

SYMPHONY

Bandiougou BOUARÉ

bandiougoubouare@gmail.com

07 83 41 90 68

FRAMEWORK

Un Framework est une **boîte à outils** pour un développeur web. Frame signifie *cadre* et work se traduit par *travail*. Un Framework contient des composants autonomes qui permettent de faciliter le développement d'un site web ou d'une application. Ces composants résolvent des problèmes souvent rencontrés par les développeurs (CRUD, arborescence, normes, sécurités, etc.). Ils permettent donc de gagner du temps lors du développement du site.

QUEL INTÉRÊT À UTILISER UN FRAMEWORK POUR UN PROJET WEB ?

L'intérêt à utiliser un Framework lors du développement de votre projet web se situe à plusieurs niveaux :

- ❑ **Rapidité** : une base de travail existe déjà, donc le développeur web n'a pas besoin de partir de zéro pour créer votre site web.
- ❑ **Flexibilité** : vous pouvez choisir d'utiliser ou non certains composants du Framework pour améliorer le référencement naturel de votre site.
- ❑ **Architecture** : en utilisant un bon Framework, vous avez du code propre et fonctionnel qui ne ralentit pas le fonctionnement du site.
- ❑ **Productivité** : que ce soit un développement en solo ou en équipe, un Framework est un outil puissant puisque tout est parfaitement organisé.
- ❑ **Communauté** : vous bénéficiez de l'appui de toute une communauté en ligne (support et forum) qui vous aidera à corriger les bugs ou résoudre des problèmes de programmation.

QUELQUES FRAMEWORKS



IL ETAIT UNE FOIS ... SYMFONY

- **Symfony est un puissant Framework** qui permet de réaliser des sites complexes rapidement, mais de façon structurée et avec un code clair et maintenable. En un mot : le paradis du développeur !
- Développé par [SensioLabs](#), la première de symfony commença en 2005.
- Cette première version était surtout une collection de librairie PHP et quelques méthodes, une architecture assez simple et pas suffisamment normée. Manque de conventions.

LES VERSION SYMFONY

- Symfony2: arrivée ce juillet 2011 avec (L'injection de dépendances, tout est un Bundle dans Symfony, Gestion de la sécurité)
- Symfony 3: novembre 2015 : (On considère la version 3 comme une version 2.8 sans les couches dépréciés)
- Symfony 4: novembre 2017 : (Symfony 4.0 = Symfony 3.0 + Flex et composer)

SYMFONY 5

- Le 21 novembre 2019, marque la **sortie de Symfony 5**.
- La mise à jour contient toutes les [fonctionnalités prévues dans Symfony 4.4](#), et des fonctionnalités expérimentales
- La [série d'articles sur Symfony 4](#) sera mise à jour progressivement avec des précisions sur les différences pour les utilisateurs de Symfony 5.
- L'injection de dépendances
- Mailer et Notifier

SYMFONY 6

Sortie en Novembre 2021, aujourd'hui on n'est la version 6.3.5

- Prise en charge de Bootstrap 5 et Tailwind pour le rendu des formulaires.
- Gestionnaires de messagerie configurables par le biais d'attributs, sans avoir à les ajouter à la configuration.
- Consommation par lot des messages du messenger via l'interface `BatchHandlerInterface`.
- **Uid** pour la gestion des identifiants uniques.
- **RateLimiter** qui permet de limiter le nombre de tentatives de connexion avec un mot de passe erroné.
- **PasswordHasher**, fonctionnalité sur la couche de sécurité et d'authentification (sous-système extrait du composant Security).
- **Translation Providers** qui fournit une interface permettant d'intégrer des services de traduction en ligne.
- **Runtime** pour rendre plus flexible la séquence de démarrage des applications PHP et donc des applications Symfony.

