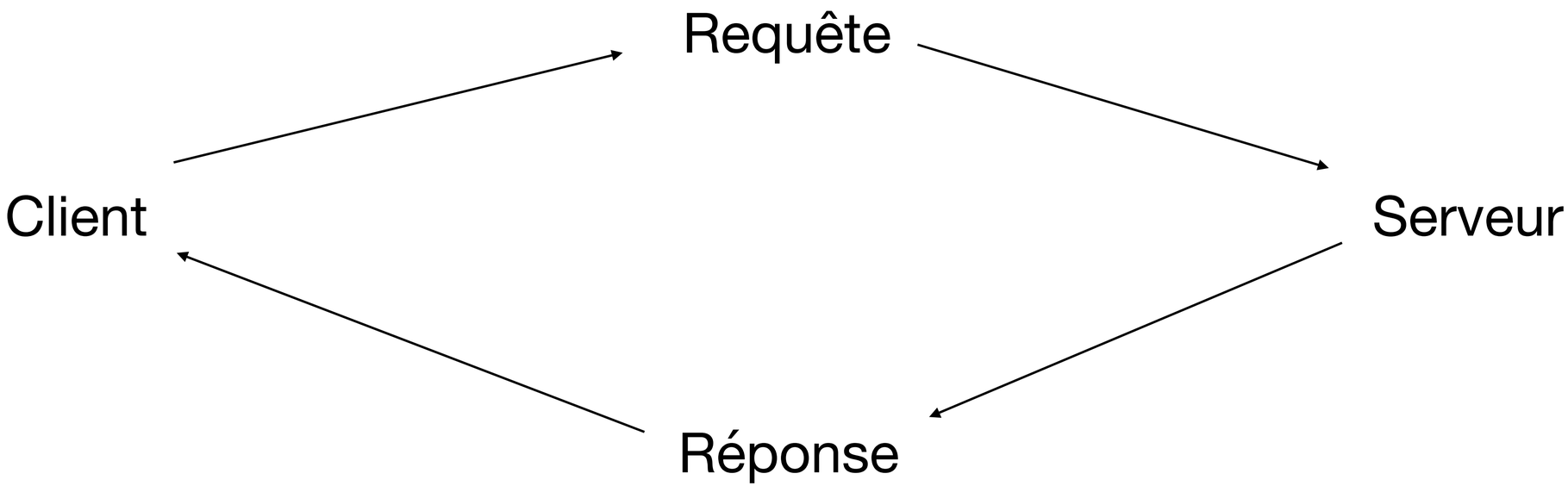
Symfony 5

EPSI/WIS - 2021
Hery DOMENICHINI - hery@meliway.fr

Pré-requis

- PHP 7 et POO
- Base de données : MCD/MLD et requêtes SQL
- Fonctionnement des requêtes HTTP
- Serveur local
- Quelques commandes de base pour le terminal

