

# Symfony

Cheikh Ahmadou Bamba TOUNKARA  
[cabt99@gmail.com](mailto:cabt99@gmail.com) / 778292807

# Le Modèle MVC

Répondre aux besoins des applications interactives en séparant les problématiques liées aux différents composants au sein de leur architecture respective.

Ce paradigme regroupe les fonctions nécessaires en trois catégories :

# Le Modèle MVC

- un **Modèle** (modèle de données) ;
  - Le modèle décrit les données manipulées par l'application et représente le cœur (algorithmique) de l'application : traitements des données, interactions avec la base de données, etc
- une **Vue** (présentation, interface utilisateur) ;
  - Sa première tâche est de présenter les résultats renvoyés par le modèle.  
Sa seconde tâche est de recevoir toute action de l'utilisateur
- un **Contrôleur** (logique de contrôle, gestion des événements, synchronisation).
  - Il reçoit tous les événements de la vue et enclenche les actions à effect

## Exemples d'architecture MVC en PHP

### Best High Ranking PHP Frameworks



Symfony



# qu'est-ce que Symfony

- Symfony est un framework php pour la création de site web et d'applications web
- Un ensemble de composant réutilisable comme Validator, HTTP Kernel(utilise dans Drupal, Laravel framework)
- Embarque les meilleurs pratiques du développement logiciel

# Pourquoi utiliser Symfony ?

- Framework Mature (première release en 2005)
- LTS Versions (Long Term Support)
- Grande communauté (facile de trouver de l'aide sur StackOverflow, beaucoup d'article de blog.....)
- Bonne documentation
- Utilise et encourage les bonnes pratiques de développement (Comme l'injection de dépendances)

# Pourquoi utiliser Symfony

- Symfony est un framework php pour la création de site web et d'applications web
- Un ensemble de composant réutilisable comme Validator, HTTP Kernel(utilise dans Drupal, Laravel framework)
- Embarque les meilleurs pratiques du développement logiciel

# Pourquoi utiliser Symfony

- **Fiabilité éprouvée**

Symfony a prouvé sa fiabilité au fil du temps alors que de nombreux autres frameworks ont échoué. De nombreuses plateformes notables comme phpBB, Drupal, Laravel, Magento et eZ Publish utilisent ses composants. Et vous connaissez certainement les



**Bla Bla Car**

**TED**  
**VOGUE**



# Installation Symfony

Télécharger et créer une application symfony:

<https://symfony.com/doc/current/setup.html>

# Installation de symfony

- Télécharger et créer une application

symfony: <https://symfony.com/doc/current/setup.html>

- Installer Symfony CLI. Cela crée un binaire appelé symfony qui fournit tous les outils dont vous avez besoin pour développer et exécuter votre application Symfony localement. <https://symfony.com/download>

# Installation de symfony

- `curl -sS https://get.symfony.com/cli/installer | bash`

```
bambas-mbp:~ bamba.tounkara$ curl -sS https://get.symfony.com/cli/installer | bash
```

# Installation de symfony

- `mv /Users/bamba.tounkara/.symfony/bin/symfony /usr/local/bin/symfony`

```
Finding the latest version (platform: "darwin_amd64")...
Downloading version 4.18.4 (https://github.com/symfony/cli/releases/download/v4.18.4/symfony_darwin_amd64.gz)...
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left     Speed
100    649    100    649     0     0    421      0  0:00:01  0:00:01 --:--:--    421
100 7895k    100 7895k     0     0  414k      0  0:00:19  0:00:19 --:--:--   466k
Uncompress binary...
Making the binary executable...
Installing the binary into your home directory...
The binary was saved to: /Users/bamba.tounkara/.symfony/bin/symfony

The Symfony CLI v4.18.4 was installed successfully!

Use it as a local file:
/Users/bamba.tounkara/.symfony/bin/symfony

Or add the following line to your shell configuration file:
export PATH="$HOME/.symfony/bin:$PATH"

Or install it globally on your system:
mv /Users/bamba.tounkara/.symfony/bin/symfony /usr/local/bin/symfony

Then start a new shell and run 'symfony'
bambas-mbp:~ bamba.tounkara$
```

## Créer un projet symfony

*# use the most recent LTS version*

```
$ symfony new my_project_name --version=lts
```

*# use the 'next' Symfony version to be released (still in development)*

```
$ symfony new my_project_name --version=next
```

