on développe une application parfaitement conforme aux règles métier, structurée, maintenable et évolutive.

C'est un ensemble de composants découplés et réutilisables sur lesquels sont construites les applications PHP : **Drupal**, **Prestashop** et **Laravel**.

2 Installation et prérequis du Framework :

2.1 Prérequis :

Windows 10,

Terminal (PowerShell, Gitbash),

Composer,

Scoop (gestionnaire de package),

Editeur de code Vscode,

Xampp (Wamp, Mamp), apache, php et MySQL.

Navigateur web (Chrome, Edge, Firefox etc.…)

2.2 Installation de scoop :

-Ouvrir le terminal PowerShell et saisir la commande suivante :

```
iex (new-object net.webclient).downloadstring('https://get.scoop.sh')
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

NB : si vous avez une erreur **SSL TLS** exécutez la commande suivante :

```
[Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12
Set-ExecutionPolicy RemoteSigned -scope CurrentUser
iwr -useb get.scoop.sh | iex
```

Lien issues : ***https://github.com/ScoopInstaller/Scoop/issues/4002***

2.3 Installation de CLI :

Toujours depuis PowerShell saisir la commande ci-dessous :

```
scoop install symfony-cli
```

NB : avec Symfony nous n'avons pas à utiliser le serveur local (**Xampp**, **wamp**), il dispose de son propre serveur intégré.

2.4 Installation de composer :

Pour installer composer téléchargez le package **Composer-Setup.exe** à l'adresse ci-dessous :

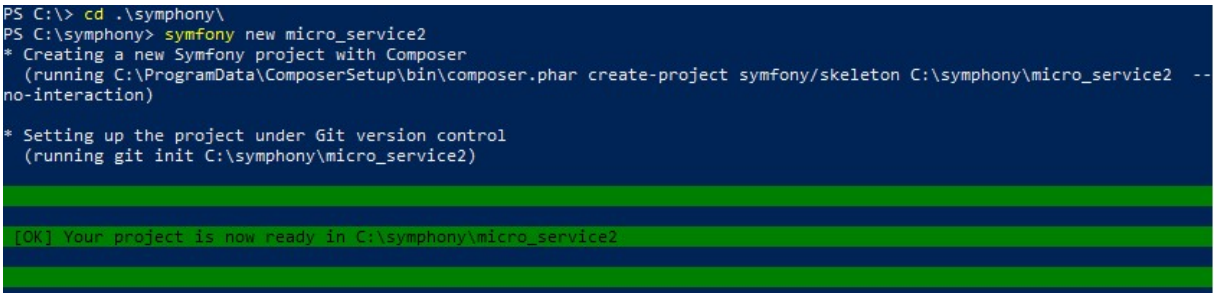
https://getcomposer.org/download/

2.5 Exemple création de projets en ligne de commande :

2.5.1 Création d'un micro service en CLI:

Depuis PowerShell saisir la commande suivante :

```
symfony new micro_service
```



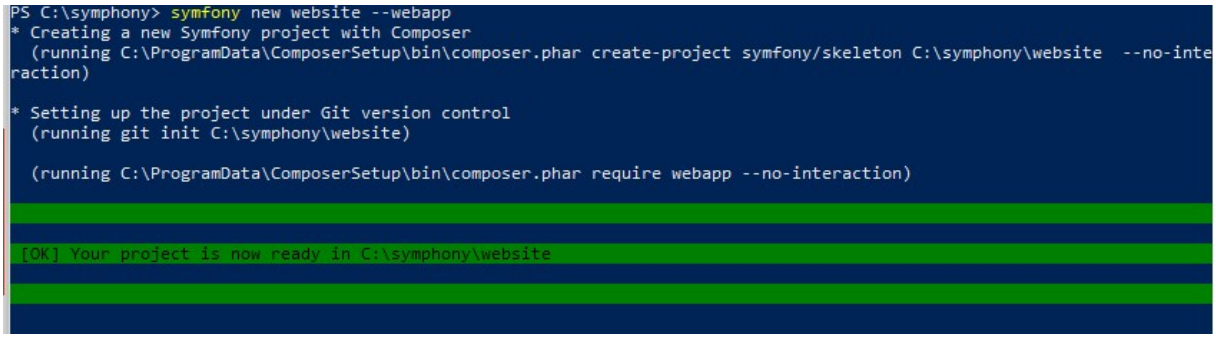
2.5.2 Création d'une Web App en CLI :

Depuis PowerShell saisir la commande suivante :

```
symfony new website --webapp
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY



2.5.3 Création d'un micro service avec composer :

Depuis PowerShell ou Gitbash saisir la commande suivante :

```
composer create-project symfony/skeleton microservice_composer
```

2.5.4 Création d'une Webapp avec composer :

```
composer create-project symfony/website-skeleton website_composer
```

NB : à la question docker taper ***n*** puis ***entrée***.

2.6 Utilisation du serveur interne de Symfony :

Pour lancer un projet avec le serveur interne de Symfony :

Ouvrir un terminal et se positionner dans le dossier du site à exécuter (par ex website),

Saisir la commande suivante :

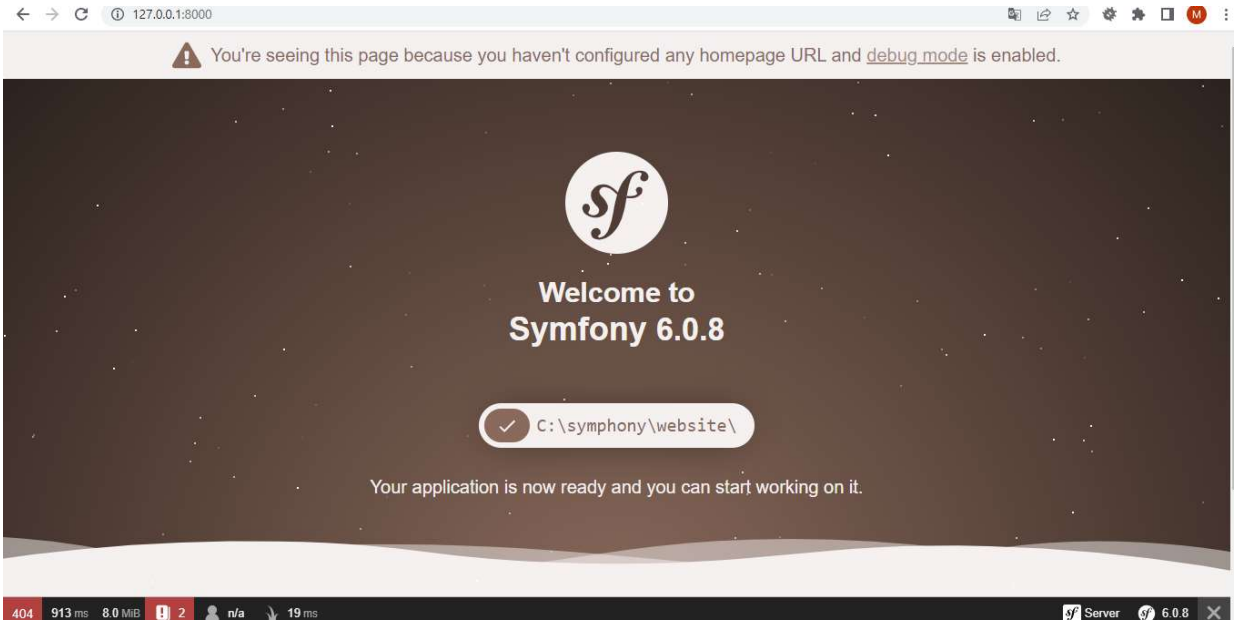
```
symfony server:start -d
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

Le site sera accessible dans le navigateur web à l'adresse suivante :

http://127.0.0.1:8000



Pour arrêter le serveur saisir la commande suivante :

