

Prise en main de Symfony 4.x

Installation et paramétrage

Première page

Environnement :

- ✓ Serveur apache : celui de wamp (préféré à celui intégré à symfony)
- ✓ Base de données : mysql en local (wamp)
- ✓ IDE : netbeans
- ✓ virtualHost : premierProjet40.Sym


Partie 1 : INSTALLATION DE COMPOSER



Ce travail n'est à faire qu'une seule fois pour l'ensemble des projets PHP et PHP/Symfony

Vérifiez que Composer n'est pas déjà installé :

Ouvrez une fenêtre de commande (ou powershell) et entrez la commande composer :

 Administrateur : Invite de commandes

```
T:\>composer
'composer' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.
```

Composer n'est pas installé !

Allez sur le site : <https://getcomposer.org/download/>

Download Composer

Windows Installer

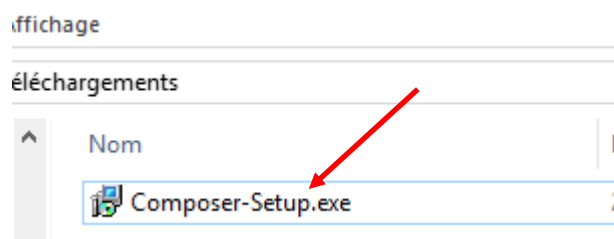
The installer will download composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

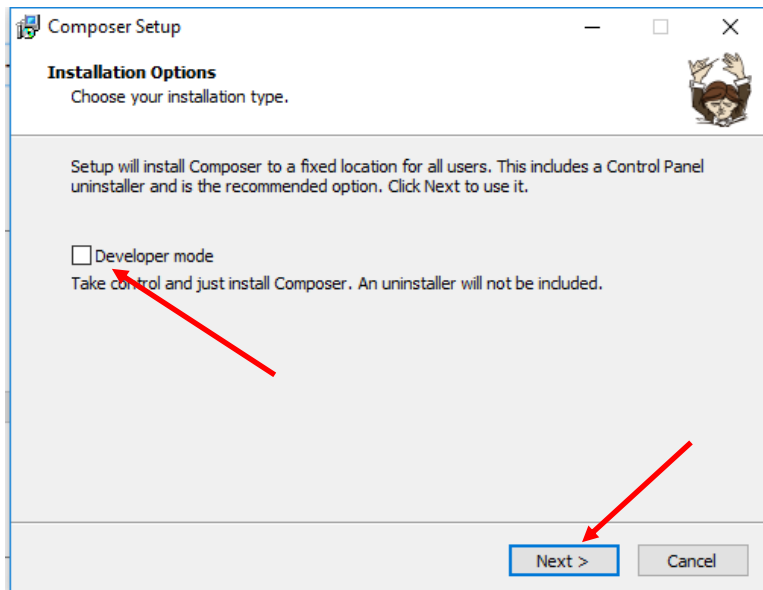
Download and run [Composer-Setup.exe](#) - it will install the latest composer version whenever it is executed.

Command-line installation

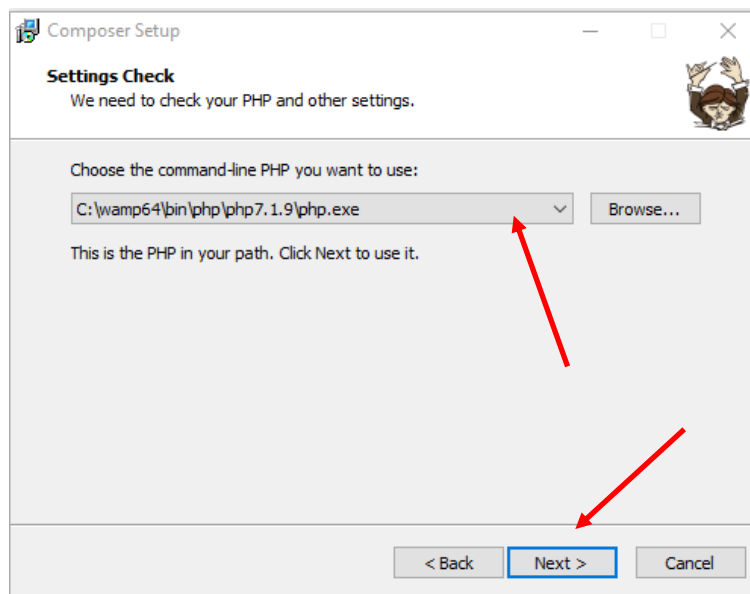
To quickly install Composer in the current directory, run the following script in your terminal. To automate the installation, use [the guide on installing Composer programmatically](#).