*# you can also select an exact specific Symfony version*

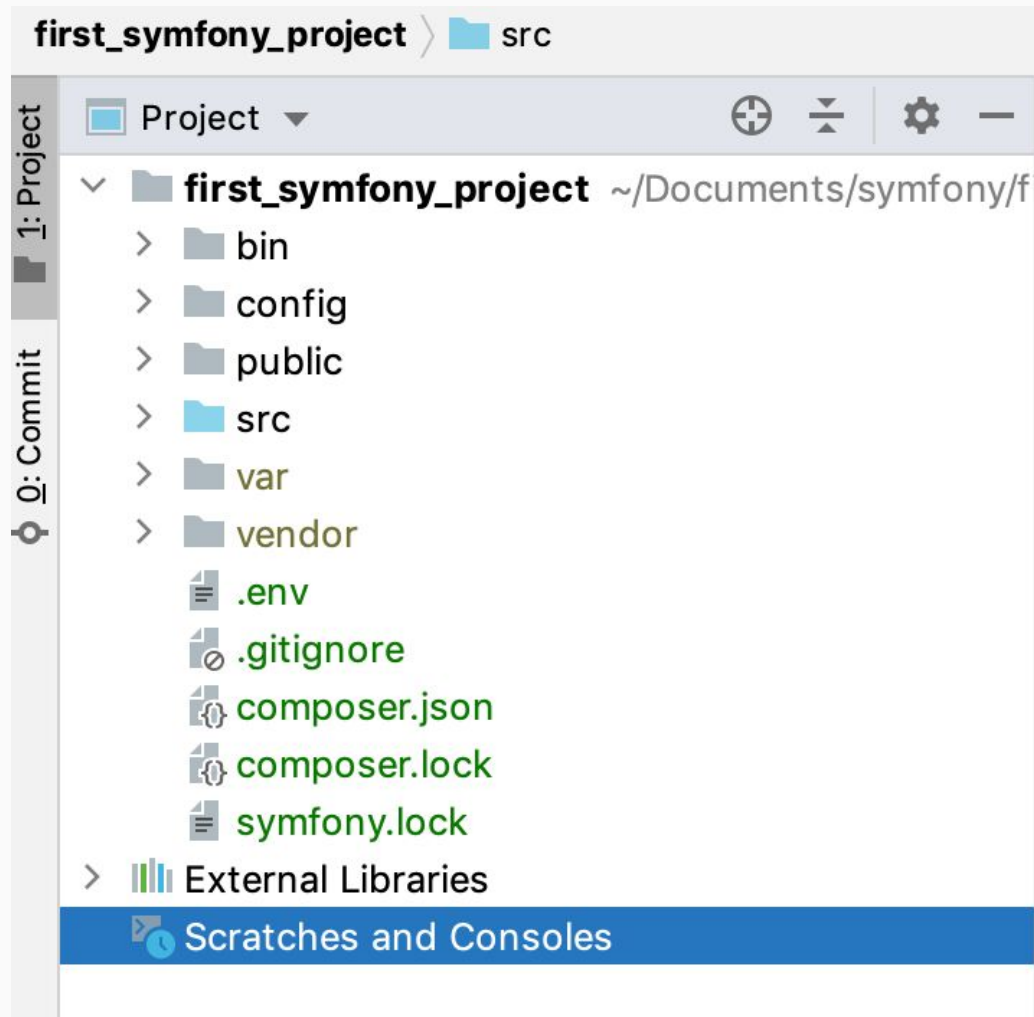
```
$ symfony new my_project_name --version=4.4
```

## Créer un projet symfony

- `symfony new first_symfony_project --version=lt5`
- Pour démarrer se déplacer dans le dossier du projet est tapé : `symfony server:start`
- Ouvrir votre navigateur est allé à l'URL <http://localhost:8000>
- `symfony check:req`

# l'architecture du framework

Après installation on a l'arborescence des dossiers suivant



# L'architecture du framework

- config/
  - La configuration de l'application avec les fichiers de configuration de chacun des packages de l'application.
- bin/
  - Fichiers exécutables (par exemple, bin / console)
- src/
  - Le code source PHP du projet: Dans ce répertoire, le code est organisé en bundles, des briques de l'application



# L'architecture du framework

- `var/`
  - Fichiers générés (cache, logs, etc.).
- `vendor/`
  - Les dépendances externes à l'application.
- `public/`
  - Le répertoire racine Web : contient tous les fichiers destinés à vos visiteurs : images, fichiers CSS et JavaScript, etc. Il contient également le contrôleur frontal (`index.php`) en d'autre terme le fichier exécuter par le web serveur