Requêtes HTTP



Présentation

Symfony : framework PHP modulaire

Design pattern : MVC

LTS : version 5.3

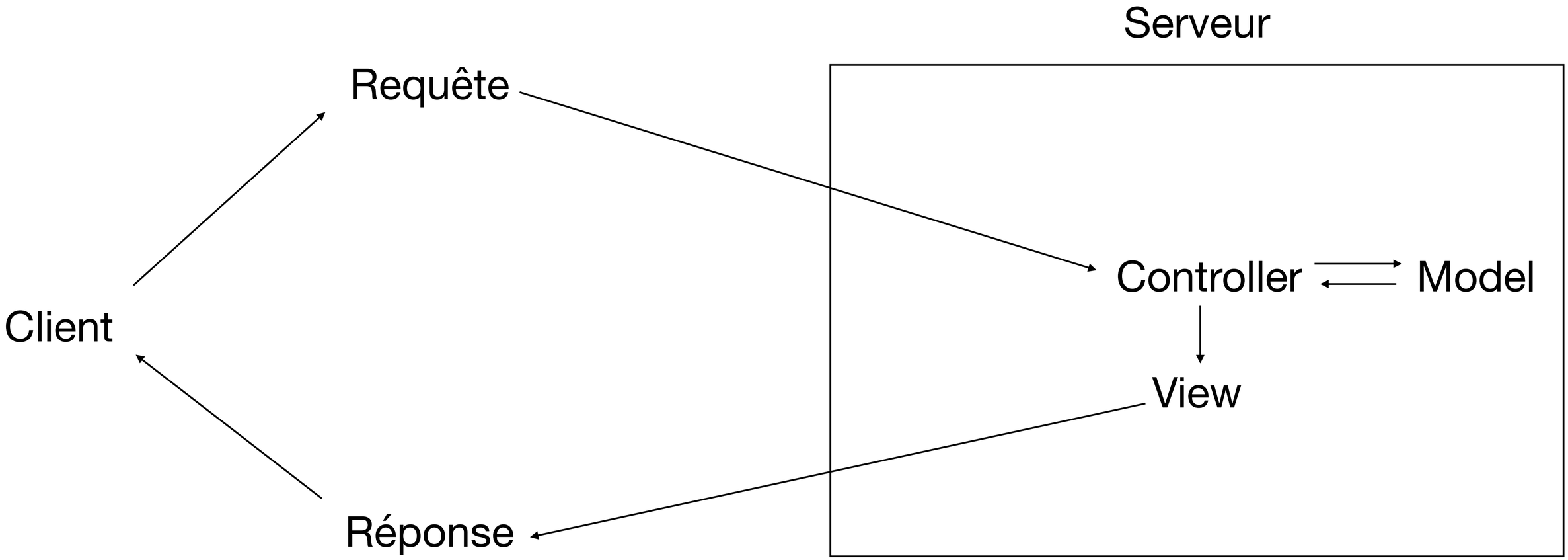
Symfony - MVC

Model : gère les interactions avec la base de données

View : gère l'affichage pour l'utilisateur

Controller : interagit entre le Model et la View

Symfony - MVC



Symfony 5

Installation

EPSI/WIS - 2021

Symfony - installation

Installer Composer = gestionnaire de dépendances

(une fois par ordinateur)

Pour vérifier si composer est installé : " php composer.phar -v"
ou "composer -v"

Symfony - installation

Créer un projet Symfony via le terminal :

1/ Se situer dans le dossier des projets du serveur local :

composer create-project symfony/website-skeleton:"5.3.x@dev" new-project

2/ Configuration d'Apache, **après s'être situé dans le projet** :

composer require symfony/apache-pack

Cela va générer un fichier htaccess (si ce n'est pas le cas, il faudra le créer)

3/ Installation des annotations :

composer require annotation

4/ Installation de Doctrine :

composer require symfony/orm-pack

Ajout des lignes de commande :

composer require --dev symfony/maker-bundle

Symfony 5

Architecture & Routing

EPSI/WIS - 2021

Symfony - architecture

Contenu des dossiers/fichiers de Symfony :

- dossier bin : contient les lignes de commandes
- dossier config : configuration des dépendances et de Symfony
- **dossier public** : fichiers disponibles pour le navigateur
- **dossier src** : coeur de l'application, fichiers qui font fonctionner le site
- **dossier templates** : les fichiers Twig pour les vues
- dossier tests : contient les scripts à coder pour les tests unitaires
- dossier translations : contient les fichiers de traduction
- dossier var : contient les logs (messages d'erreur) et les fichiers de caches
- dossier vendor : contient les dépendances installées (ne jamais modifier !!)
- fichier .env : contient les informations sensibles comme celles de connexion à la base de données
- fichier .gitignore : liste les fichiers devant être ignorés par Git
- fichier composer.json : liste les dépendances nécessaire au fonctionnement de l'application, ainsi que leur version minimum
- fichier composer.lock : liste les dépendances avec leur version actuellement chargée
- fichier phpunit.xml.dist : permet de configurer les paramètres des tests unitaires
- fichier symfony.lock : contient les dépendances de Symfony

Symfony - routing

Une **route** permet de définir une URL, qui est composée de :

- un protocole : http ou https
- un nom de domaine : www.example.com
- un chemin : spécifie le contenu auquel on accède via une arborescence : www.example.com/**blog**
- des paramètres éventuels : www.example.com/blog/**?page=1**

A partir de la **route** (**requête**) demandée par le client, le système de routing de Symfony va rechercher la partie du code du **Controller** qui correspond à la route.

Puis celui-ci va faire le traitement nécessaire pour renvoyer une **réponse** (souvent sous forme d'affichage d'une vue).