Téléchargez le fichier composer-setup.exe et exécutez le

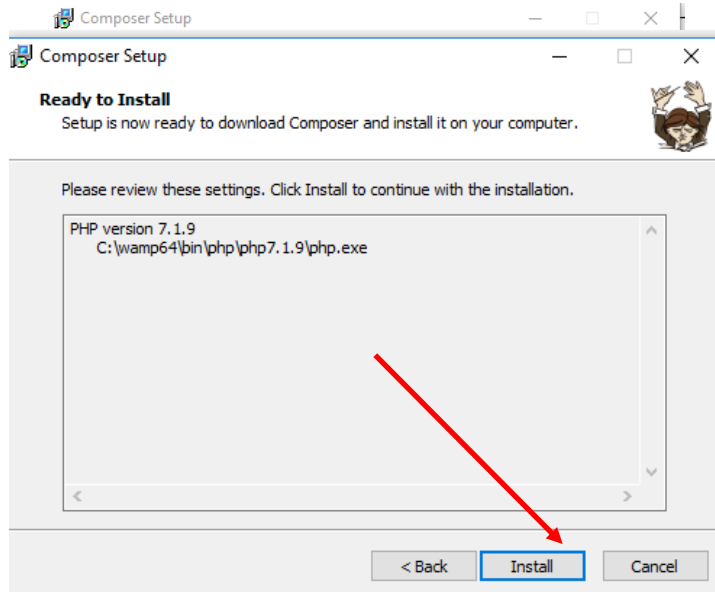




Indiquez le dossier contenant votre fichier php.exe

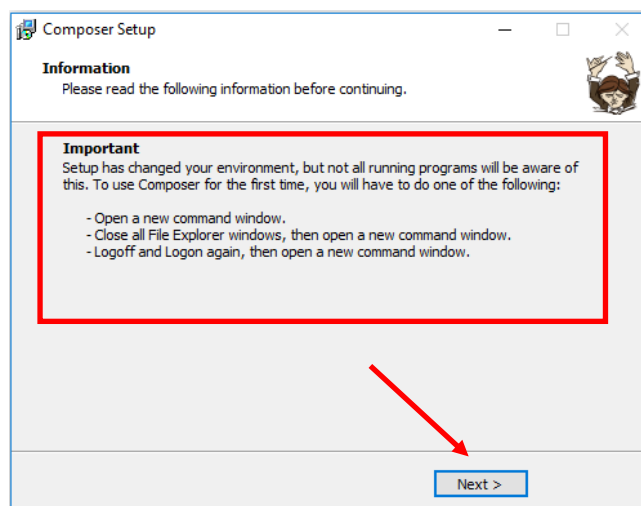


Indiquez si vous avez un proxy pour la sortie sur internet :

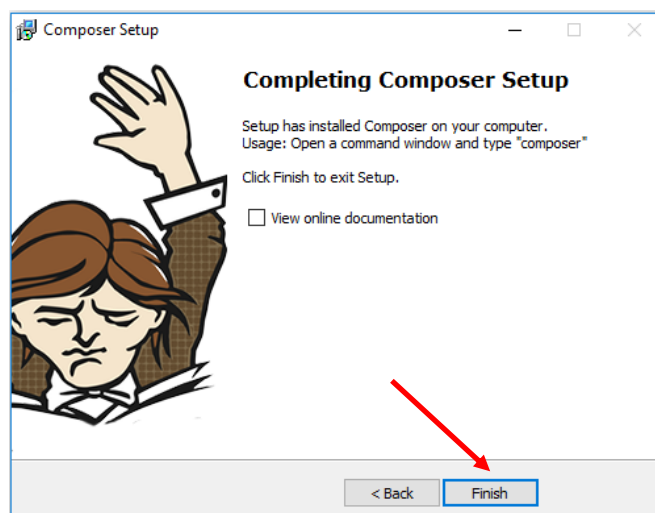


C'est parti :