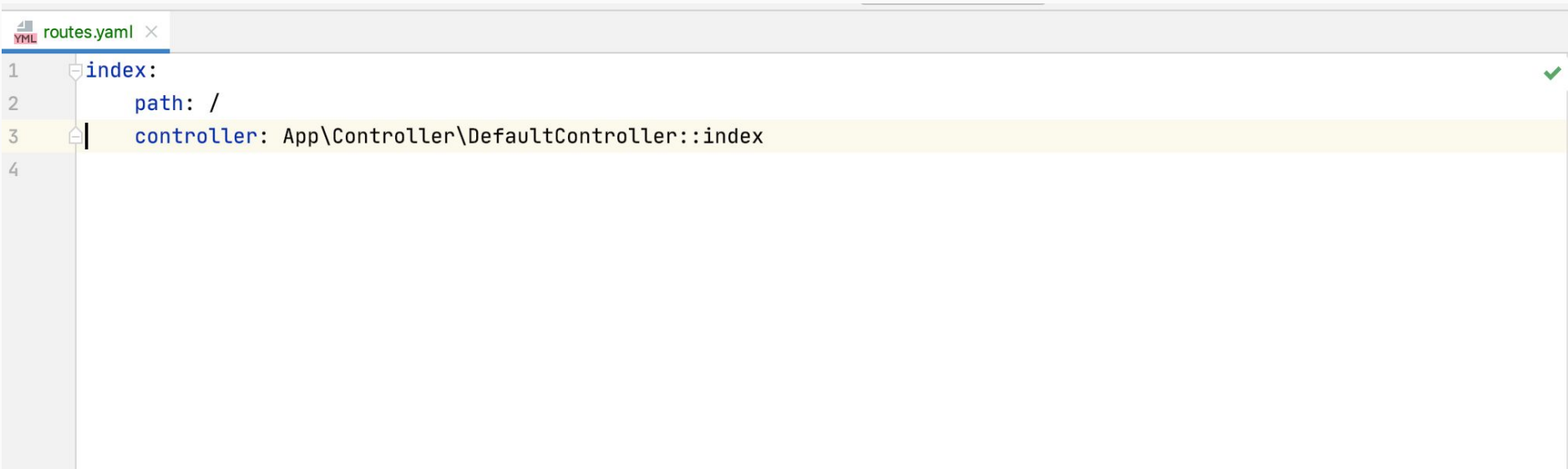
## Route, Controllers & Responses!

- Tous les framework dans tous les langages ont le même principe principe: de donner une route et un système de contrôleur, les 2 étapes pour construire une page
- **Route + Controller = Page**

Une route définit l'URL de la page et le contrôleur est l'endroit où nous écrivons le code PHP pour construire cette page, comme le HTML ou JSON.

# Route, Controllers & Responses!

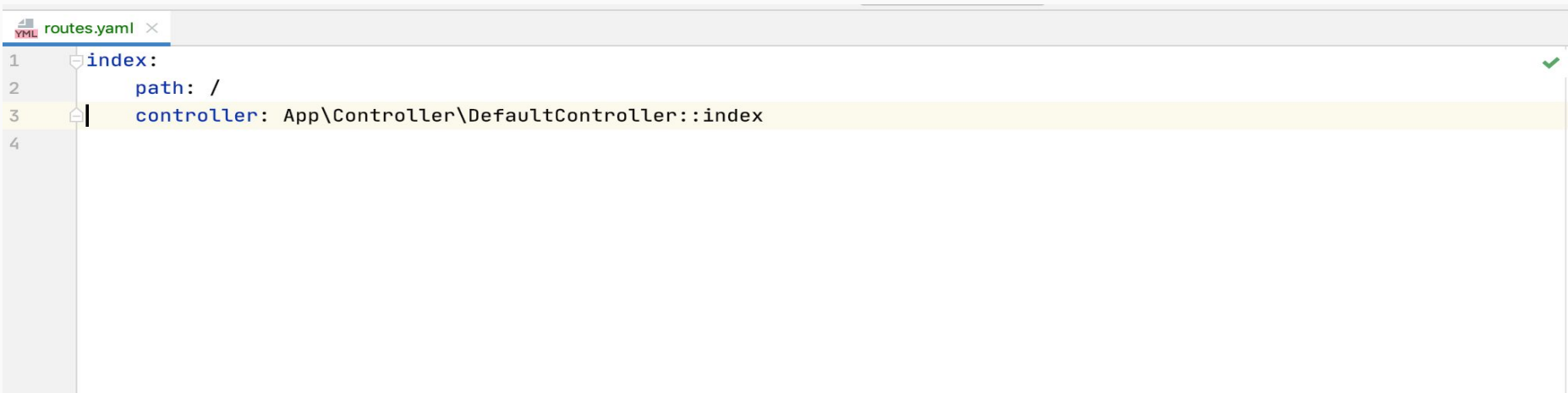
- Ouvrir le fichier config/route.yaml
- YAML est très convivial: c'est un format de configuration clé-valeur séparé par deux-points. L'indentation est également importante.



```
1 index:
2   path: /
3   controller: App\Controller\DefaultController::index
4
```

## Route, Controllers & Responses!

- Cela crée une route unique dont l'URL est /. Le contrôleur pointe vers une fonction qui va construire cette page ... il pointe vers une méthode sur une classe.
- Autrement dit lorsque l'utilisateur accède à la page d'accueil, veuillez exécuter la méthode d'index sur la classe DefaultController.