COMPOSER

- Composer est un Gestionnaire de dépendance
- Il gère les dépendances qu'on lui demande de gérer via un fichier `composer.json`
- Composer n'est pas un gestionnaire de paquets comme Yum ou Apt. Il ne gère les dépendances que localement et non globalement comme le ferait un gestionnaire de paquets
- En plus de gérer les dépendances, il s'occupe aussi de faire l'autoload pour nous
- Pour avoir un système d'autoload gratuitement, il nous suffira de faire un import du fichier autoload de composer qu'il place dans le dossier `vendor/autoload.php`

LES MÉTHODES HTTP

- Ce qui nous intéresse dans les méthodes HTTP
- Méthodes:
 - GET
 - POST
 - PUT/PATCH
 - DELETE

API REST

- Le REST signifie “Representational State Transfer” est un style d’architecture qui repose sur le protocole HTTP créer en 2000 par Roy Fielding. [Source de données](#) fiable.
- Le REST est bien une architecture et non une technologie. Il impose les contraintes suivantes:
 - la séparation entre le client et le serveur
 - La gestion du cache
 - Une interface uniforme
 - Un système hiérarchisé par couche
 - Code-on-demand (facultatif)
- A noter qu’il existe une autre architecture appelé [SOAP](#) de moins en moins utilisé mais qui reste tout de même une référence.

TWIG

- Twig est un moteur de template en PHP fait par Sensiolabs
- Il facilite le développement front end et est plus simple à apprendre.
- L'un des objectifs de Sensiolabs était de faire un moteur de template avec une syntaxe qu'on retrouve souvent dans d'autres langages de sorte que les développeurs front-end puissent développer sans avoir à connaître le PHP
- Il existe d'autres moteurs de template comme Blade, Mustache ou encore Smarty
- A noter:
 - Blade est le moteur de template de Laravel
 - Mustache est un moteur de template logic-less et portable (utilisable dans presque tous les langages)
 - Smarty, un moteur de template très répandu et réputé plus performant que Twig (dans un comparatif fait en 2011... donc à revoir car depuis twig a beaucoup évolué)
- Doc et site officiel : <http://twig.sensiolabs.org/>

SYMFONY 6 – INSTALLATION SYSTÈME

Pour votre environnement de développement et production vous devez:

- Installer [installer Symfony CLI](#) ([avec scoop](#))
- Avoir au moins PHP 8.1 installer ([WampServer](#))
- Installer Composer [Install Composer](#)
- Git (Recommandé)
- Installer [visual studio code](#)

La commande suivante permet si votre ordinateur répond à toutes les exigences d'utilisation de symfony `symfony check:requirements`