```
symfony server:stop
```

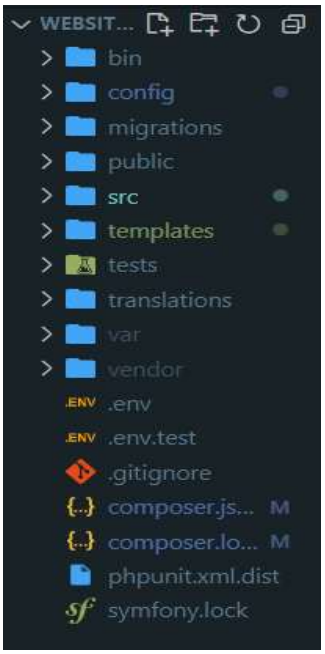
Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

3 Structure des projets :

3.1 Présentation de la structure des projets :

Les projets symfony sont structurés comme ci-dessous :



Dossier **bin** contient 2 fichiers :

Console (exécutable pour manager le projet),

Phpunit (exécutable pour le testing de l'application).

Dossier **config** (contient les informations de configuration du projet),

Dossier **migrations** (contient le code SQL du projet),

Dossier **public** (contient les fichiers et dossiers accessible au public),

Dossier **src** (contient le code source du projet, fichiers et dossier non accessible au public),

Dossier **templates** (contient le templates HTML des pages),

Dossier **test** (contient les tests du projet),

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

Dossier **translations** (contient les différentes traductions (langues)du projet),

Dossier **var** (contient les variables et les fichiers temporaires),

Dossier **vendor** (contient les dépendances du projet),

Fichier **.env** (contient les variables d'environnement).

Fichier **.env.test** (contient les variables des tests),

Fichier **.gitignore** (fichier git pour exclure les fichiers et dossier),

Fichier **composer.json** (fichier qui contient les dépendances du projet),

Fichier **composer.lock** (fichier qui contient les dépendances du projet),

Fichier **phpunit.xml.dist** (fichier de configuration des tests du projet),

Fichier **symfony.lock** (fichier de configuration des dépendances installées sur le projet).

4 Routing avec Symfony

Un des principes de base de Symfony, va être de rattacher nos **méthodes** (qui sont contenues dans nos classes **controller**) à des **routes** (à la différence de ce que l'on a vu précédemment avec PHP où nous rattachions les **routes** à des **controllers**).

Exemple :

Création d'une fonction qui va afficher bonjour dans une page web, elle sera accessible depuis la **route /bonjour**.

4.1 Création de route avec routes.yaml

Création d'un nouveau controller dans le répertoire **src/controller**

Intégrer le code ci-dessous :

```
<?php

    namespace App\Controller;


    use Symfony\Component\HttpFoundation\Response;
```

SYMFONY

```
    class HelloCtrl{

        public function sayHello():Response{
            return new Response("Bonjour");
        }

        public function sayHelloUtil($name):Response{
            return new Response("Bonjour ".$name);
        }
    }
?>
```

Paramétrage de la route dans le fichier **routes.yaml**.

Ajouter le code comme ci-dessous :

```
HelloController:
    path: /bonjour
    controller : App\Controller\HelloCtrl::sayHello
HelloControllerName:
    path: /bonjour/{toto}
    controller : App\Controller\HelloCtrl::sayHelloUtil
```

Pour tester les routes veuillez saisir :

http://127.0.0.1/bonjour

Affiche *Bonjour* (dans la page)

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

http://127.0.0.1//bonjour/Adrar

Affiche *Bonjour Adrar* (dans la page)

NB : le serveur web intégré de **symfony** doit être lancé préalablement avec la commande :

```
symfony server:start -d
```

4.2 Création de route avec les annotations :

Symfony permet de **générer automatiquement** les routes avec le système **d'annotations**. (Commentaire sur une fonction).

Pour pouvoir utiliser cette fonctionnalité nous allons devoir installer la **bibliothèque** avec la commande suivante (dans un **terminal**) :

```
composer require --dev annotations
```

Créer un nouveau controller dans le dossier src/controller

```
<?php
    //src/Controller/NameController.php
    namespace App\Controller;
    use Symfony\Component\HttpFoundation\Response;
    use Symfony\Component\Routing\Annotation\Route;
    class NameController
    {
        /**
         * @Route("/hello")
        */
        public function Hello(): Response
        {
        return new Response(
```

SYMFONY

```
                '<html><body>Hello World</body></html>'
            );
        }
        //fonction avec Paramètre (route/param)
        /**
         * @Route("/hello/{name}")
        */
        public function HelloUser($name): Response
        {
        return new Response(
                '<html><body>Hello '.$name.'</body></html>'
            );
        }
    }
?>
```

Pour générer les routes nous utilisons la syntaxe suivante :

-**@Route**("**/nom_route**") pour une route simple,

-**@Route**("**/nom_route/{param}**") pour une route avec un paramètre.

4.3 Création de route avec maker-bundle :

Installer la bibliothèque **maker-bundle** avec la commande suivante (dans un terminal) :

```
symfony console make:controller
```

Nous devons saisir un nom de **controller** (exemple Accueil) :

SYMFONY

```
$ symfony console make:controller

 Choose a name for your controller class (e.g. TinyPopsicleController):
 > Accueil
```

La commande va générer automatiquement le controller Accueil et un template HTML :

```
created: src/Controller/AccueilController.php
created: templates/accueil/index.html.twig

 Success!
```

Fichier **AccueilController** (**src/controller**):

```
<?php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
class AccueilController extends AbstractController
{
    #[Route('/accueil', name: 'app_accueil')]
    public function index(): Response
    {
        return $this->render('accueil/index.html.twig', [
            'controller_name' => 'AccueilController',
        ]);
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

```
    }
}
```

Le fichier **AccueilController.php** contient : une méthode **index** qui va générer la **vue** avec la **fonction render** (qui possède **2 paramètres** en entrée : un **Template html twig** et un **tableau**).