Symfony 5

Controller

EPSI/WIS - 2021

Symfony - Controller

```php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController as AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;

/**
 * Description of HomeController
 *
 * @author lapiscine
 */
class HomeController extends AbstractController {
    /**
     * @Route("/", name="home")
     */
    public function home()
    {
        return $this->render("home.html.twig");
    }
}
```

Note : le fichier contenant cette classe doit avoir le même nom que la classe en elle-même (ici "HomeController")

Symfony - Controller

```php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController as AbstractController;
use Symfony\Component\Routing\Annotation\Route;
```

Le **namespace** est l'emplacement du fichier dans l'application

Les **use** sont les liens vers des fichiers nécessaires à l'utilisation de certaines fonctions via des classes

Symfony - Controller

```php
/**
 * @Route("/", name="home")
 */
public function home()
{
    return $this->render("home.html.twig");
}
```

La **route** est en annotation et nécessite deux paramètres : le chemin et le nom

La méthode qui suit sera toujours **public** et aura un nom unique.
Cette méthode peut prendre des paramètres.

Particularité de Symfony : pour tous les services, passer une classe en paramètre d'une méthode l'instancie automatiquement et la stocke dans la variable indiquée.

Exemple : `public function home(Request $request)`

Ici, la variable $request contient l'instance de la classe Request.

Attention à bien mettre le "use" nécessaire pour utiliser la classe : `use Symfony\Component\HttpFoundation\Request;`

Symfony - Controller

```php
/**
 * @Route("/", name="home")
 */
public function home()
{
    return $this->render("home.html.twig");
}
```

Les réponses valides dans Symfony sont :

- la classe Response du composant HttpFoundation :
  return new Response("message");

- une redirection : **return $this->redirectToRoute("nom_de_la_route");**

- en retournant un fichier twig : **return $this->render("home.html.twig");**

Symfony 5

TP

EPSI/WIS - 2021

Symfony - TP

Objectif : créer ses premières routes

1/ Créer un fichier (PHP class) dans le dossier "Controller"

2/ Ajouter l'extension "AbstractController" à la nouvelle classe

3/ Créer une méthode : (ex : *public function contact( ))*

4/ Y associer sa route en annotation (ne pas oublier le "use" de la Route)

5/ Coder le contenu de la méthode en n'oubliant pas de renvoyer une réponse

6/ Si la réponse est un fichier Twig, il faut le créer dans le dossier "templates"

7/ Créer une deuxième route qui redirige vers la première

1 route = 1 méthode = 1 réponse

Symfony 5

Twig

EPSI/WIS - 2021

Symfony - Twig

Twig : moteur de templates

Permet de simplifier et alléger l'écriture des vues en évitant PHP.

Bases de l'écriture de Twig :

- {{ variable }} : permet d'afficher le contenu de la variable
- {{ dump(variable) }} : affiche le dump de la variable
- {# commentaire #} : commentaire qui ne sera pas visible côté navigateur
- {% set variable = true %} : attribue une valeur à une variable

Symfony - Twig

Les variables :

- variable de type string, float, integer, bool :

  {% set nomDeLaVariable = *valeur* %}

- variable de type array :

  {% set array = ["php", "javascript", "java", "python"] %}

  {{ array[0] }} : affiche "php"

- variable de type array associatif (= objet) :
  {% set livre = { "titre":"Coder en PHP", "prix":"20"} %}

  {{ livre.titre }} : affiche "Coder en PHP"

Symfony - Twig

Les conditions :

```
{% if variable > 0 %}
   <p>La variable est supérieure à zéro</p>
{% elseif variable == 0 %}
   <p>La variable est égale à zéro</p>
{% else %}
   <p>La variable est inférieure à zéro</p>
{% endif %}
```

Symfony - Twig

Les boucles (*for "simple"*) :

```
{% for i in 0..10 %}
   <p>Le compteur est à {{ i }}</p>
{% endfor %}
```

Les boucles (*foreach)* :

```
{% for key, value in array %}
    <p>La clé {{ key }} a pour valeur {{ value }}</p>
{% endfor %}
```

Symfony - Twig

Les filtres

Permettent de transformer une valeur d'entrée

Exemple 1 : {{ birthDate|date('d m Y') }} : affiche la date au format indiqué

Note : le filtre est obligatoire pour afficher une date

Exemple 2 : {{ name|upper }} : affiche la valeur de la variable en majuscule

Symfony - Twig

Les balises "link"

Leur lien se fait avec la fonction twig "asset" :

<link href="{{ **asset**('assets/css/styles.css') }}">

Il s'agit d'un lien relatif avec pour point de départ le dossier "public"

Symfony - Twig

Les chemins

Les liens via les balises "a" se font avec la fonction twig "path" :

< a href="{{ **path**('*nomDeLaRoute*') }}" > Mon lien

Pour un lien vers un site externe, il faudra utiliser le lien absolu de celui-ci.