VS CODE

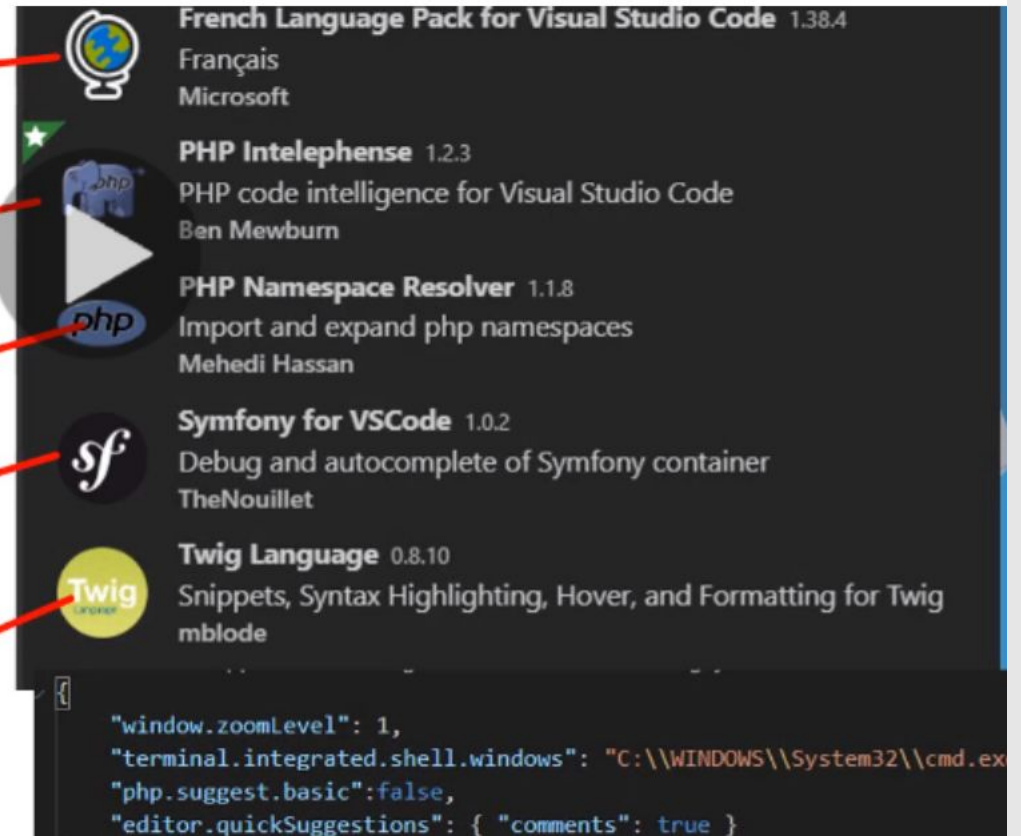
Mettre Visual Studio Code en Français

Autocomplétion php (et d'autres fonctionnalités)

Faciliter l'importation des espaces de noms

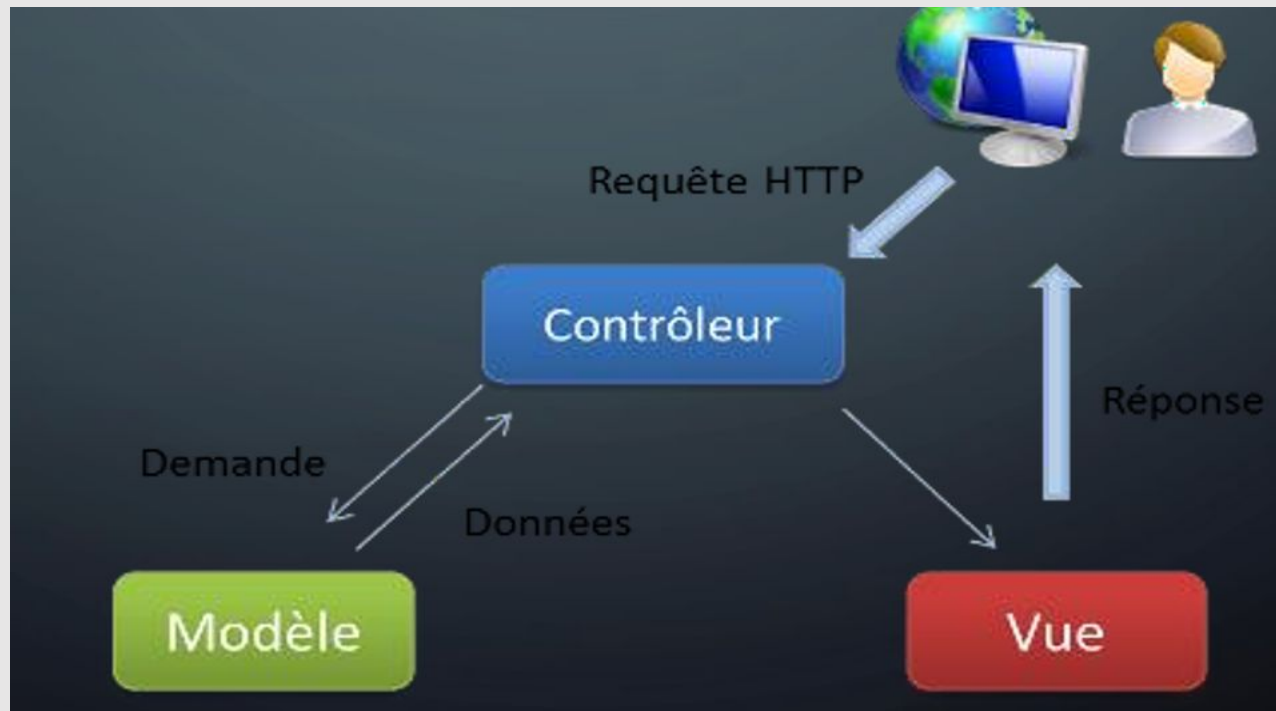
Autocomplétion symfony

Permet à Visual Studio Code de comprendre la syntaxe de Twig



MVC

MVC est un patron de conception (*design pattern* en anglais) très répandu pour réaliser des sites web. Ce patron de conception est une solution éprouvée et reconnue permettant de séparer l'affichage des informations, les actions de l'utilisateur et l'accès aux données.