Fichier **Template** html twig (**templates/accueil/index.html.twig**) :

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

{% block title %}Hello AccueilController!{% 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:/symphony/website_composer/src/Controller/AccueilController.php'|file_li
nk(0) }}">src/Controller/AccueilController.php</a></code></li>

        <li>Your template at <code><a href="{{
'C:/symphony/website_composer/templates/accueil/index.html.twig'|file_link(
0) }}">templates/accueil/index.html.twig</a></code></li>

    </ul>

</div>

{% endblock %}
```

Le fichier **index.html.twig** (correspond à la **vue** dans le model **MVC**) contient le style **CSS** au sein d'une balise (**style**), les scripts **JS** (balise **script**) et le code **HTML**.

4.4 Exercice :

Dans la class ***AccueilController,*** en vous inspirant de la méthode **index** généré par **make-bundle**, ajoutez une nouvelle méthode **index2** qui va :

-prendre en paramètre (un nom).

-Récupérer le paramètre dans la fonction render (Vous allez devoir pour cela modifier le tableau en paramètre et ajouter à celui-ci une entrée **'test_name'** => **$name**).

-Modifier le **template html** pour gérer l'affichage du nom dans la page.

-Créez une **route** pour appeler la fonction.

Le nom devra être récupéré dans la route comme ceci : **/accueil/nom**.

5 Les templates Twig :

Les templates Twig sont des fichiers qui vont contenir le code HTML et CSS de nos vues. Si nous avons généré un projet complet nous avons dans le dossier templates un fichier **base.html.twig**, celui-ci va contenir le code HTML.

Si le **package twig** n'est pas présent dans notre projet nous allons l'intégrer avec la commande suivante (dans un **terminal**) :

```
composer require --dev twig
```

5.1 Block de code HTML :

Les pages HTML contiennent des blocs de code avec la syntaxe suivante :

```
{% block title %}titre de la page{% endblock %}
```

Ces blocs (encadrés par des accolades) fonctionnent comme des variables, nous allons pouvoir les modifier en fonction de nos besoins en les surchargeant (réécriture de ceux-ci).

Dans la partie précédente (**routing**) nous avons généré une page avec **make-bundle** (**accueil**).

Nous allons remplacer son contenu afin de comprendre l'utilisation des blocs.

SYMFONY

Balise extends :

La balise **extends** sert à indiquer depuis quel fichier elle doit hériter la structure html de base dans notre cas nous allons nous servir du fichier **base.html.twig** qui se trouve à la racine de templates.

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

Balise title :

La balise title permet de redéfinir le titre de notre page (comme en HTML). Elle s'ouvre avec **block** et se referme avec **enblock**.

```
{% block title %} Accueil {% endblock %}
```

Balise body :

La balise body permet de redéfinir le corps de la page (body en HTML). Elle s'ouvre avec **block** et se referme avec **enblock**. (Elle va **surcharger** le body de **base.html.twig**)

```
{% block body %}
    <h1>
        Bienvenue sur la page Accueil
    </h1>
{% endblock %}
```

5.2 Tester son interface :

Pour tester notre **interface** nous allons revenir dans le **controller accueil** et réécrire la méthode **index** comme ci-dessous :

```
/*Attention !!! pensez à renommer le name (app_accueil en app_accueil2) si vous avez plusieurs routes dans votre controller */
 #[Route('/home', name: 'app_accueil2')]
 public function index(): Response
 {
    return $this->render('accueil/index.html.twig') ;
 }
```

SYMFONY

La page est accessible à l'adresse suivante :

localhost :8000/home ou **localhost :8000/accueil**

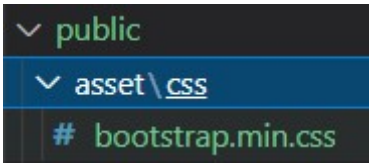
5.3 Intégrer des assets (Bootstrap) à un projet :

Pour utiliser des **thèmes Bootstrap** dans un projet nous allons les télécharger depuis le site ci-dessous :

https://bootswatch.com/

Nous allons créer à l'intérieur du répertoire **public** (à la **racine** du projet) un dossier **asset->css**

Nous déplacerons le fichier **bootstrap.min.css** téléchargé précédemment à l'intérieur de celui-ci :



5.4 Liaison fichier Bootstrap à base.html.twig :

Pour **lier** le fichier **style CSS** à notre page nous allons intégrer le code ci-dessous :

```
{% block stylesheets %}
    <link rel="stylesheet" href="../asset/css/bootstrap.min.css">
{% endblock %}
```

NB : Le fichier **style CSS** se trouve dans le répertoire **public/asset/css/**.

5.5 Exercice :

Recréer la vue index.html.twig en intégrant :

-Un Menu Bootstrap, (Menu, liens (Accueil, Présentation, Prix et A propos)

-Un titre H1 avec Bienvenue sur la page d'accueil

Un titre H1 avec Bienvenue sur la page d'accueil **prénom** (version qui récupère le prénom dans la **route**)

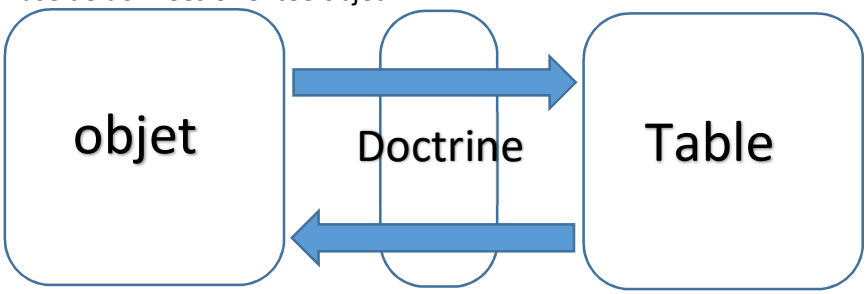
Réaliser les 2 versions (avec et sans le prénom : 2 méthodes dans la classe **AccueilController**)

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

6 ORM et Doctrine :

Un **ORM** (**Object-Relationnal Mapping**) est un programme qui va servir d'interface entre une base de données relationnelle (Nous utilisons **MySQL** ou **MariaDB**) et une application afin de **simuler** une Base de données orientée objet.



6.1 Création d'une base de données avec Symfony :

Symfony va nous permettre de créer les bases de données de façon automatique.

Pour se faire nous allons utiliser l'outil **make-bundle** avec la commande suivante (depuis un **terminal**) :

symfony console make:entity

L'outil (**make-bundle**) va nous demander de choisir un nom pour **l'entité** (**table**), nous allons saisir **Article**.

```
Class name of the entity to create or update (e.g. VictoriousElephant):
> Article

created: src/Entity/Article.php
created: src/Repository/ArticleRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.
```

Il va nous demander ensuite d'ajouter les attributs et leurs types, nous allons saisir **title** à la question :

```
New property name (press <return> to stop adding fields):
> title
```

SYMFONY

Nous allons ensuite définir le **type** de **l'attribut** :

NB : Pour connaitre les différents **types** il suffit de saisir ***?*** et ***entrée*** pour afficher la liste de celle-ci.