Les chemins avec paramètres

Pour transférer un paramètre en URL :

```
{{ path('nomDeLaRoute', {'id':42 }) }}
```

Exemple concret avec une boucle :

```
{% for article in array %}
    <a href="{{ path('article_read',{ 'id':article.id }) }}
{% endfor %}
```

Symfony - Twig

Récupérer des informations via l'URL

Deux types d'information : les paramètres et les wildcards

Symfony - Twig

Les paramètres

L'URL sera de la forme : www.example.com/blog?id="42"

On utilise la classe Request.

```
$id = $request->query->get('nomDuParamètre');
```

Symfony - Twig

Les wildcards

L'URL sera de la forme : www.example.com/blog/42

C'est un élément qui doit être prévu dès l'écriture du chemin de la route :

```php
/**
 * @Route("/blog/{id}", name="blog")
 */
public function blog($id)
{

    return $this->render('blog.html.twig');
}
```

Ainsi, la récupération de la wildcard se fait en indiquant juste une variable en paramètre de la méthode, avec un nom identique à la wildcard.

A noter que la wildcard devient obligatoire.

Symfony - Twig

Il est possible de transmettre une variable du Controller vers le Twig :

```php
/**
 * @Route("/multi/{nb}/{multiplicateur}", name="multi")
 */
public function division($nb, $multiplicateur)
{
    $result = $nb * $multiplicateur;

    return $this->render( view: 'multi.html.twig',[
        'result' => $result
    ]);
}
```

Controller

```twig
<p>Réslutat de la multiplication : {{ result }}</p>
```

Twig

Résultat :

Réslutat de la multiplication : 4

Symfony - Twig

Il est possible de transmettre une variable du Controller vers le Twig :

```php
/**
 * @Route("/multi/{nb}/{multiplicateur}", name="multi")
 */
public function division($nb, $multiplicateur)
{
    $result = $nb * $multiplicateur;

    return $this->render( view: 'multi.html.twig',[
        'result' => $result
    ]);
}
```

Les variables transmises se font à travers un array, comme 2ème paramètre de la méthode "render".

La clé de cet array sera le nom de la variable côté Twig.

Symfony - Twig

Les blocs

2 fichiers :

- base.html.twig : contient les parties répétitives d'une page à l'autre

  {% block content %} Affichage par défaut {% endblock %}

- home.html.twig : contient la partie variable et spécifique de cette page

  {% block content %} Affichage si défini {% endblock %}

Symfony - Twig

```twig
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Bienvenue</title>
    </head>
    <body>
        <header>
            Header
        </header>

        <main>

            {% block content %}
                Affichage par défaut
            {% endblock %}

        </main>

        <footer>
            Footer
        </footer>
    </body>
</html>
```

base.html.twig

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

{% block content %}

    Affichage pour la page "home"

{% endblock %}
```

home.html.twig

Symfony 5

TP

EPSI/WIS - 2021

Symfony - TP

Objectif : créer un mini-site vitrine avec :
- un header et un footer commun aux différentes pages
- une page d'accueil
- une page "À propos"
- une page qui affiche un article (titre - texte - auteur) selon l'ID dans l'URL

Indication : pour les articles, puisqu'il n'y a pas de base de données, utilisez une méthode avec un array :

Pour utiliser cette méthode dans vos contrôleurs, procédez ainsi :
$article = $this->getArticles($id);

Où $id doit être récupéré via l'URL (paramètre ou wildcard)