Développement d'un site (blog)

Dans cette partie de la formation nous allons voir les différentes fonctionnalités de base que propose symfony à travers le développement d'un blog

Création du projet Symfony

- Run dans le terminal `composer create-project symfony/skeleton blog`
(Commande pour créer un projet symfony Basic)
- Installation de notre premier bundle
`composer require --dev symfony/maker-bundle`



Welcome to Symfony 6.3.5

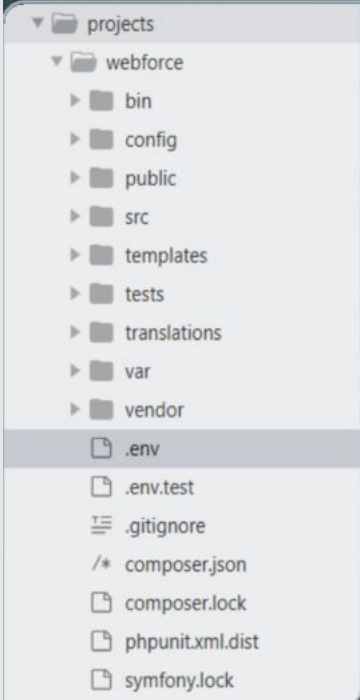


C:\Users\torre\Desktop\Formations\test\

Your application is now ready and you can start working on it.

SYMFONY 6 - ARCHITECTURE

SYMFONY 5 - ARCHITECTURE



- Le dossier bin/ contiens les exécutables
- Le dossier config/ contient les fichiers de config
- Le dossier public contient le fichier dans lequel vous devez ajouter les Bundles que vous créer ou importer afin que Symfony puisse les trouver quand il en a besoin
- Le dossier src/ contiens le code que vous allez développer. C'est dans ce dossier que vous passerez 95% de votre temps
- Le dossier Templates lui contient des layouts avec une haute priorité d'exécution. (On verra ce que ça veut dire)
- Le dossier vendor/ qui contient toutes librairie ou bundle télécharger via composer dont Symfony qui est un bundle
- Le dossier var/ lui contient les logs, caches et sessions

Chapitre 1 : Les notions de bases de symfony

Dans cette partie on va afficher notre <<Hello word>>

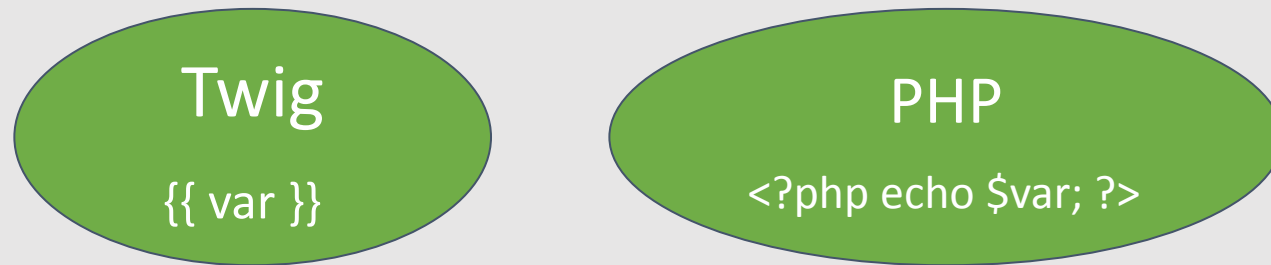
- **Les annotations** : les permettent de définir la redirection des routes pour l'installer : `composer req annotations`

- **Twig** : template est le moteur templating utilisé par symfony pour l'affichage des vues. (`composer req twig`)

- **Controller** : il permet de faire le traitement avant l'affichage à la vue. On crée notre premier controller avec (`symfony console make:controller home`)

Etape 1 : Les fonctionnalités de Twig

Différence [twig](#) et php



Le template twig permet de séparer du code php au code THML/XML/Text...

Le moteur twig offre son pseudo-langage à lui c'est du php mais c'est plus adapté.

Avec du twig nous allons pouvoir définir des variables, afficher des valeurs, faire des boucles sur des tableaux...