```
Main types
  * string
  * text
  * boolean
  * integer (or smallint, bigint)
  * float

Relationships / Associations
  * relation (a wizard will help you build the relation)
  * ManyToOne
  * OneToMany
  * ManyToMany
  * OneToOne

Array/Object Types
  * array (or simple_array)
  * json
  * object
  * binary
  * blob

Date/Time Types
  * datetime (or datetime_immutable)
  * datetimetz (or datetimetz_immutable)
  * date (or date_immutable)
  * time (or time_immutable)
  * dateinterval

Other Types
  * ascii_string
  * decimal
  * guid
```

Nous allons saisir **string** pour le type, **100** pour la **longueur** du champ et **no** (not null)

```
Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 100

Can this field be null in the database (nullable) (yes/no) [no]:
> no
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

Nous allons ajouter les champs supplémentaires suivants :

-**contenu** (nom attribut), **text** (type), **no** (not null),

```
Add another property? Enter the property name (or press <return> to stop adding fields):
> contenu

Field type (enter ? to see all types) [string]:
> text

Can this field be null in the database (nullable) (yes/no) [no]:
> no
```

-**image** (pas de **type**, **longueur** et **not null**), Symfony va utiliser les attributs par défaut (**string**, **255** et **no**) en tapant juste sur la touche ***entrée***.

```
Add another property? Enter the property name (or press <return> to stop adding fields):
> image

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
>
```

-**createThe** (nom attribut), **datetime** (type) et **no** (not null) :

```
Add another property? Enter the property name (or press <return> to stop adding fields):
> createThe

Field type (enter ? to see all types) [string]:
> datetime

Can this field be null in the database (nullable) (yes/no) [no]:
> no
```

Pour terminer la création de la **table** nous allons simplement utiliser la touche ***entrée***.

```
updated: src/Entity/Article.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>


 Success!

Next: When you're ready, create a migration with php bin/console make:migration
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

L'outil **make-bundle** crée automatiquement la classe dans le **répertoire src/Entity/Article.php** (car nous avons donné comme nom à la table **Article**).

Le fichier contient la classe **Article**, les **attributs** et les **Getter** et **Setter**.

NB : Notons que nous n'avons pas besoin de créer l'attribut **id** l'outil (**make-bundle**) le fait pour nous

Exercice :

En utilisant la commande suivante (dans un terminal) :

symfony console make:entity

Ajoutez les entités suivantes :

User (name, firstName, mail, password et createAt),

Category (title, description, createAt).

NB : Nous devons répéter la commande ***make:entity*** pour chaque entité (taper sur la touche ***entrée*** quand l'outil demande d'ajouter un attribut). Nous devons nous retrouver avec les 3 classes suivantes dans le répertoire **src/Entity/** : **Article**, **User** et **Category.**

6.2 Modifier les entités (classes) et ajout des contraintes :

Avec **Symfony** plutôt que de gérer nos tables d'association et les clés étrangères nous allons utiliser l'outil de génération d'entités (make-bundle).

Pour cela nous allons réutiliser la commande suivante :

symfony console make:entity

A l'invite de la commande nous allons saisir le nom d'une entité existante à laquelle nous souhaitons en connecter une autre (Article et Category) pour pouvoir lister les Articles par catégories.
Nous allons saisir Article :

```
mathieumithridate@FORMATEURS17-19 MINGW64 /c/symphony/website_composer (master)
$ symfony console make:entity

 Class name of the entity to create or update (e.g. OrangeGnome):
 > Article

 Your entity already exists! So let's add some new fields!
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

L'outil demande si nous souhaitons ajouter des nouveaux champs (**Field**) à l'entité.

Nous allons saisir le nom du champ pour les relier, dans ce cas-ci **Category** (qui va représenter la liaison avec l'entité **Category**) :

```
New property name (press <return> to stop adding fields):
> category
```

Dans l'invite de commande pour le choix du **type** nous allons choisir **relation** (la liste complète est accessible en cliquant sur ***?*** et ***entrée***) **:**

```
Field type (enter ? to see all types) [string]:
> relation
```

Elle va ensuite nous demander la classe avec (**entity**) à laquelle nous souhaitons la relier. Nous allons choisir **Category** (à ne pas confondre avec le champ créé qui lui se nomme **category**):

```
What class should this entity be related to?:
> Category
```

Il va nous falloir définir le type de **relation** entre les **Classes** :

Many To One : Chaque **objet Article** possède un **objet Category**, un **objet Category** possède plusieurs **objets Article** (représente une relation **0.1 0.N**),

One To Many : Chaque **objet Article** possède plusieurs **objets Category**, un **objet Category** pointe vers un seul **objet Article** (représente une relation **0.N 0.1**),

Many To Many : Chaque **objet Article** possède plusieurs **objets Category**, chaque **objet Category** possède plusieurs **objets Article** (représente une relation **0.N 0.N** -> **table d'association**),

One To One : Les **objets Article** sont associés à un **seul objet Category**, les **objets Category** sont associé à un **seul objet Article** (représente une relation **0.1 0.1**).

Dans notre cas nous allons choisir la relation **Many To Many.**

```
Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToMany
```

A la question ajouter une nouvelle propriété à la classe **Category** nous cliquons sur ***entrée*** pour accepter (**make:entity** va effectuer la même action que dans **Article** au sein de la classe **Category**).

6.3 Configuration de l'accès à la base de données :

Afin de créer la base de données sur notre serveur MySQL nous allons configurer le fichier **.env** qui se trouve à la racine du projet **Symfony**.

6.3.1 Editer le fichier .env :

Dans ce fichier nous trouvons à la fin de celui-ci 3 modes de connexions (**PostgreSQL**, **Sqlite** ou **MySQL**).

```
29  # DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
30  # DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7&charset=utf8mb4"
31  DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
```

Nous allons commenter avec **#** les lignes **PostgreSQL** et **Sqlite** et dé commenter **MySQL**.

La connexion à la base de données et les divers paramètres sont représentés de cette façon :

mysql://nom_compte:mot_de_passe@url_serveur_bdd/nom_bdd?version&langue_base

Exemple :