Lisez bien les recommandations :



C'est fini :



Ouvrez une fenêtre de commande et tapez l'instruction composer :

```

Administrateur : Invite de commandes

T:\Wampsites>composer

Composer version 1.6.3 2018-01-31 16:28:17

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display this help message
  -q, --quiet                Do not output any message
  -V, --version              Display this application version
  --ansi                     Force ANSI output
  --no-ansi                  Disable ANSI output
  -n, --no-interaction       Do not ask any interactive question
  --profile                  Display timing and memory usage information
  --no-plugins                Whether to disable plugins.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  -v|vv|vvv, --verbose       Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and
                              3 for debug

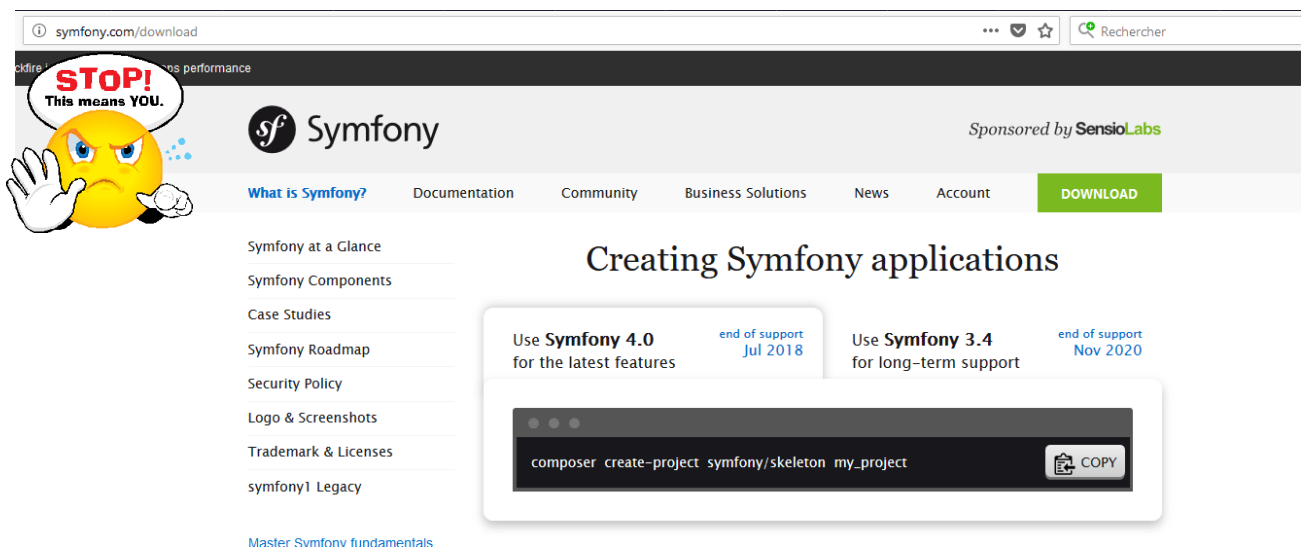
Available commands:

```

Partie 2 : CREATION DU PROJET PREMIERPROJET40

On va utiliser composer pour installer le framework symfony

Rendez-vous sur le site symfony/download : <http://symfony.com/download>



Vous allez installer la dernière version de Symfony, à savoir la version 4. On peut aller voir la roadmap de symfony pour voir que la prochaine version (4.1) sera disponible en mai 2018 :

Symfony version checker

Symfony

Symfony 4.1 will be a **stable version** published in May 2018.

Roadmap

○ ——— ○ ——— ○

May 2018 **Jan 2019** **Jul 2019**
Release End of support for bug fixes End of support for security fixes

TIP Get this information in JSON format: <https://symfony.com/roadmap.json?version=4.1>



Un petit check nous montre que la version 5.0 sortira en ... novembre 2019 ...