Etape 1 : Les fonctionnalités de Twig

- **Comment définir une variables en twig?:**

```
{% set myVariable = "Contenu variable en twig"%}  
{{ myVariable }}
```

- **Les boucles en twig:**

```
{% for user in users %}  
    {{ user }}  
{% endfor %}
```

Les tableaux en twig :

```
{% set users = ["Tom","Jerry","Alice"]%}  
    {% for user in users %}  
        {{ user | upper }}  
    {% endfor %}
```

Etape 1 : Les fonctionnalités de Twig

- **Filter Date twig:**

```
{{ "now" | date("m/d/Y") }}
```

```
{{ "now" | date('d/m/y', 'Europe/Paris') }}
```

```
{{ "now" | date('d/m/y H:i:s', 'Europe/Paris') }}
```

- **Include twig:** en plus des méthodes filter twig nous permet d'aller plus loin dans comme inclure des templates avec la méthode **'include'**. Pour ce faire nous allons créer une méthode une template myInclude.html.twig dans /home et l'appeler dans notre template home.

- **Extends & Block twig:** permet de surcharge une vue sur une autre

exemple:

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

 qui dans 'home' template.

Etape 2 : Les Assets

Pour charger nos css et images qui sont dans le dossier public symfony propose une solution **asset**

La méthode asset nous retourne le dossier public

Exemple:

`{{asset(images)}}` \Rightarrow ça retourne le dossier
.public/images

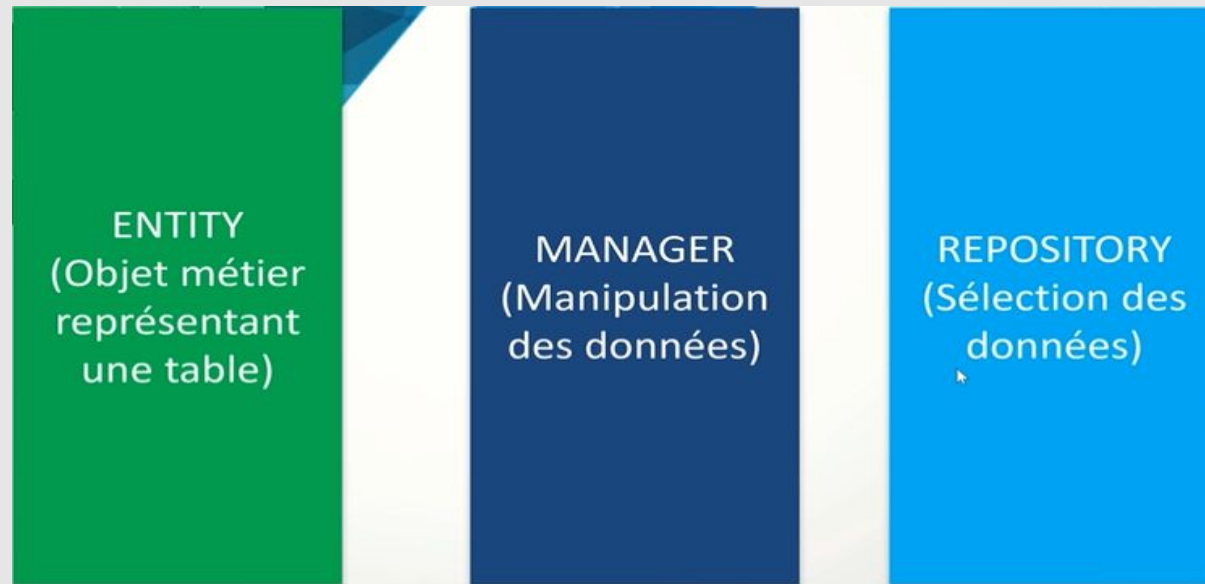
Pour l'installer on fait : **composer require symfony/asset**

Etape 3 : Bootswatch

[Bootswatch](#) nous propose des thèmes Bootstrap gratuit qu'on va utiliser dans notre projet.

- Dans public créé un dossier assets/styles pour mettre notre css téléchargé sur [bootswatch](#)
 - Ajouter la balise <link> dans base.html.twig pour prendre le css
 - Choisir une balise <nav> sur bootswatch pour le mettre dans notre projet (base.html.twig)

Etape 4 ORM Doctrine (Object Relationnal Mapper)



- Installation du package ORM (**composer req orm**)

composer require symfony/orm-pack

--update-with-all-dependencies

Etape 4 ORM Doctrine (Object Relationnal Mapper)

Symfony fournit tous les outils dont vous avez besoin pour utiliser des bases de données dans vos applications grâce à [Doctrine](#), le meilleur ensemble de bibliothèques PHP pour travailler avec des bases de données. Ces outils prennent en charge les bases de données relationnelles comme MySQL et PostgreSQL ainsi que les bases de données NoSQL comme MongoDB.