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/symfonyBDD?serverVersion=5.7&charset=utf8mb4"
```

6.3.2 Créer la base avec la Doctrine :

Pour créer la base de données sur notre serveur **MySQL** nous n'avons pas besoin de le faire à la main par avance. **Symfony** va la créer pour nous, en utilisant la commande suivante (dans un **terminal**) :

symfony console doctrine:database:create

NB : attention à bien dé commenter la ligne ***extension=pdo_mysql*** (enlever le **;**) dans le fichier **php.ini** de votre serveur web (**Xampp**, **Wamp** etc…).

Si les informations de connexion sont valides Symfony va créer la base sur notre serveur. Celle-ci sera par contre vide (elle ne contient pas les tables), nous allons avoir besoin d'utiliser l'outil make-bundle pour les ajouter.

6.4 Effectuer une Migration :

Pour transformer nos **classes** (générées avec **make-bundle**) en **table** dans une base de données, nous allons effectuer l'opération de **migration**.

Une **migration** est une **classe** du projet qui va décrire les changements à apporter, pour mettre à jour un schéma de base de données, de son état actuel (**vide**) vers le nouveau (**tables ajoutées**), en fonction des **attributs** des différentes **entités** (**classe**). La migration va alors consister à créer les 3 **tables** au sein de la base de données (**Article**, **Category**, **User**).

6.4.1 Créer le fichier de migration :

Nous allons dans un premier temps générer un fichier pour la migration qui va être nommé comme ceci :

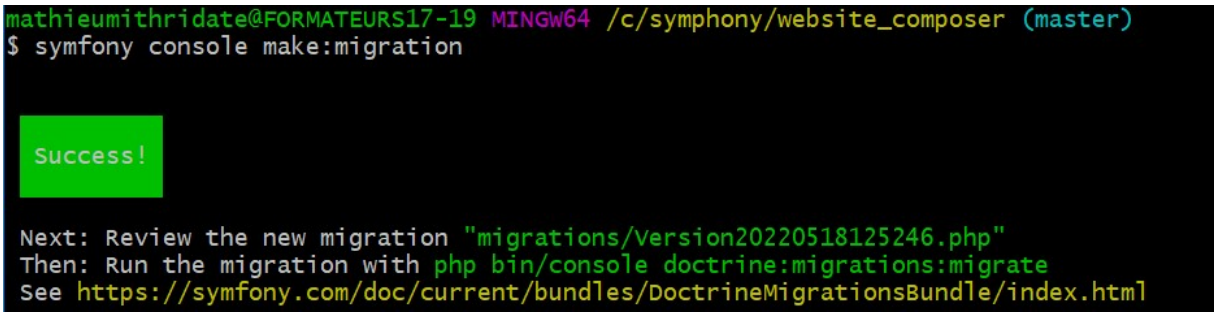
VersionDatedujourNumaleatoire.php : **Version20220518211615 64.php**

Il se trouvera dans le répertoire **/migrations** du projet.

Pour le créer nous utiliserons la commande suivante (dans un **terminal**) :

```
symfony console make:migration
```

Si la migration se déroule comme il faut, vous aurez un message tel que :



NB : Il faut vérifier que le fichier **version.php** a bien été ajouté dans le répertoire **/migrations** du projet.

Le fichier **version.php** contient une **classe** et différentes **méthodes**. La fonction qui va nous intéresser est : ***up***, elle contient le code SQL de création des différentes **tables**. Au besoin nous pouvons éditer le code pour ajouter, modifier ou supprimer des **attributs** (attention que les **attributs** existent dans les **classes** correspondantes).

6.4.2 Effectuer la migration :

Pour créer les tables dans notre base de données nous allons utiliser le service de **migration** de **doctrine** en utilisant la commande suivante (dans un **terminal**) :

symfony console doctrine:migrations:migrate

Il faut répondre ***Yes*** pour continuer. Attention la doctrine migration va supprimer la totalité des tables et les données.

```
$ symfony console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "symfonybdd" that could result
in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
>
```

Si tout s'est bien déroulé nous aurons un message tel que :

```
[notice] Migrating up to DoctrineMigrations\Version20220518125246
[notice] finished in 1214.2ms, used 20M memory, 1 migrations executed, 6 sql queries
```

Nous pouvons vérifier au niveau de la base de données (avec par ex : **PhpMyAdmin**)

- article
- article_category
- category
- doctrine_migration_versions
- user

Nous pouvons remarquer qu'il y a une table qui ne correspond pas à notre structure :

doctrine_migration_versions.

Cette table va stocker les différents enregistrements des **doctrines** de **migration** (fichier **version.php**).

6.5 Exercice :

Mettez en pratique les notions abordées jusqu'à présent (Chapitre **6 Doctrine et ORM**), en ajoutant la relation entre **User** et **Article**.

SYMFONY

NB : Si après votre migration vos routes ne fonctionnent plus, il faut éditer le fichier **.env** et remplacer la connexion à la BDD avec le code suivant :

```
DATABASE_URL=mysql://root:@127.0.0.1:3306/testtest?serverVersion=mariadb-10.4.11
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

7 ORM-Fixtures et Faker :

Régulièrement quand on développe un projet, nous avons besoin d'ajouter des données factices afin de tester nos fonctionnalités. Nous allons voir un outil, intégrable à Symfony qui va générer pour nous des données de test aléatoires.

7.1 Installation de l'outil ORM-Fixtures :

Pour installer la dépendance nous allons saisir la commande suivante (dans un terminal) :

composer require --dev orm-fixtures

```
Symfony operations: 1 recipe (0db395d46a72ba46fda430aeecc74375)
  - Configuring doctrine/doctrine-fixtures-bundle (>=3.0): From
s:main
Executing script cache:clear [OK]
Executing script assets:install public [OK]

 What's next?
```

7.2 Création de la classe AppFixtures.php :

Nous allons éditer dans le répertoire **/src/DataFixtures** le fichier php suivant :

AppFixtures.php va contenir le code qui va nous permettre de générer des jeux de données dans notre BDD.

Nous allons éditer le fichier avec le code ci-dessous (**partie 1**) :

```
<?php

    namespace App\DataFixtures;

    use App\Entity\User;

    use Doctrine\Bundle\FixturesBundle\Fixture;

    use Doctrine\Persistence\ObjectManager;

    use Faker;

    class AppFixtures extends Fixture
```

```
    {
        public function load(ObjectManager $manager): void
        {
            //Tableau vide qui va stocker les utilisateurs que je génère
            $users = [];
            //Boucle qui va itérer 20 utilisateurs factices
            for($i=0; $i<20; $i++){
                $user = new User();
                $user->setName();
            }
            $manager->flush();
        }
    }
```

7.3 Installation de la librairie Faker :

Pour intégrer la librairie Faker qui va générer des données aléatoires au projet nous allons saisir la commande suivante (dans un terminal) :

```
composer require fakerphp/faker
```

SYMFONY

```
$ composer require fakerphp/faker
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^1.19 for fakerphp/faker
./composer.json has been updated
Running composer update fakerphp/faker
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking fakerphp/faker (v1.19.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading fakerphp/faker (v1.19.0)
  - Installing fakerphp/faker (v1.19.0): Extracting archive
Generating optimized autoload files
109 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.

Executing script cache:clear [OK]
Executing script assets:install public [OK]
```

7.4 Intégration de Faker :

Nous allons éditer le fichier **AppFixtures.php** qui se trouve dans le répertoire **/src/DataFixtures** en remplaçant le contenu par le code suivant :

```
<?php

    namespace App\DataFixtures;

    use Faker\Factory;

    use App\Entity\User;

    use Doctrine\Bundle\FixturesBundle\Fixture;

    use Doctrine\Persistence\ObjectManager;

    use Faker;

    class AppFixtures extends Fixture

    {

        public function load(ObjectManager $manager): void

        {
```

```
            //création d'une variable qui va contenir
            $faker = Faker\Factory::create();
            //Tableau vide qui va stocker les utilisateurs que l'on génère
            $users = [];
            //Boucle qui va itérer 20 utilisateurs factices
            for($i=0; $i<20; $i++){
                $user = new User();
                //génération d'un utilisateur factice
                $user->setName($faker->name());
                $user->setFirstName($faker->firstname());
                $user->setMail($faker->email());
                $user->setPassword($faker->password());
                $user->setCreateAt(new \DateTime());
                //stockage dans le manager
                $manager->persist($user);
                $users[] = $user;
            }
            $manager->flush();
        }
    }
```

7.5 Exercice :

Utiliser Faker pour générer 10 **Category** et 100 **Article**, réaliser le code correspondant dans le la fonction **load** du fichier **AppFixtures.php**.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

7.6 Sauvegarder des données en BDD :

Maintenant que nous avons généré des enregistrements factices nous allons les sauvegarder dans la base de données **MySQL**.