Symfony version checker

Symfony

Symfony 5.0 will be a **stable version** published in November 2019.

Roadmap


○ ——— ○ ——— ○

Nov 2019 **Jul 2020** **Jan 2021**
Release End of support for bug fixes End of support for security fixes

TIP Get this information in JSON format: <https://symfony.com/roadmap.json?version=5.0>



Ouvrez votre fenêtre de commande et placez-vous sur le dossier où vous souhaitez installer symfony

 Administrateur : Invite de commandes

T:\Wampsites\CoursSymfony>

Symfony sera installé dans un dossier dans le dossier CoursSymfony

Installation dernière version :

composer create-project symfony/skeleton premierprojet40

Installation d'une version spécifique ou LTS (RECOMMANDE) :

composer create-project symfony/skeleton:~4.4 premierprojet40

Et voilà en quelques secondes, c'est fait :

```

C:\> Invite de commandes
Microsoft Windows [version 10.0.18362.535]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\Benoît>composer create-project symfony/skeleton:~4.4 premierprojet40

```



Pour ceux qui ont travaillé avec Symfony 3.x, le chargement se fait beaucoup plus rapidement !!!

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

Some files may have been created or updated to configure your new packages.
Please review, edit and commit them: these files are yours.

What's next?

* Run your application:
  1. Change to the project directory
  2. Execute the php -S 127.0.0.1:8000 -t public command;
  3. Browse to the http://localhost:8000/ URL.

  Quit the server with CTRL-C.
  Run composer require server --dev for a better web server.