```php
public function getArticles(int $id)
{
    $array = [
        0 => [
            'titre' => 'Mon 1er article',
            'texte' => 'lorem ipsum',
            'auteur' => 'John DOE'
        ],
        1 => [
            'titre' => 'Mon 2ème article',
            'texte' => 'lorem blabla',
            'auteur' => 'Jane DOE'
        ]
    ];

    return $array[$id];
}
```

Symfony 5

La base de données (BDD)

EPSI/WIS - 2021

Symfony - BDD

En PHP : instanciation de PDO

Dans Symfony : pas d'instanciation de PDO, c'est automatique

Doctrine : un ORM qui va nous aider à interagir avec la base de données

ORM = Object Relational Mapping
=> permet de transformer une base de données relationnelle en base de données orientées objet + évite les requêtes SQL brutes

Symfony - BDD

Paramétrage du fichier .env :

1/ Remplacer la ligne "DATABASE_URL":

DATABASE_URL=mysql://login:password@localhost:8889/nameDB

ou DATABASE_URL=mysql://login:password@127.0.0.1:8889/nameDB

2/ Création de la base de données via le terminal :

php bin/console doctrine:database:create

Symfony - BDD

Vocabulaire :

- le terme "table" est utilisé quand on parle de la base de données
- le terme "entité" est utilisé quand on parle de l'objet dans Symfony

Dans les faits, une table (côté BDD) correspond à une entité (côté Symfony)

Symfony - BDD

Création d'une entité via le terminal en se situant à la racine du projet :

1/ Ligne de commande : **php bin/console make:entity**

2/ Renseigner le nom de l'entité à modifier ou créer

3/ Renseigner le nom d'une propriété, son type etc

4/ Indiquer si le champ peut être nul

Cela va créer deux fichiers

Symfony - BDD

Dans le dossier "Entity"

```php
namespace App\Entity;

use App\Repository\ArticleRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ArticleRepository::class)
 */
class Article
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $title;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getTitle(): ?string
    {
        return $this->title;
    }

    public function setTitle(?string $title): self
    {
        $this->title = $title;

        return $this;
    }
}
```

Symfony - BDD

Dans le dossier "Repository" = pour faire des requêtes de sélection à la BDD

```php
namespace App\Repository;

use App\Entity\Article;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @method Article|null find($id, $lockMode = null, $lockVersion = null)
 * @method Article|null findOneBy(array $criteria, array $orderBy = null)
 * @method Article[]    findAll()
 * @method Article[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class ArticleRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Article::class);
    }
```

Symfony - BDD

Mettre à jour la base de données, toujours via le terminal :

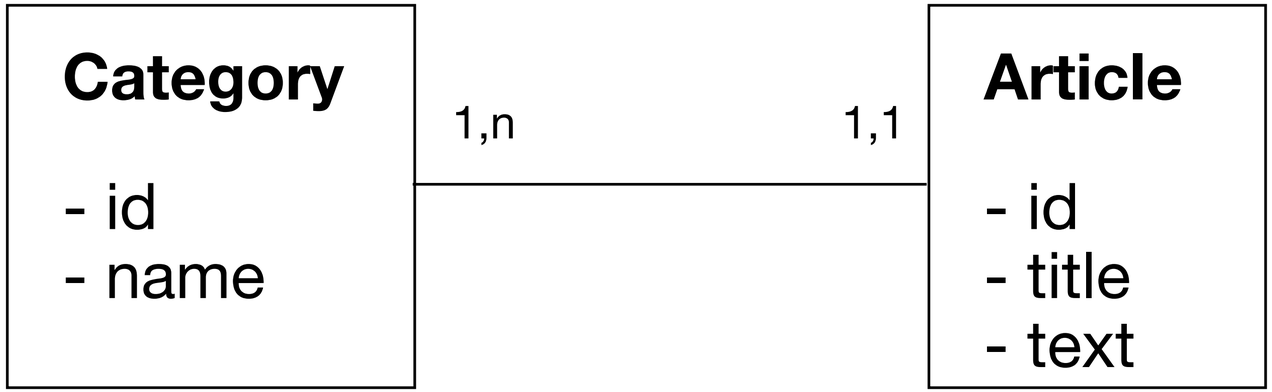
1/ Commande : **php bin/console make:migration**
Cela va créer un fichier "version" dans le dossier "migration"

2/ Commande : **php bin/console doctrine:migrations:migrate**
Puis valider avec "**y**"
Cela va mettre à jour la structure de la base de données

Symfony - BDD

Cas particulier : les champs relationnels

MCD :



Symfony - BDD

Même procédure, mais en sélectionnant le type "relation",
puis sélectionner la classe à mettre en relation,
le type de relation,
si la propriété peut être nulle (oui par défaut),
si il faut créer une propriété dans l'autre entité (oui par défaut),
le choix du nom de la propriété (nom de l'entité par défaut)