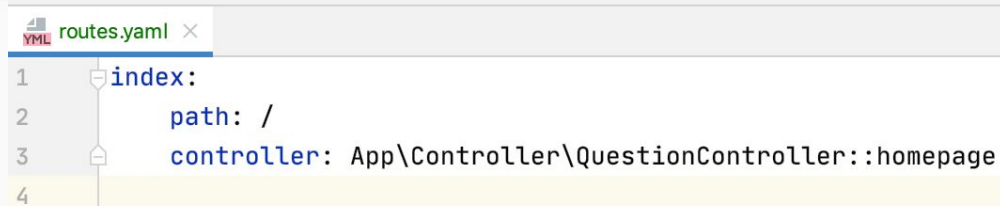
```
routes.yaml
1 index:
2   path: /
3   controller: App\Controller\DefaultController::index
4
```

The screenshot shows a code editor with a file named 'routes.yaml'. The file contains a single route configuration for the 'index' action. The configuration is as follows:

- Line 1: `index:`
- Line 2: `path: /`
- Line 3: `controller: App\Controller\DefaultController::index`
- Line 4: (empty line)

A green checkmark is visible in the right margin next to line 1, indicating that the configuration is valid.

## Creation du controleur



```
routes.yaml
1 index:
2   path: /
3   controller: App\Controller\QuestionController::homepage
4
```

- L'url étant défini et pointe vers le contrôleur qui construira la page, Maintenant nous devons créer ce contrôleur! Dans le répertoire src/, il y a déjà un répertoire Controller/. Appelez-le QuestionController.

## Creation du controleur

Espaces de noms(namespace) et src /Directory

Chaque classe que nous créons dans le répertoire src/ aura besoin d'un espace de noms. l'espace de noms doit être App\ suivi du répertoire dans lequel se trouve le fichier. Puisque nous créons ce fichier dans le répertoire Controller/, son espace de noms doit être App \ Controller.

## Creation du controlleur

YML routes.yaml x C QuestionController.php x

1 <?php

2

3

4 namespace App\Controller;

5

6

7 class QuestionController

8 {

9 |

10

11 }

⚠ 1 ^ v

# Creation du controleur

```
<?php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

class QuestionController
{
    public function homepage() {
        return new Response( content: "On redirige ici avec le controleur");
    }
}
```

⚠ 1 ✓ 3 ^



## Création du contrôleur

Le contrôleur, qui est aussi parfois appelée une "action" ....

Notre travail ici est simple: construire la page. Nous pouvons écrire n'importe quel code dont nous avons besoin pour faire cela - comme faire des requêtes de base de données, mettre en cache des choses, effectuer des appels d'API, extraire des crypto-monnaies ... peu importe. La seule règle est qu'une fonction de contrôleur doit renvoyer un objet Symfony Response.

## Création du contrôleur

il existe plusieurs classes de réponse dans notre application. Celui que nous voulons provient de `Symfony\Component\HttpFoundation`. `HttpFoundation` est l'une des parties les plus importantes - ou «composants» - de Symfony.

## Route avec annotations

Créer une route dans YAML qui pointe vers une fonction de contrôleur est assez simple. Mais il existe un moyen encore plus simple de créer des routes .. Ça s'appelle: des annotations.

Tout d'abord, commentez la route YAML. En gros, supprimez-le entièrement. Pour prouver que cela ne fonctionne pas, actualisez la page d'accueil. Oui! C'est de retour à la page d'accueil.

```
#index:  
# 🟡 path: /  
# controller: App\Controller\QuestionController::homepage
```

### Installation des annotations

Les annotations sont un format de configuration spécial ... et la prise en charge des annotations n'est pas standard dans notre petite application Symfony. En fait, c'est toute la philosophie de Symfony: commencez petit et ajoutez des fonctionnalités lorsque vous en avez besoin.

Pour ajouter la prise en charge des annotations, nous utiliserons Composer pour exiger un nouveau package. Si Composer n'est pas déjà installé, accédez à <https://getcomposer.org>.