* Read the documentation at https://symfony.com/doc

T:\Wampsites\CoursSymfony>
```

Pour le serveur web, on peut utiliser

- ✓ le serveur intégré dans le dossier contenant l'exécutable php.exe, ce qui est suggéré par le message ci-dessus
- ✓ un autre serveur apache, par exemple celui fourni par wamp ou xamp.

Nous choisirons la deuxième possibilité pour pouvoir bénéficier de xdebug notamment

De même, pour avoir de belles URL, vous allez créer un VirtualHost qui va pointer sur le dossier Public du projet :

Windows hosts

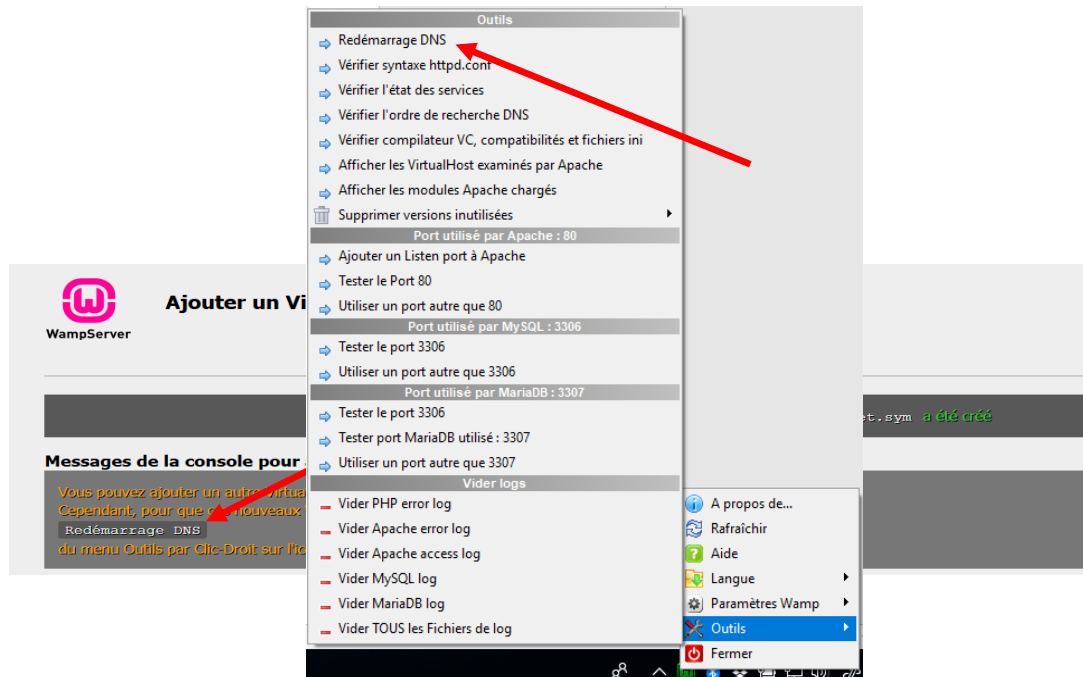
Nom du Pas de caractères diacritiques (éçëñ) - Pas d'espace - Pas de tiret bas (_) **Requis**

Chemin complet absolu du VirtualHost - Exemples : C:/wamp/www/projet/ ou E:/www/site1/ **Requis**

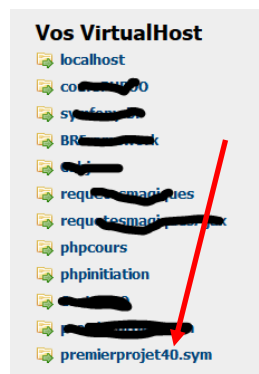
☒ Si vous voulez utiliser les VirtualHost par IP : 127.x.y.z **Optionnel**



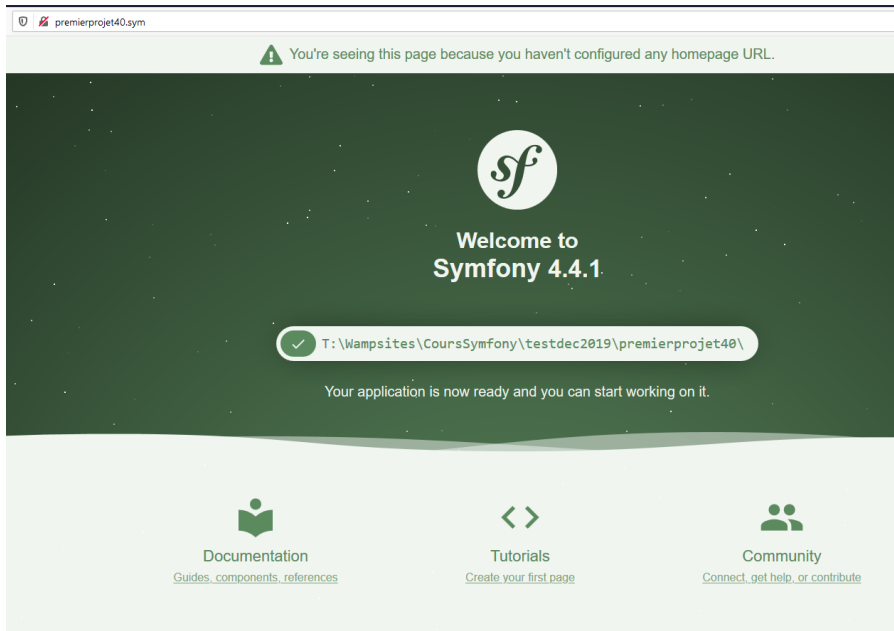
Redémarrer le DNS.



Dans la page <http://localhost>, vous devez voir votre virtualhost