```php
/**
 * @ORM\ManyToOne(targetEntity=Category::class, inversedBy="articles")
 */
private $category;
```

```php
public function getCategory(): ?Category
{
    return $this->category;
}

public function setCategory(?Category $category): self
{
    $this->category = $category;

    return $this;
}
```

Symfony 5

Repository

EPSI/WIS - 2021

Symfony - Repository

Rappel : permet de gérer la sélection dans la base de données

```php
namespace App\Repository;

use App\Entity\Article;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @method Article|null find($id, $lockMode = null, $lockVersion = null)
 * @method Article|null findOneBy(array $criteria, array $orderBy = null)
 * @method Article[]    findAll()
 * @method Article[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class ArticleRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Article::class);
    }
```

Symfony - Repository

Les méthodes de base :

$repository->**findAll( )** : sélectionne toutes les données d'une table

$repository->**findBy(['key'=>'value'], ['id'=>'DESC'])** : renvoie un array
Sélectionne les données selon un critère (2ème paramètre : critère de tri)

$repository->**findOneBy(['key'=>'value'], ['id'=>'DESC'])** : ne renvoie qu'une entité

$repository->**find($id)** : sélectionne une entité selon l'ID

Symfony - Repository

Pour créer sa propre méthode de sélection :

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

    public function personnalSelect($name)
    {
        return $this->createQueryBuilder('a')
                    ->select('a')
                    ->leftJoin('a.category', 'c')
                    ->addSelect('c')
                    ->where('c.name LIKE :name')
                    ->setParameter('name', '%'.$name.'%')
                    ->getQuery()
                    ->getResult();
    }
```

Symfony 5

CRUD

EPSI/WIS - 2021

Symfony - CRUD

CRUD : toutes les actions réalisables avec une base de données

Create - Read - Update - Delete

Symfony - Create

Objectif : créer un nouvel objet à partir d'une classe du dossier "Entity"

```php
use App\Entity\Article;
use Doctrine\ORM\EntityManagerInterface;
```

```php
/**
 * @Route("/create", name="create")
 */
public function createArticle(EntityManagerInterface $en)
{
    $article = new Article;
    $article->setTitle("Mon véritable premier article");

    $en->persist($article);
    $en->flush();

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

}
```

Symfony - Read

Objectif : afficher toutes les entités

```php
use App\Repository\ArticleRepository;
```

```php
/**
 * @Route("/blog", name="blog")
 */
public function blog(ArticleRepository $repository)
{


    $articles = $repository->findAll();

    return $this->render('blog.html.twig',[
        'articles' => $articles
    ]);

}
```

Twig

```twig
{% for article in articles %}

    <div>
        <h1>{{ article.title }}</h1>
    </div>

{% endfor %}
```

Symfony - Read

Objectif : afficher une entité particulière

```php
use App\Repository\ArticleRepository;
```

```php
/**
 * @Route("/article/{id}", name="article")
 */
public function readArticle($id, ArticleRepository $repository)
{
    $article = $repository->find($id);

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

Twig

```twig
<h1>{{ article.title }}</h1>
```

Symfony - Update

Objectif : mettre à jour une entité particulière

```php
use Doctrine\ORM\EntityManagerInterface;
use App\Repository\ArticleRepository;
```

```php
/**
 * @Route("/update/{id}", name="update")
 */
public function updateArticle($id, ArticleRepository $repository, EntityManagerInterface $en)
{
    $article = $repository->find($id);

    $article->setTitle('Nouveau titre');

    $en->persist($article);
    $en->flush($en);

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

}
```

Symfony - Delete

Objectif : supprimer une entité particulière

```php
use Doctrine\ORM\EntityManagerInterface;
use App\Repository\ArticleRepository;
```

```php
/**
 *
 * @Route("/delete", name="delete")
 */
public function deleteArticle($id, ArticleRepository $repository, EntityManagerInterface $en)
{
    $article = $repository->find($id);