Une fois que vous avez fait, exécutez: **php composer.phar require annotations** ou **composer require annotations**

# Route avec annotations

```
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class QuestionController
{
    /**
     * @Route("/")
     */
    public function homepage() {
        return new Response( content: "On redirige ici avec le controleur");
    }
}
```

# Création d'une seconde page

Créons une deuxième route et un contrôleur pour cela.

```
class QuestionController
{

    /**
     * @Route("/")
     */
    public function homepage() {
        return new Response( content: "On redirige ici avec le contrôleur");
    }

    /**
     * @Route("/questions/comment-devenir-web-developer")
     */
    public function show()
    {
        return new Response( content: 'Future page pour repondre au question!');
    }
}
```

## Route dynamic

Finalement, nous allons avoir une base de données pleine de questions. Et donc, non, nous n'allons pas créer manuellement une route par question. Au lieu de cela, nous pouvons rendre cette route plus intelligente. Remplacez la partie comment devenir développeur web par une variable dynamic {slug}.

```
/**
 * @Route("/questions/{slug}")
 */
public function show($slug)
{
    return new Response(sprintf(
        'Future page to show the question "%s"!',
        $slug
    ));
}
```

# Route dynamic

```
/**
 * @Route("/questions/{slug}")
 */
public function show($slug)
{
    return new Response(sprintf(
        ucwords(str_replace( search: '-', replace: ' ', $slug))
    ));
}
```



php bin/console

Example php bin/console debug:router

## Les vues avec twig

- Comme expliqué dans chapitres sur les contrôleurs, ces derniers sont chargés de gérer chaque requête qui arrive dans une application Symfony et ils finissent généralement par rendre un modèle pour générer le contenu de la réponse.
- En réalité, le contrôleur délègue la majeure partie du travail lourd à d'autres endroits afin que le code puisse être testé et utilisé. Lorsqu'un contrôleur a besoin de générer du code HTML, CSS ou tout autre contenu, il transmet le travail au moteur de template.
- Pour installer twig `php composer.phar require twig/twig` ou `composer require twig/twig`

# Les vues avec twig

- Le moteur de gabarits (moteur de templates) est un composant qui permet d'isoler le code PHP de la couche présentation.
- Principe de fonctionnement:
  - La préparation des données à afficher est réalisée dans les pages PHP.
  - Le moteur de gabarits ne s'occupe que de la mise en forme et de la présentation des données à afficher.
- Twig: <https://twig.symfony.com/>

# Les vues avec twig

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Welcome!{% endblock %}</title>
    {% block stylesheets %}{% endblock %}
  </head>
  <body>
    <h1>Hello {{ message }}</h1>
    {% block body %}
    {% endblock %}
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

# Les vues avec twig

Twig définit deux types de syntaxe spéciale :

- `{{ ... }}`: « Dit quelque chose »: écrit une variable ou le résultat d'une expression dans le template ;
- `{% ... %}`: « Fait quelque chose »: un tag qui contrôle la logique du template ; il est utilisé pour exécuter des instructions comme la boucle for par exemple

## Les vues avec twig

- Faisons en sorte que notre contrôleur `show()` retourner du vrai HTML en utilisant un modèle. Dès que vous souhaitez renvoyer un template, vous devez faire en sorte que votre contrôleur hérite **AbstractController**.

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

```
class QuestionController extends AbstractController  
{
```

```
    ...
```

# Les vues avec twig

nous allons utiliser `return $this->render()` et passez deux arguments:

- Le premier est le nom de fichier du template
- Le deuxième argument est un tableau de toutes les variables que nous voulons transmettre au modèle.

```
/**
 * @Route("/questions/{slug}")
 */
public function show($slug)
{
    return $this->render([
        'question' => ucwords(str_replace([
            'search' => '-',
            'replace' => ' ',
            $slug
        ]));
    ]);
}
```

# Les vues avec twig

- Création du template: Dans templates/, créez un sous-répertoire de questions, puis un nouveau fichier appelé show.html.twig. Commençons simplement: un `<h1>` puis `{{question}}` pour rendre la variable question.

```
<h1>{{ question }}</h1>
```

```
<div>
```

💡 Eventually, we'll print the full question here!

```
</div>
```



# Les vues avec twig

```
/**
 * @Route("/questions/{slug}")
 */
public function show($slug)
{
    $answers = [
        'Etre bon en algo',
        'Beaucoup reflechir c\'est du | pas de apt-get install et config... 🤔',
        'Etre passionnée?',
    ];

    return $this->render( view: 'question/show.html.twig', [
        'question' => ucwords(str_replace( search: '-', replace: ' ', $slug)),
        'answers' => $answers
    ]);
}
```

# Les vues avec twig

```
<h1>{{ question }}</h1>
{# oh, I'm just a comment hiding here #}
<div>
    Eventually, we'll print the full question here!
</div>
<h2>Answers</h2>
<ul>
    {% for answer in answers %}
        <li>{{ answer }}</li>
    {% endfor %}
</ul>
```

# Les vues avec twig

La plupart de nos pages partageront une mise en page HTML. Pour le moment, nous n'avons aucune structure HTML. Pour lui en donner, en haut du template, ajoutez {% extends 'base.html.twig'%}.