- Configuration de la base de données :

Les informations de connexion à la base de données sont stockées sous forme de variable d'environnement appelée **DATABASE_URL**. Pour le développement, vous pouvez retrouver et personnaliser ceci à l'intérieur **.env**:

```
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"
```

Etape 4 ORM Doctrine (Object Relationnal Mapper)

- **Création de la base de donnée :**

Maintenant que vos paramètres de connexion sont configurés, Doctrine peut créer la db_name base de données pour vous.

`symfony console doctrine:database:create`

Etape 5 : Création des entités

Supposons que pour la création de notre application 'blog' dans laquelle les **articles**, les **users** et **category** doivent être affichés. Sans même penser à la doctrine ou aux bases de données, vous savez déjà que vous avez besoin d'un Articleobjet, Userobjet et Categoryobjet pour représenter les articles, les users, et les category.

- Création des entités (Article, User, Category)

- **symfony console make :entity**

- article**

- title: string

- content: text

- image: string

- createdAt: datetime

- user**

- username: string

- firstname: string

- lastname: string

- password: string

- email: string

- createdAt: datetime

- category**

- title: string

- description : text (Null)

- image : string

- Mapping entres les entités

- Article et Category (ManyToMany)

- Article et User (ManyToOne)

Etape 5 : Création des entités

Les entités sont entièrement configurées et prêtes à être enregistrées sous forme de table dans la base. Si vous venez de définir les classes, votre base de données ne dispose pas encore des tables. On va créer maintenant une version de migration pour la base donnée.

`symfony console make:migration`

Si le fichier de migration contient le SQL nécessaire à la mise à jour de votre base de données, On peut lancer la commande suivante pour l'envoi à la base de donnée.

`symfony console doctrine:migrations:migrate`

Etape 5 : Création des entités

Résumé des commandes de doctrine ORM

- `symfony console doctrine:database:create` (créer la base)
- `symfony console make:migration` (fabrique une migration)
- `symfony console doctrine:migrations:migrate` (migration vers la BD)

Etape 6 : Fixtures avec doctrine

Les fixtures sont les jeux de données pour notre base de données. Elles peuvent être utilisées à des fins de test ou pour vous fournir des données intéressantes pendant que vous développez votre application.

- `composer req --dev orm-fixtures` (Installation du composant)

[fakerPHP](#): permet la création des données aléatoires pour être insérer dans la base.

- `composer req fakerphp/faker` (création des données aléatoires).

Dans le dossier Fixtures préparer vos données.

- `symfony console doctrine:fixtures:load` (Chargement des données dans BD)

Chapitre 2 : Récupération des données

- **Affichage des Articles:** Dans cette partie on va essayer d'afficher dans le home.html.twig les articles articles dans la base. Dans Home/home.html.twig en vous basant sur la classe card de bootstrap pour plus de style d'affichage.
- **Affichage d'un article:** Dans cette partie on va essayer d'afficher la page de présentation d'un article.
 - Dans le HomeController.php crée la méthode show (action) pour récupérer un article.
 - Créer le template show.html.twig (pour le détail d'un article)
 - Dans la home.html.twig faites en sorte que le lien « En savoir plus » vous ramène sur le détail d'un article.

Chapitre 3 : Création du Dashboard d'administration

- Installation du composant Admin

`composer req admin` (installation du composant)

`symfony console make:admin:dashboard` (Installation du Dashboard)


Tester la page admin en faisant /admin dans le navigateur


- Création de crud pour nos entités (Article, Category, User)


`symfony console make:admin:crud`


- Dans le DashboardController.php mettez à jour la méthode configureMenuItems() pour avoir l'affichage ci-dessous

- [Ici EasyAdmin](#)

 Dashboard

 Categories

 Articles

 User

Welcome to EasyAdmin 3

You have successfully installed EasyAdmin 3 in your application!