    $en->remove($article);
    $en->flush();

    return $this->redirectToRoute('blog');
}
```

Symfony 5

TP

EPSI/WIS - 2021

Symfony - TP

Objectif : créer un CRUD pour les livres, en liant les différentes pages entre elles

Préalable : les entités à créer :

- book : title, text, author, categoryBook (champ ManyToOne), tag (champ ManyToMany)
- categoryBook : name, book (champ OneToMany)
- tag : name, book (champ ManyToMany)

Symfony 5

Les formulaires 1/2

EPSI/WIS - 2021

Symfony - les formulaires

Création d'un formulaire en lignes de commande :

1/ Commande **: php bin/console make:form**

2/ Renseigner le nom du fichier (convention : celui-ci doit finir par "...Type")
Exemple : ArticleCreateType

3/ Renseigner le nom de l'entité qui doit être liée au formulaire

Symfony - les formulaires

Résultat : création d'un fichier dans le dossier "Form"

```php
namespace App\Form;

use App\Entity\Article;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ArticleCreateType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
        ;
    }

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

Symfony - les formulaires

Utilisation du formulaire dans le Controller :

```php
use App\Form\ArticleCreateType;
```

```php
/**
 * @Route("/create", name="create")
 */
public function createArticle(EntityManagerInterface $en)
{
    $article = new Article;


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


    return $this->render('create.html.twig',[
        'form' => $form->createView()
    ]);

}
```

Symfony - les formulaires

Utilisation du formulaire en Twig:

```
{% block content %}

    {{ form(form) }}

{% endblock %}
```

Symfony - les formulaires

Utilisation du formulaire en Twig:

```twig
{% block content %}

    {{ form_start(form) }}

        {{ form_label(form.title,"Titre") }}

        {{ form_widget(form.title) }}

        <button type="submit">Valider</button>

    {{ form_end(form) }}

{% endblock %}
```

Symfony - les formulaires

Traitement du formulaire (dans le Controller) :

```php
use Symfony\Component\HttpFoundation\Request;
```

```php
$article = new Article;


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

$form->handleRequest($request);

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

    $en->persist($article);
    $en->flush();

    return $this->redirectToRoute('blog');
}


return $this->render('create.html.twig',[
    'form' => $form->createView()
]);
```

Symfony 5

TP

EPSI/WIS - 2021

Symfony - TP

Objectif : sur le CRUD de l'entité "book" du TP précédent, y intégrer un formulaire

Aide : dans un premier temps, ignorer les propriétés "categoryBook" et "tag" de l'entité Book, et leur donner une valeur par défaut via les setters

Symfony 5

Les formulaires 2/2

EPSI/WIS - 2021

Symfony - Retour sur les FormType

Paramétrage pour le bouton de soumission :

```php
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
```

```php
->add( child: 'Modifier', type: SubmitType::class)
```

Symfony - Retour sur les FormType

Paramétrages généraux :
1/ Spécifier le type
2/ Dans un array, mettre les paramètres

```php
use Symfony\Component\Form\Extension\Core\Type\TextType;
```

```php
->add( child: 'titre', type: TextType::class,[
    'required' => true,
    'empty_data' => 'Mon titre'
])
```

Remarque : "placeholder" est utilisable de la même manière

Symfony - Retour sur les FormType

Paramétrage pour les dates :

```php
use Symfony\Component\Form\Extension\Core\Type\DateType;
```

```php
->add( child: 'date',  type: DateType::class,[
    'widget' => 'single_text',
    'required' => false
])
```

Symfony - Retour sur les FormType

Paramétrage pour les entités :

```php
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
```

```php
->add( child: 'category', type: EntityType::class,[
    'class' => ArticleCategory::class,
    'choice_label' => 'name'
])
```

Symfony - Retour sur les FormType

Paramétrage pour les choix multiples + entité :

```php
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
```

```php
->add( child: 'tag', type: EntityType::class,[
    'class' => Tag::class,
    'multiple' => true,
    'expanded' => false,
    'required' => false,
    'choice_label' => 'name'
])
```

Symfony - Retour sur les FormType

Paramétrage pour un "select", sans entité :

```php
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
```

```php
->add( child: 'ville',  type: ChoiceType::class,[
    'choices' => [
        'bordeaux' => 'Bordeaux',
        'paris' => 'Paris',
        'montpellier' => 'Montpellier'
    ],
    'label' => 'Choix de la ville',
    'required' => false
])
```