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

```
<h1>{{ question }}</h1>
```

```
{# oh, I'm just a comment hiding here #}
```

```
<div>
```

```
    Eventually, we'll print the full question here!
```

```
</div>
```

```
<h2>Answers</h2>
```

```
<ul>
```

```
    {% for answer in answers %}
```

```
        <li>{{ answer }}</li>
```

```
    {% endfor %}
```

```
</ul>
```

## Les vues avec twig

Mais si vous actualisez la page ... cachez-vous! Énorme erreur!

Un modèle qui en étend un autre ne peut pas inclure de contenu en dehors des blocs Twig.

Comment y remédier? En remplaçant le bloc. Au-dessus du contenu, ajoutez `{% block body%}`, et après, `{% endblock%}`.

# Les vues avec twig

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

{% block title %}Question: {{ question }}{% endblock %}

{% block body %}
    <h1>{{ question }}</h1>
    {# oh, I'm just a comment hiding here #}
    <div>
        Eventually, we'll print the full question here!
    </div>
    <h2>Answers {{ answers|length }}</h2>
    <ul>
        {% for answer in answers %}
            <li>{{ answer }}</li>
        {% endfor %}
    </ul>
{% endblock %}
```

## Assets: CSS, Images, etc

- ❏ Télécharger le dossier `/tutorial-assets`
- ❏ copier le fichier `base.html.twig` de `/tutorial-assets`, le coller dans `templates/base.html.twig`.
- ❏ L'une des balises `css` pointe vers `/css/app.css`. C'est un autre fichier qui se trouve dans ce répertoire `/tutorial-assets`. En fait, sélectionnez le répertoire `images/` et `app.css` et copiez les deux. Maintenant, sélectionnez le dossier `public/` et collez. Ajoutez un autre répertoire `css/` et déplacez `app.css` à l'intérieur.
- ❏ Rappelez-vous: le répertoire `public/` est la racine de notre document. Donc, si vous avez besoin qu'un fichier soit accessible par le navigateur d'un utilisateur, il doit vivre ici. Le chemin `/css/app.css` chargera ce fichier `public/css /app.css`.

Symfony a deux niveaux d'intégration différents avec CSS et JavaScript.

- ❑ le niveau de base. Vraiment, pour le moment, Symfony ne fait rien pour nous: nous avons créé un fichier CSS, puis y avons ajouté une balise de lien très traditionnelle en HTML.
- ❑ L'autre niveau d'intégration, plus important, consiste à utiliser la librairie Webpack Encore: une bibliothèque fantastique qui gère la minification, le support Sass, le support React ou Vue.js et bien d'autres choses.

Mais pour le moment, nous allons garder les choses simples: vous créez des fichiers CSS ou JavaScript, vous les placez dans le répertoire public/, puis vous créez des balises de lien ou de script qui pointent vers eux.

## Assets: CSS, Images, etc

A chaque fois pour référencer un fichier statique sur votre site - comme un fichier CSS, un fichier JavaScript ou une image, au lieu de simplement mettre /css/app.css, vous devez utiliser une fonction Twig appelée **asset ()**. Donc, `{{asset ()}}` et ensuite le même chemin qu'avant.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>{% block title %}Welcome!{% endblock %}</title>
  {% block stylesheets %}
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkyl8W/kyOYUjgkvKafwix9fL8XpryOC3q5ljHmVigSv4drQzQ5QiKX8qSEULvYA" crossorigin="anonymous">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Spartan&display=swap">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome@5.12.1/css/all.min.css" integrity="sha384-0w0A9/vuSRAQOLlY7d4Rv6Ul8P2+bdn6SVtNQpRwdoDufkSRlqLv6Qsdu2d7991" crossorigin="anonymous">
    <link rel="stylesheet" href="{{ asset('app/css') }}">
  {% endblock %}
</head>
<body>
```



- ❑ Premièrement, si vous décidez de déployer votre application dans un sous-répertoire d'un domaine - comme `lacgaa.com/tdsi_overflow`, la fonction `asset()` prefixera automatiquement tous les chemins avec `/tdsi_overflow`.
- ❑ La deuxième chose qu'il fait est plus utile: si vous décidez de déployer vos actifs sur un CDN, en ajoutant une ligne à un fichier de configuration, tout à coup, Symfony prefixera chaque chemin avec l'URL de votre CDN.

- ❑ Premièrement, si vous décidez de déployer votre application dans un sous-répertoire d'un domaine - comme `lacgaa.com/tdsi_overflow`, la fonction `asset()` prefixera automatiquement tous les chemins avec `/tdsi_overflow`.
- ❑ La deuxième chose qu'il fait est plus utile: si vous décidez de déployer vos actifs sur un CDN, en ajoutant une ligne à un fichier de configuration, tout à coup, Symfony prefixera chaque chemin avec l'URL de votre CDN.