Pour lancer l'ajout des enregistrements en **BDD** nous allons utiliser la commande suivante (dans un **terminal**) :

symfony console doctrine:fixtures:load

```
$ symfony console doctrine:fixtures:load

Careful, database "symfonybdd" will be purged. Do you want to continue? (yes/no) [no]:
> yes

  > purging database
  > loading App\DataFixtures\AppFixtures
```

NB : attention **Symfony** va nous indiquer qu'il va vider la base de données (saisir ***yes***).

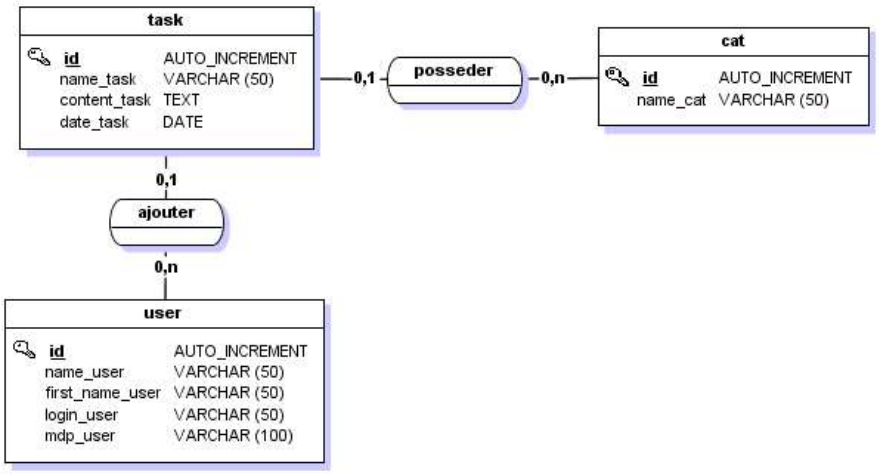
SYMFONY

8 Création d'une API (Micro-services) :

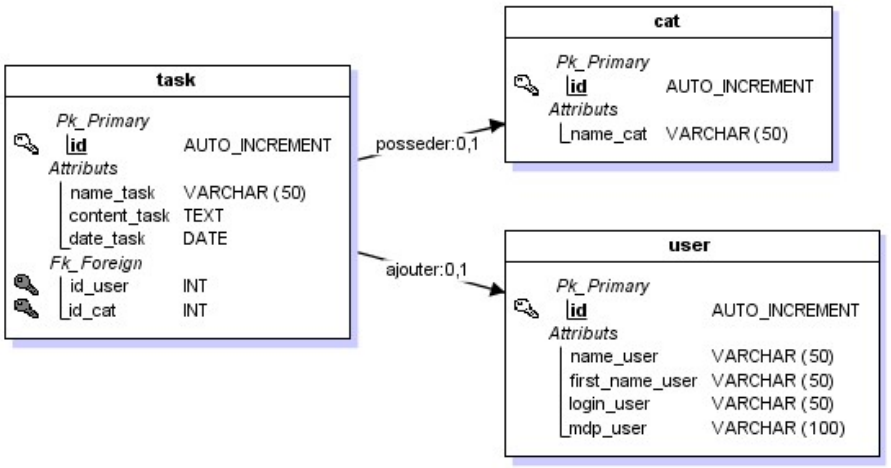
Dans cette partie nous allons voir comment créer une **API** en utilisant les **micro-services.** Nous allons reconstruire **l'API** du projet **task** que nous avions développé dans le module **PHP**.

8.1 MCD MLD du projet task :

MCD :



MLD :



Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

8.2 Définitions :

Un **micro-services** est un logiciel qui ne va faire qu'une seule tâche mais de façon optimale (par exemple renvoyer une liste d'article).

Une **API** est une interface de programmation qui va permettre d'interagir avec un ou plusieurs **micro-services** pour accéder, ajouter, modifier, supprimer des enregistrements en base de données, en passant par une **URL** et des paramètres
(Exemple d'**API** : https://adrardev.fr/task/api/task.php?task=1).

8.3 Structure de l'API :

Nous allons utiliser **composer** pour créer la base du projet sous la forme de micro-services, pour se faire saisir la commande suivante (depuis un **terminal**) :

composer create-project symfony/skeleton microservice_composer

8.4 Création des controllers :

Comme vu précédemment nous allons créer nos controller avec make-Bundle, saisir la commande suivante (dans un **terminal**) :

symfony console make:controller

Nous allons rencontrer l'erreur suivante :

```
There are no commands defined in the "make" namespace.

You may be looking for a command provided by the "MakerBundle" which is currently no
t installed. Try running "composer require symfony/maker-bundle --dev".
```

Celle-ci est normale car nous avons créé un projet sous forme de **micro-services**, **Symfony** n'installe alors que les composants de base.

Symfony nous propose pour installer l'outil la commande suivante à saisir (dans un **terminal**) :

composer require symfony/maker-bundle --dev

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

Quand nous allons tenter de relancer la commande précédente (***symfony console make:controller***), nous pouvons avoir une nouvelle erreur :

```
[ERROR] Missing package: to use the make:controller command, run:

composer require doctrine/annotations
```

Symfony nous propose pour installer l'outil la commande suivante à saisir (dans un **terminal**) :

composer require doctrine/annotations

Cela va nous rajouter l'outils **d'annotations** pour générer nos routes.

Nous pouvons maintenant générer nos controllers :

Task, **Cat** et **User**, **make-bundle** va générer les 3 controllers **TaskController**, **CatController** et **UserController** dans le répertoire **src/Controller**.

Nous devons relancer la commande ***symfony console make:controller*** 3 fois.

Pour vérifier le bon fonctionnement nous pouvons tester dans le navigateur internet les URL suivantes :

localhost :8000/task, localhost :8000/cat, localhost :8000/user, elles doivent renvoyer un json comme ci-dessous :

```
{"message":"Welcome to your new controller!","path":"src\/Controller\/CatController.php"}
```

NB : Si jamais cela ne fonctionne pas il faut arrêter et relancer le serveur, si vous avez déjà un site qui tourne sur le serveur, le nouveau (api) prendra le port 8001 (avec les commandes : ***symfony server:stop*** et ***symfony server:start -d***).