You are seeing this example page because you haven't configured the start page of your Dashboard. To do that, open the following file in your editor:

```
C:\Users\ASUS\Desktop\Formation-Lyon\TP-Lyon\src\Controller\Admin\DashboardController.php
```

Then, add the following method to it to customize the Dashboard start page:

```
use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractDashboardController;  
use EasyCorp\Bundle\EasyAdminBundle\Router\AdminUrlGenerator;
```

Personnalisation du dashboard

- Dans la liste des articles de l'admin on ne voit pas la catégorie et les users affichés (à corriger)
- Lors de l'ajout d'un article on ne voit pas la possibilité de choisir le user et la catégorie (à corriger)

composer require symfony/mime: est une norme Internet qui étend le format de base d'origine des e-mails pour prendre en charge des fonctionnalités telles que :

- Lors de l'ajout d'un article avoir la possibilité d'uploader une image.

Pour ces corrections nous allons agir dans ArticleCrudController.php

Une fois que tout est ok vide la table article pour ajouter un article via admin

Supprimez la base de données afin de mettre les vraies données depuis notre page admin

- `symfony console doctrine:database:drop --force` (Suppression BD)
- `symfony console d:d:c` (Créer la BD)
- `symfony console make:migration` (Créer une migration)
- `symfony console d:m:m` (Migration des données)

Afficher les nouvelles données enregistrées correctement sur la page home

TP:

1 - Afficher les catégories sur la page d'accueil à droite avec le nombre d'articles que contient la catégorie comme indiqué sur la page suivante du TP

2 - Rendre cliquable les catégories pour être redirigé sur la même page qui [home.html.twig](#) et affiche la liste des articles contenant dans cette catégorie

Indices : Pour la cette partie créer une nouvelle méthode dans `HomeController.php` [showArticlesCategory\(\)](#).

3 - Mettre à jour notre navbar avec la route accueil pour être redirigé à la page d'accueil.

Page Acceuil

Ecrit le 08/10/23 à 14:10 par PierreCat



Voiture BM

En Savoir plus

Ecrit le 08/10/23 à 14:10 par PierreCat



Mercedes

En Savoir plus

Ecrit le 14/10/23 à 00:10 par claudhost



videos

En Savoir plus

Catégories

[photo](#) 2 article(s)

[ciné](#) 2 article(s)

[acteur](#) 1 article(s)

Chapitre 4 : Les Formulaires

Un formulaire Web est composé de champs de saisie (texte, liste déroulante, cases à cocher, etc.). Ce chapitre présente la gestion de ces champs en utilisant le système de formulaires de symfony.

Pour l'installation : **composer require symfony/form**

- Construisez le formulaire dans un contrôleur Symfony ou en utilisant une classe de formulaire dédiée.
- Rendre le formulaire dans un modèle afin que l'utilisateur puisse le modifier et le soumettre.
- Traitez le formulaire pour valider les données soumises, transformez-les en données PHP et faites-en quelque chose comme par exemple, conservez-les dans une base de données.

composer require symfony/validator

Chapitre 4 : Les Formulaires

Étape 1: Créer un formulaire de **Contact** pour notre blog. pour le faire on va d'abord une entité contact avec les champs (`firstname_lastname`, `email`, `subject`, `description`) et un `ControllerContact.php` et par la suite créer notre formulaire de contact

- `symfony console make:form`

Étape 2: Mettez quelques règles de validation de contrôle dans les champs.

Chapitre 5 : Sécurité et Authentification

Le SecurityBundle, que vous découvrirez dans ce guide, fournit toutes les fonctionnalités d'authentification et d'autorisation nécessaires pour sécuriser votre application.

`composer require symfony/security-bundle`

- On va essayer de faire en sorte que nos users puissent se connecter à notre application.
- Pour le faire on va créer une sécurité pour nos users avec

`php bin/console make:user`.

- Créer un `registerController` pour notre formulaire de user.
- Créer le formulaire `userType.php`.
- Mettre à jour notre entité user avec un nouveau champ (`private $passwordConfirm`) qui ne sera pas envoyée dans la bd.