Pour l'installer `composer require symfony/asset`

## Assets: CSS, Images, etc

- ❏ De retour dans le répertoire `tutorial-assets/`, ouvrez `show.html.twig`, copiez son contenu, et collez dans `templates/question/show.html.twig`.
- ❏ Utilisez `asset` pour l'image

## Assets: CSS, Images, etc

- ❑ La dernière page que nous n'avons pas stylisée est la page d'accueil qui pour le moment ... imprime du texte.

```
/**
 * @Route("/")
 */
public function homepage() {
    return $this->render( view: 'question/homepage.html.twig');
}
```

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

```
{% block body %}
```

```
    <h1>Page d'accueil</h1>
```

```
{% endblock %}
```

```
|
```

## Assets: CSS, Images, etc

copier `homepage.html.twig` de `tutorial-assets` et coller sur notre code `homepage.html.twig`.

## Générer une url

créer un nom pour l'url

```
/**
 * @Route("/", name="app_homepage")
 */
public function homepage() {
    return $this->render( view: 'question/homepage.html.twig');
}
```

# Générer une url

## Générer l'URL

```
<nav class="navbar navbar-light bg-light" style="...">
  <a class="navbar-brand" href="{{ path('app_homepage') }}">
    <i style="color: #444; font-size: 2rem;" class="pb-1 fad fa-cauldron"></i>
    <p class="pl-2 d-inline font-weight-bold" style="color: #444;">TDSI Overflow</p>
  </a>
  <button class="btn btn-dark">Sign up</button>
</nav>
```



# Générer une url

## Générer une URL

```
</div>
<div class="col">
  <a class="q-title" href="{ { path('app_question_show', { slug: 'comment-devenir-dev' }) } }"><h2>
  <div class="q-display p-3">
    <i class="fa fa-quote-left mr-3"></i>
    <p class="d-inline">I've been turned into a cat, any thoughts on how to turn back? While I'
    <p class="pt-4"><strong>--Tisha</strong></p>
  </div>
</div>
</div>
</div>
<a class="answer-link" href="{ { path('app_question_show', { slug: 'comment-devenir-dev' }) } }" style="...">
  <p class="q-display-response text-center p-3">
    <i class="fa fa-magic magic-wand"></i> 6 answers
  </p>
</a>
```

# Base de données en Symfony

Symfony ne fournit pas de composant pour travailler avec la base de données, mais offre une intégration avec une bibliothèque appelée Doctrine.

Doctrine consiste en ORM(Object Relational Mapping) et DBAL (Database Abstraction Layer) et ODM (Object Document Mapper)

# Base de données en Symfony

- ORM est pour le mapping des tables en classe PHP
- DBAL est couche d'abstraction sur PHP PDO
- ODM est pour le mapping des documents MongoDB en classe PHP

# Base de données en Symfony

- ORM est pour le mapping des tables en classe PHP
- DBAL est couche d'abstraction sur PHP PDO
- ODM est pour le mapping des documents MongoDB en classe PHP

# Qu'est ce qu'un ORM(Object relational Mapper)

- Technique pour interroger et manipuler les données en utilisant le paradigme de la programmation orientée objet
- Pour représenter le modèle de données en utilisant des classes PHP
- La BD devient abstrait, pas besoin d'écrire des requêtes SQL manuellement

# Installation de Doctrine

- se connecter dans la machine vagrant et se placer dans le projet

-> composer require symfony/orm-pack

->composer require --dev symfony/maker-bundle

# Installation de Symfony

- se connecter dans la machine vagrant et se placer dans le projet

-> composer require symfony/orm-pack

-> composer require --dev symfony/maker-bundle

# Configuration de Symfony

Les informations de connexion à la base de données sont stockées sous la forme d'une variable d'environnement appelée `DATABASE_URL`. Pour le développement, vous pouvez trouver et personnaliser ceci dans `.env`:



# Création de la BD

-> `php bin/console doctrine:database:create`

# Création d'un entité

-> `php bin/console make:entity`

Pour le nom `MicroPost` avec deux attributs `text` et `time`

# Migration: création de table/schéma

La classe de MicroPost est entièrement configurée et prête à être enregistrée dans une table de `micro_post`. Si vous venez de définir cette classe, votre base de données ne contient pas encore la table `micro_post`. Pour l'ajouter, vous pouvez utiliser DoctrineMigrationsBundle, qui est déjà installé:

-> `php bin/console make:migration`

# Migration: création de table/schéma

Pour exécuter les migrations

-> `php bin/console doctrine:migrations:migrate`

Cette commande exécute tous les fichiers de migration qui n'ont pas encore été exécutés sur votre base de données.

# Migration et ajout d'autres champs

Mais que se passe-t-il si vous devez ajouter une nouvelle propriété de champ à MicroPost, comme le titre? Vous pouvez modifier la classe pour ajouter la nouvelle propriété. Mais, vous pouvez également utiliser `make:entity`.