8.5 Créer les entités :

En utilisant les notions du **chapitre 6 Doctrine et ORM**, nous allons créer les différentes **entités** correspondantes à nos 3 **controllers**. Avec l'aide de l'outil **make-bundle** et de la commande suivante à saisir (dans un **terminal**) :

symfony console make:entity

SYMFONY

Nous allons devoir créer les **entités** comme défini dans les **diagrammes MCD MLD** (qui se trouvent plus haut dans ce chapitre). Pour reproduire les **Foreign Key** nous utiliserons les **relations** (**many to many**, **one to many**, etc…).

Elles seront les suivantes (**relations**) :

Task many to one avec **Cat**,

Task many to one avec **User**,

User one to many avec **Task**,

Cat one to many avec **Task**.

NB : Nous pouvons voir apparaitre l'erreur suivante :

```
 [ERROR] Missing package: to use the make:entity command, run:

composer require orm
```

Dans ce cas-là, nous allons simplement devoir installer ORM (répondre ***n*** à la question **docker module** puis ***entrée***) avec la commande suivante à saisir (dans un **terminal**) :

```
composer require orm
```

8.6 Configurer l'accès à la base de données :

Nous allons devoir configurer le fichier **.env** avec les informations de connexion à la base de données.

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/taskapi?serverVersion=5.7&charset=utf8mb4"
```

NB : Nous éditerons le fichier comme vu dans la partie 6.3.1, pensez à commenter les lignes avec un **PostgreSQL** et **Sqlite** avec un **#**.

8.7 Créer la base de données :

Nous allons créer la base en utilisant la commande suivante (dans un **terminal**)

```
symfony console doctrine:database:create
```

NB : En cas d'erreurs, vérifier si les informations dans le fichier **.env** sont correctes, que vous n'ayez pas une base qui porte le même nom sur votre serveur, enfin que le serveur **MySQL** soit bien lancé (port 3306-3307 **MariaDB**).

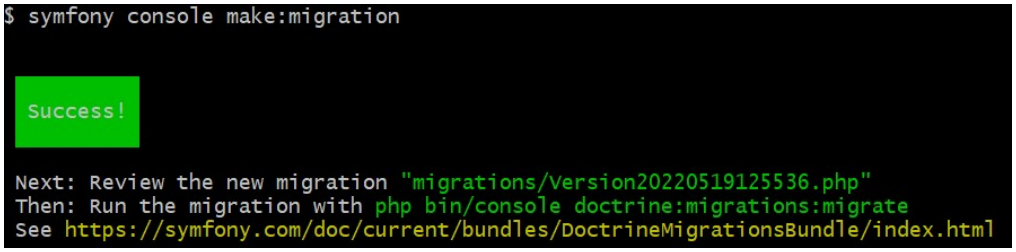
Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

8.8 Paramétrage et migration :

Créer le fichier de migration **version.php** en utilisant la commande suivante (dans un **terminal**) :

symfony console make:migration



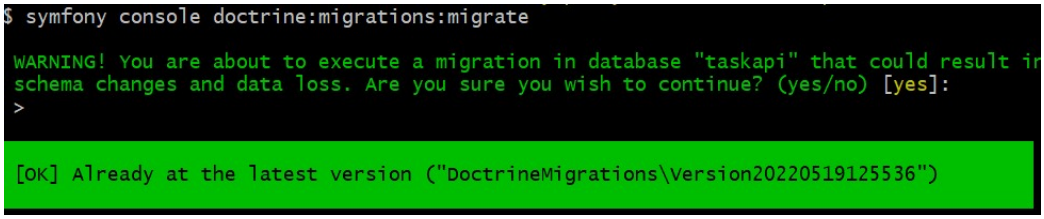
Nous pouvons vérifier que celle-ci c'est bien déroulé en allant dans le répertoire **/migrations**

Si vous avons bien notre fichier **version….php**

Ensuite nous allons effectuer la migration en utilisant la commande suivante (dans un **terminal**) :

symfony console doctrine:migrations:migrate

Si la **procédure** s'est bien déroulé, nous devons avoir un **message** de la sorte dans le **terminal** :



Et les 3 tables (**Task**, **Cat** et **User**) ainsi que la table de **migration** sur notre serveur **MySQL** (**Wamp**, **Xampp**, **Mamp etc…**)

8.9 Exercice :

En utilisant les notions abordées au **chapitre 7** (**Fixtures** et **Faker**), vous allez générer des données factices : 100 utilisateurs, 400 taches et 50 catégories. Editez le fichier AppFixtures.php qui se trouve dans le répertoire : **/src/DataFixtures/**.

NB : Vous allez avoir besoin d'installer **Fixtures** et **Faker**. Pensez à bien importer vos **différentes entités** (exemple : **use App\Entity\Task**) et la librairie **Faker** (use Faker).

NB : Vous trouverez à l'adresse ci-dessous des exemples d'utilisation de **Faker** :

https://fakerphp.github.io/formatters/

9 Création d'une API suite (outils et Méthodes) :

Dans ce chapitre nous allons voir comment utiliser des outils de test et comment créer des méthodes qui vont nous retourner des fichiers JSON.

9.1 Installation de Postman :

Afin de tester et d'afficher les retours de notre **API** (**GET, POST, PUT, PATCH, DELETE**, etc…), nous allons installer **Postman**.

L'outil se télécharge à l'adresse ci-dessous :

https://www.postman.com/downloads/

Il fonctionne sous **Windows**, **Mac OS** ou **LINUX**, nous pouvons également utiliser une **extension** de **Google Chrome** qui se nomme **Advanced Rest Client**.

Elle est disponible à l'adresse ci-dessous :

https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo?hl=fr

9.2 Edition des controllers :

Afin de vérifier si nous récupérons bien les données depuis la base de données, nous allons éditer le **TaskController.php** qui se trouve dans le répertoire **src/controller** comme ci-dessous :

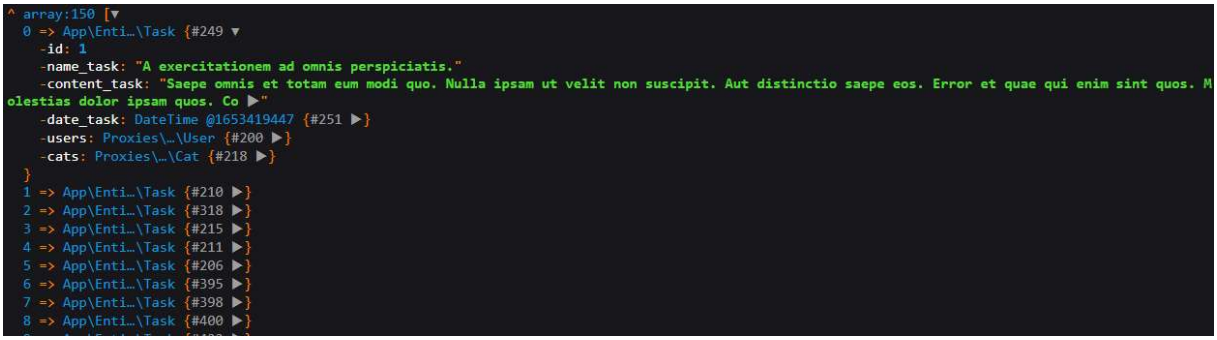
```
<?php