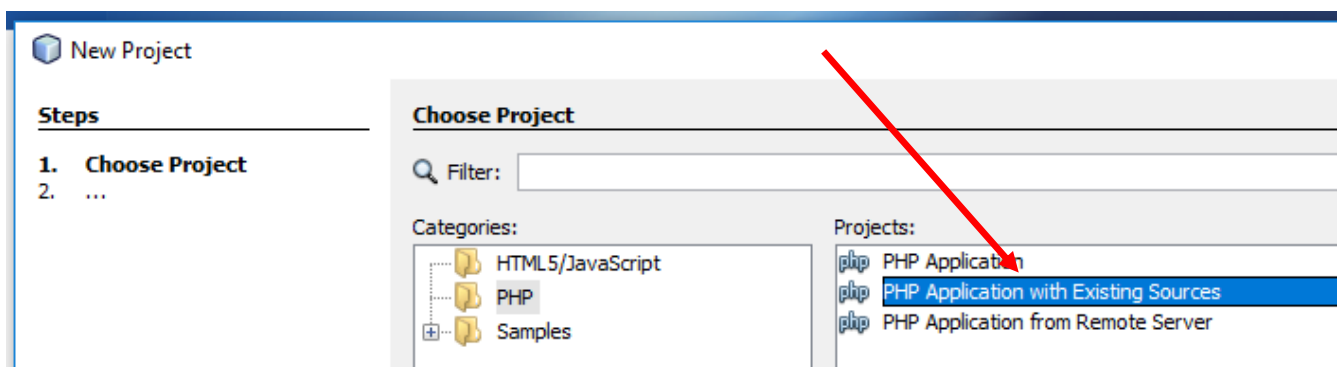
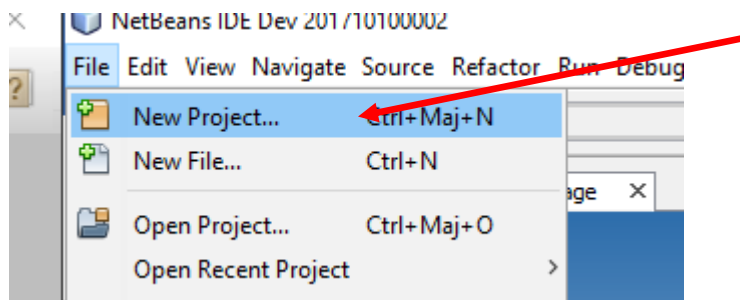


Testez :



Partie 3 : PROJET NETBEANS

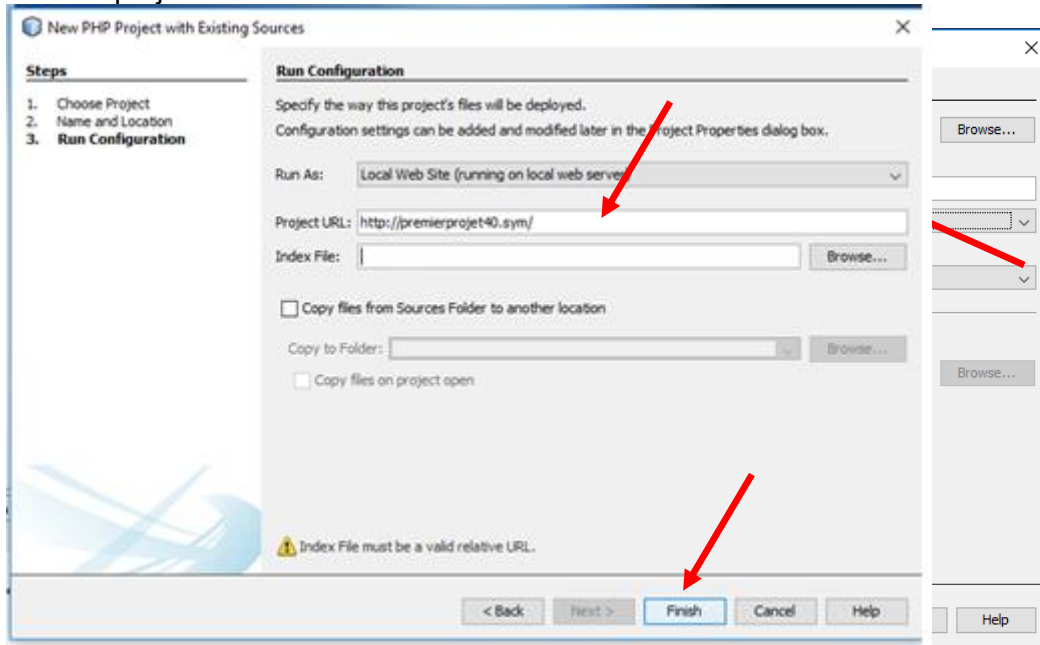
L'idée est de créer un nouveau projet ... **Avec les sources existantes**
Ouvrez Netbeans et créez un nouveau projet



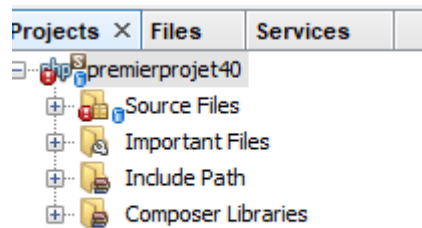
Choisir un projet php **AVEC APPLICATION EXISTING SOURCES**

Indiquez :

- ✓ le chemin du projet.
- ✓ le nom du projet :



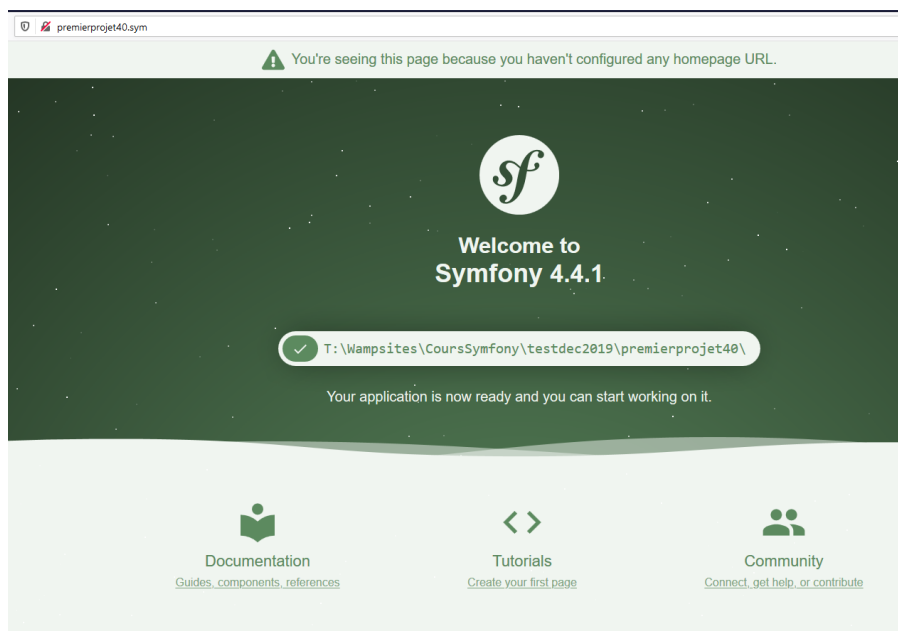
Notre projet a bien été créé :



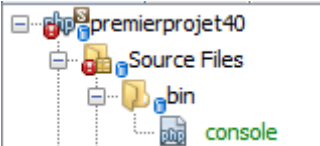
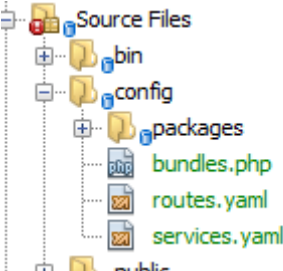
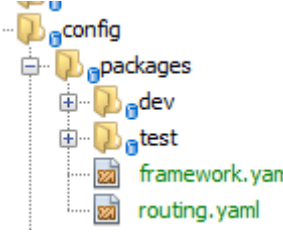
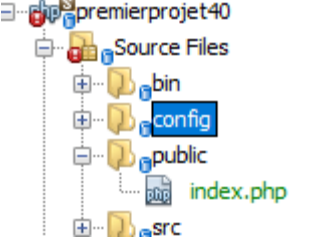

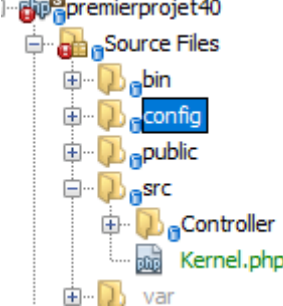

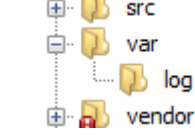
Testez :

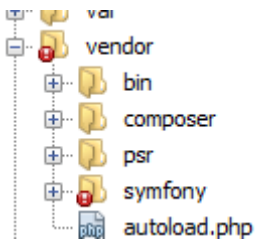


Ça fonctionne :



Organisation du projet

	<p>Le dossier <i>bin</i>, il l'exécutable <i>console</i> qui va permettre d'entrer les commandes symfony de type <i>php bin/console xxx</i></p>