-> `php bin/console make:entity`

on ajoute le champ `title`

# Migration et ajout d'autres champs

La nouvelle propriété est mappée, mais elle n'existe pas encore dans la table product!  
Générez une nouvelle migration:

-> `php bin/console make:migration`

# Migration et ajout d'autres champs

Supprimer la modifications précédente

-> `php bin/console doctrine:migrations:status --show-versions`

-> `php bin/console doctrine:migrations:execute YYYYMMDDHHMMSS --down`

# Fixtures

Doctrine fournit une bibliothèque qui vous permet de charger par programme des données de test dans votre projet (c'est-à-dire "données de fixation").

-> `composer require --dev doctrine/doctrine-fixtures-bundle`



# Fixtures

Doctrine fournit une bibliothèque qui vous permet de charger par programme des données de test dans votre projet (c'est-à-dire "données de fixation").

-> `composer require --dev doctrine/doctrine-fixtures-bundle`

# Fixtures

```
namespace App\DataFixtures;

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

class MicroPostFixture extends Fixture
{
    public function load(ObjectManager $manager)
    {
        for($i=0; $i<10; $i++) {
            $microPost = new MicroPost();
            $microPost->setText( text: "Text aleatoire".rand(0, 100));
            $microPost->setTime(new \DateTime( time: '2019-05-03'));
            $manager->persist($microPost);
        }

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

php bin/console doctrine:fixtures:load

# Utilisation de CRUD

->php bin/console make:entity

->php bin/console make:migration

-> php bin/console doctrine:migrations:migrate

-> php bin/console make:crud

# Repository

- Repository est un design pattern
- Permet d'éviter une désorganisation des requêtes SQL du projet
- Seul point d'accès au données (Single Responsibility Principle)
- cache les détails sur la manipulation des données
- Pas besoin de comprendre le fonctionnement en profondeur du serveur de BD

# Repository

Générer un controller pour MicroPost

-> `php bin/console make:controller MicroPostController`

# Repository

## Utilisation

```
/**
 * @Route("/", name="micro_post_index")
 */
public function index() {
    $repository = $this->getDoctrine()->getRepository(persistentObject: MicroPost::class);

    return $this->render( view: 'micro_post/index.html.twig', [
        'posts' => $repository->findAll()
    ]);
}
```

## Méthode find du repository

```
/**
 * @Route("/{id}", name="micro_post_post")
 */
public function post(MicroPost $post) {

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

## Méthode findBy du repository

```
/**
 * @Route("/", name="micro_post_index")
 */
public function index() {
    $repository = $this->getDoctrine()->getRepository(persistentObject: MicroPost::class);

    return $this->render( view: 'micro_post/index.html.twig', [
        'posts' => $repository->findBy([], ['time' => 'DESC'])
    ]);
}
```



## Méthode findBy du repository

```
/**
 * @Route("/", name="micro_post_index")
 */
public function index() {
    $repository = $this->getDoctrine()->getRepository(persistentObject: MicroPost::class);

    return $this->render(view: 'micro_post/index.html.twig', [
        'posts' => $repository->findBy([], ['time' => 'DESC'])
    ]);
}
```

# Formulaire

Generation formulaire:

-> `php bin/console make:form`

# Formulaire

```
namespace App\Form;

use App\Entity\MicroPost;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class MicroPostFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add( child: 'text', type: TextareaType::class, ['label' => false])
            ->add( child: 'Enregistrez', type: SubmitType::class)
        ;
    }

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

```
/**
 * @Route("/add", name="micro_post_add")
 */
public function add(Request $request) {
    $microPost = new MicroPost();
    $microPost->setTime(new \DateTime());

    $form = $this->createForm( type: MicroPostFormType::class, $microPost);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($microPost);
        $entityManager->flush();

        return $this->redirectToRoute( route: 'micro_post_index');
    }

    return $this->render( view: 'micro_post/add.html.twig', [
        'form' => $form->createView(),
    ]);
}
```

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

{% block body %}
    {{ form_start(form) }}

    {{ form_widget(form) }}

    {{ form_end(form) }}
{% endblock %}
```

# Validation des données

La validation est une tâche très courante dans les applications Web. Les données saisies dans les formulaires doivent être validées. Les données doivent également être validées avant d'être écrites dans une base de données ou transmises à un service Web.

# Validation des données

```
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Entity(repositoryClass="App\Repository\MicroPostRepository")
 */
class MicroPost
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     * @Assert\NotBlank()
     * @Assert\Length(min=10, minMessage="Le texte doit contenir minimum 10 caractere")
     */
    private $text;

    /**
     * @ORM\Column(type="datetime")
     */
    private $time;
```