namespace App\Controller;

use App\Entity\Task;

use App\Repository\CatRepository;

use App\Repository\TaskRepository;

use App\Repository\UserRepository;

use Doctrine\ORM\EntityManagerInterface;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

use Symfony\Component\Serializer\SerializerInterface;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;


class TaskController extends AbstractController

{

    // je rajoute à la route la méthode associée GET

    #[Route('/task', name: 'app_task_index', methods: 'GET')]

    public function index(

        TaskRepository $taskRepository

    ): Response {

        //récupération de toutes les taches
```

```
        $tasks = $taskRepository->findAll();
        dd($tasks);
    }
}
```

La méthode **findAll()** est une des **méthodes** auto générée par **Symfony**, elle se trouve dans le fichier **TaskRepositoy.php** du répertoire **src/repository**. Elle va nous retourner un **tableau d'objets** de toutes les tâches en **BDD**.

La méthode **dd()** est un **var_dump()** amélioré intégré à **Symfony** qui va nous permettre de visualisé le contenu de nos **variables**. Elle est plus rapide, plus efficace et surtout nous permet d'avoir un affichage beaucoup plus lisible de nos **variables**.

Si nous exécutons la route ***localhost :8000/task*** nous allons avoir un affichage comme ci-dessous :



9.3 Exercice :

Editez les controllers **CatController.php** et **UserController.php**, pour faire en sorte que les routes correspondantes à la méthode index retourne un affichage équivalent au controller **TaskController.php**. (En utilisant la méthode **dd()**). Affichez dans **Postman** ou **Advanced Rest Client** le résultat de la requête **GET** : **localhost :8000/task**

Bonus : Ajoutez des méthodes **index2()** et les **routes** correspondantes dans chaque **controller** afin qu'elles retournent un **var_dump** plutôt que **dd** et faites une capture d'écran pour comparer les résultats.

9.4 Encodage en Json :

Nous allons modifier nos méthodes **index()** dans chaque controller (ex **TaskController**) pour faire en sorte que l'affichage soit un fichier **json**. Nous allons éditer la méthode index comme ci-dessous :

```
#[Route('/task', name: 'app_task_index', methods: 'GET')]
    public function index(
        TaskRepository $taskRepository
    ): Response {
        //récupération de toutes les taches
        $tasks = $taskRepository->findAll();
        //encodage en json
        $json = json_encode($tasks);
        dd($json);
    }
```

La route **localhost :8000/task** va nous retourner un résultat comme ci-dessous :

Le Résultat n'est pas ce que nous attendions, nous obtenons bien du **json** mais il contient un tableau d'objets vides, Pourquoi ?

Tout simplement à cause des **getters** et des attributs en **private**.

La fonction **json-encode** n'est pas assez puissante et n'arrive pas à utiliser les fonctions **get** de chaque attribut.

Pour pouvoir résoudre à Le problème nous allons devoir transformer notre tableau qui contient nos **objets**.

9.5 Sérialisation :

Nous allons installer la librairie **serialise** avec la commande ci-dessous (à lancer dans un **terminal**) :

```
composer require symfony/serializer-pack
```

Le processus de transformation est fait par **Normalizer**, il va faire passer d'un objet complexe à un tableau associatif simple, ce processus se nomme la **normalisation**.

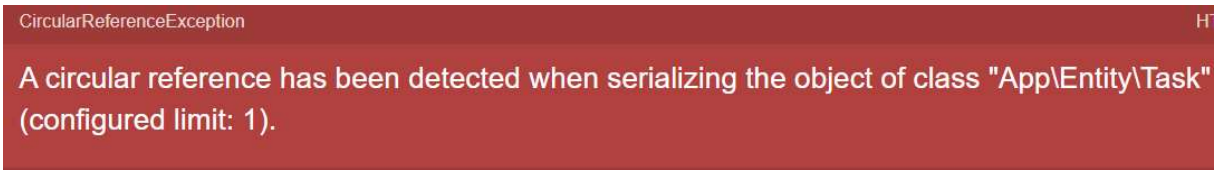
Nous allons modifier la fonction index de **TaskController.php** comme ci-dessous :

```
<?php

namespace App\Controller;

use App\Entity\Task;

use App\Repository\CatRepository;

use App\Repository\TaskRepository;

use App\Repository\UserRepository;

use Doctrine\ORM\EntityManagerInterface;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

use Symfony\Component\Serializer\SerializerInterface;

use Symfony\Component\Serializer\Normalizer\NormalizerInterface;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;


class TaskController extends AbstractController

{

    // je rajoute à la route la méthode associée GET

    #[Route('/task', name: 'app_task_index', methods: 'GET')]
```

SYMFONY

```
    public function index(
        TaskRepository $taskRepository,
        NormalizerInterface $normalizer
    ): Response {
        //récupération de toutes les taches
        $tasks = $taskRepository->findAll();
        //stockage des tasks passées dans la méthode normalize
        $tasksNormalize = $normalizer->normalize($tasks);
        dd($tasksNormalize);
    }
}
```

Nous allons nous retrouver avec une erreur circulaire. Cette exception est appelée **circular reference** (référence circulaire). Une **référence circulaire** c'est quoi ? C'est tout simplement à cause de la relation entre task et cat. Une tache fais référence à une catégorie qui fait référence à une tâche etc... Ils bouclent sur eux-mêmes.



Nous allons corriger cette erreur en utilisant les groupes :

Nous allons éditer le fichier **Task** (**entity**) comme ci-dessous en groupant à l'aide des annotations tous les attributs (or **users** et **cats**) comme ci-dessous :

SYMFONY

```
<?php
namespace App\Entity;
use App\Repository\TaskRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;
#[ORM\Entity(repositoryClass: TaskRepository::class)]
class Task
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    #[Groups('task:readAll')]
    private $id;
    #[ORM\Column(type: 'string', length: 255)]
    #[Groups('task:readAll')]
    private $name_task;
    #[ORM\Column(type: 'text')]
    #[Groups('task:readAll')]
    private $content_task;
    #[ORM\Column(type: 'datetime')]
    #[Groups('task:readAll')]
    private $date_task;
```

Nous allons ensuite éditer la fonction **index** de **TaskController.php** comme ci-dessous :

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

SYMFONY

```php
<?php
namespace App\Controller;
use App\Entity\Task;
use App\Repository\CatRepository;
use App\Repository\TaskRepository;
use App\Repository\UserRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Component\Serializer\Normalizer\NormalizerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
class TaskController extends AbstractController
{
    // je rajoute à la route la méthode associée GET
    #[Route('/task', name: 'app_task_index', methods: 'GET')]
    public function index(
        TaskRepository $taskRepository
    ): Response {
        return $this->json($taskRepository->findAll(),200, [], ['groups' =>
'task:readAll']);
    }
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			