Chapitre 5 : Sécurité et Authentification

```
public function getpasswordConfirm(): ?string {  
    return $this->passwordConfirm;  
}
```

```
public function setpasswordConfirm(string $passwordConfirm): self {  
    $this->passwordConfirm = $passwordConfirm; return $this;  
}
```

Faites en sorte que le champs soit identiques avant l'envoi à la BD



The image shows a web form for password confirmation. It consists of two input fields. The first field is labeled 'Password' and the second field is labeled 'Password confirm'. Below the second field, there is a red error message that reads 'Les deux mots de passe doivent être identiques'. At the bottom of the form, there is a dark blue button labeled 'Send'.

Chapitre 5 : Sécurité et Authentification

Etape 1: On va essayer de hasher nos mots de passes lors de l'enregistrement du user dans la BD. Pour le faire on va ajouter ces implémentations à notre entité User “**implements UserInterface, PasswordAuthenticatedUserInterface**” cette implémentation nécessite des méthodes suivantes:

Etape 2: On va essayer de hacher notre mot de passe lors de l'enregistrement de notre user dans la BD en vous basant sur `UserPasswordHasherInterface` avant la soumission du formulaire.

Etape 2: Une fois le user enregistré faites en sorte qu'on soit rediriger vers une page de login.

Chapitre 5 : Authentification de user

Etape 1: Dans cette étape on va essayer de connecter nos utilisateurs votre a application à travers le Auth de symfony.

- `symfony console make:auth` cette commande permet de nous fournir éléments pour mener à bien l'authentification.

Dans security.yaml ajouter les paramètres suivants:

```
app_user_provider:  
    entity:  
        class: App\Entity\User  
        property: email
```

le providers permet de récupérer des utilisateurs depuis une base de données en fonction d'un "identifiant d'utilisateur" par exemple l'adresse email ou le nom d'utilisateur de l'utilisateur. La configuration ci-dessus utilise Doctrine pour charger User entité en utilisant la propriété email comme « identifiant utilisateur ».

dans logout ajouter le `target : app_home` pour la redirection.

`composer require --dev symfony/profiler-pack` pour voir le profiler

Chapitre 5 : Authentification de user

Etape 2: Faites en sorte que si l'utilisateur est connecté d'afficher seulement << Déconnexion >> et contraire s'il est déconnecté d'afficher << Inscription et Connexion >>. Le traitement le fait dans notre `base.html.twig`. Basez-vous sur les conditions que propose twig

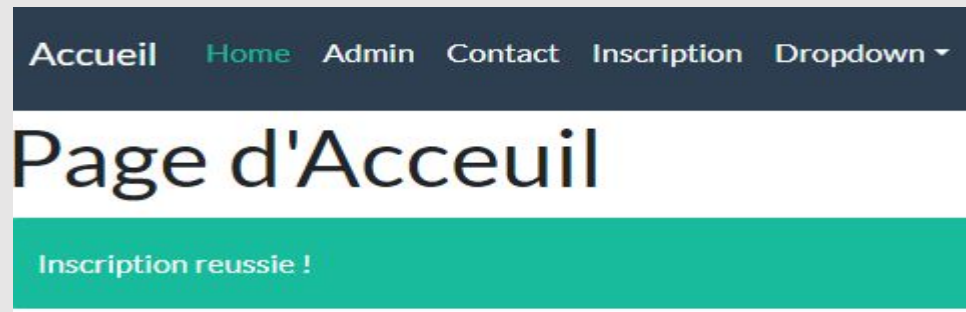
Etape 3 : Gestion des rôles

- Faites en sorte qu'on puisse afficher la page `Admin` que quand on a le rôle <<ROLE_ADMIN>>.

Tester l'accès à la page `Admin` avec un user dont on va donner le rôle admin manuelle à base de données.

- Faites en sorte qu'on puisse afficher le détail d'un article que quand l'utilisateur est connecté.

Etape 3 : Afficher un message de succès sur la page home avec `addFlash()` quand l'utilisateur est bien inscrit.



Chapitre 5 : Authentification de user

Etape 4: Une fois l'utilisateur connecté, il sera redirigé automatiquement avec un petit message affiché de Bonjour avec son nom et prénom dans son espace et cette espace sera dans un nouveau controller `accountController`.

Etape 5: Mettez en place dans `ArticleRepository.php` une méthode `findArticlesByUser()` qui récupère les articles d'un utilisateur donné. Récupérer le résultat dans le `accountController` et afficher ces articles dans son template.