Symfony - Retour sur Twig

Ajouter des classes :

```
{{ form_label(form.title,"Titre",{"label_attr":{"class":"my-class"}}) }}

{{ form_widget(form.titlre,{"attr":{"class":"my-class"}}) }}
```

Symfony 5

Les images

EPSI/WIS - 2021

Symfony - Les images

Dans le fichier config/services.yaml :

```yaml
parameters:
    app.path.general_images: /
    profile_images: '%kernel.project_dir%/public/assets/img'
```

Symfony - Les images

Dans le FormType :

```php
use Symfony\Component\Validator\Constraints\File;
```

```php
->add( child: 'picture', type: FileType::class,[
    'mapped' => false,
    'constraints' => [
        new File([
            'maxSize' => '3M',
            'mimeTypes' => [
                'image/png',
                'image/jpeg'
            ]
        ])

    ]
])
```

Symfony - Les images

```php
$form->handleRequest($request);

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

    $imageFile = $form['picture']->getData();

    if ($imageFile) {
        $originalFilename = pathinfo($imageFile->getClientOriginalName(), options: PATHINFO_FILENAME);
        $safeFilename = transliterator_transliterate( transliterator: 'Any-Latin; Latin-ASCII; [^A-Za-z0-9_] remove; Lower()', $originalFilename);
        $newFilename = $safeFilename . '-' . uniqid() . '.' . $imageFile->guessExtension();


        try {
            $move = $imageFile->move(
                $this->getParameter( name: 'profile_images'),
                $newFilename
            );
            if(!$move){
                throw new FileException( message: 'Erreur lors du chargement de l\'image');
            }

        } catch (FileException $e) {
            echo 'Erreur reçue : '.$e->getMessage();
        }

        $article->setPicture($newFilename);


    }
}
```

Symfony - Les images

```
<img src="{{ asset('assets//img/'~article.picture) }}" alt="{{ article.pictureAlt }}">
```

Symfony 5

Conseils bonus

EPSI/WIS - 2021

Symfony - Bonus

Pour vérifier toutes les routes existantes, dans le terminal : php bin/console debug:router

Pour connaître toutes les commandes utiles : php bin/console

Espace admin : https://symfony.com/doc/current/security.html

Symfony - Bonus

Pour renforcer la sécurité du site : toujours vérifier les informations provenant du client (paramètres d'URL, wildcards, données envoyées via formulaire, cookies etc)

Règle d'or : **ne jamais faire confiance à l'utilisateur !!**

Symfony - Bonus

Pour l'envoi de mails : Swift Mailer
https://symfony.com/doc/current/email.html?utm_source=recordnotfound.com

Créer ses propres services
https://symfony.com/doc/current/service_container.html

Symfony 5

TP FINAL

EPSI/WIS - 2021

Symfony - TP

Objectif : créer un mini-annuaire professionnel (3 pages) et un espace admin pour le CRUD d'une société
MCD :
- société : nom de la société, *date de création*, image principale (picture), description,
- catégorie : nom
Une société est liée à une catégorie, une catégorie est liée à plusieurs sociétés.

1ère page : la page principale de l'annuaire doit contenir :
- header :
    - des liens vers les différentes catégories de société (cf 2ème page)
    - un système de recherche de société par son nom
- une liste de toutes les sociétés, avec : nom de la société, une image principale + lien vers une page détaillant la société

2ème page : la page d'une catégorie doit contenir :
- header :
    - des liens vers les différentes catégories de société
    - un système de recherche de société par son nom
- la liste des sociétés de la catégorie (titre + image)

3ème page : la page d'une société doit contenir :
- header :
    - des liens vers les différentes catégories de société
    - un système de recherche de société par son nom
- le nom de la société
- l'image principale
- la date de création
- la description de la société
- un lien retour vers la page principale de l'annuaire

Date de rendu : avant le 15 octobre - 23h59 - fichier ZIP sur myLearningBox ou Github

Symfony - TP

Précisions :

- pour la société, la date de création doit être renseignée automatiquement selon la date du jour, lors de la création d'une société.

- le système de recherche doit renvoyer vers la 1ère page en filtrant les résultats

- aucun retard ne sera toléré

- le rendu se fera via myLearningBox : soit un dépôt Github, soit le projet compressé (ZIP)
  Note : pour alléger le projet, vous pouvez ne pas envoyer les dossiers "var" et "vendors"