	<p>Le dossier config qui va contenir l'ensemble des fichiers de configuration. Ils sont au format yaml.</p> <p>On trouve de base le fichier routing.yaml : routing principal de l'application services.yaml gère l'Injection de dépendance bundles.php : va contenir l'ensemble des bundles nécessaires à l'application</p>
	<p>Le dossier config/packages qui contient les fichiers de configuration par package.</p> <p>framework.yaml : Fichier de configuration du framework</p> <p>Dans ce dossier on trouvera tous les fichiers de configuration des packages que l'on installera (ex : twig.yaml) Les dossiers dev et tests contiennent les fichiers spécifiques aux environnements (test ou dev).</p>
	<p>Le dossier public est le dossier racine à l'application web, il contient tous les fichiers desservis par les serveurs HTTP</p> <p> Il n'y a plus les fichiers app.php et app_dev.php</p> <p>C'est le dossier pointé par notre virtualHost</p>
	<p>Le dossier src contenant la structure de notre application.</p> <p> A noter qu'on n'a plus les bundles comme en symfony 3.x</p> <p>Notre application ne sera plus découpée en bundle et l'espace de nom racine est app</p>
	<p>Le dossier var qui va contenir les dossiers cache et log pour le cache symfony et les différents logs renvoyés par l'application.</p>

	<p>Le dossier vendor qui contiendra toutes les ressources récupérées par composer.</p>
---	--

Partie 4 : CREATION DE LA PREMIERE PAGE

Pour cette première page, on va faire très simple !!!

Vous allez afficher un message de bienvenue dans une page HTML !

Dans cet exercice vous allez :

- ✓ Créer un contrôleur
- ✓ Créer une action,
- ✓ Créer une route au format yaml
- ✓ Afficher un texte basique

1. Création du contrôleur

Un contrôleur est une classe dans le dossier src/Controller



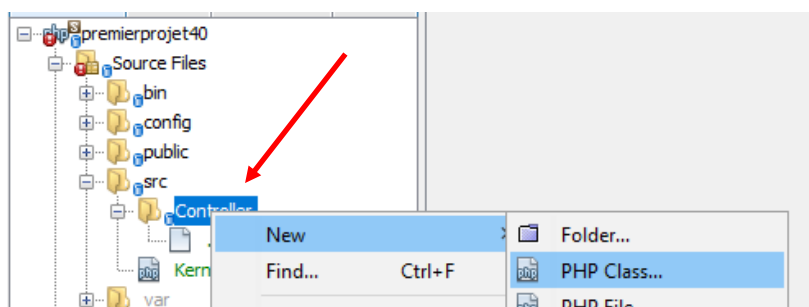
Le nom du contrôleur est TOUJOURS de la forme **SonNomController**

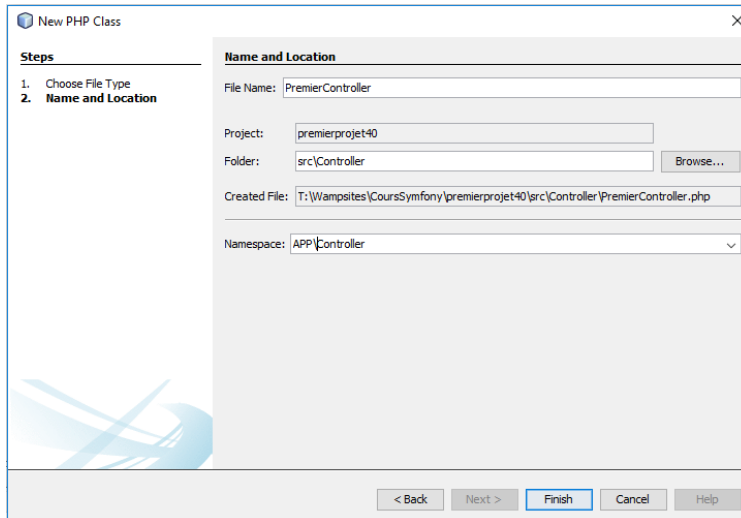
Votre contrôleur se nommera : PremierController

Espace de nom : App\Controller

En effet, il est directement placé dans le dossier App

Click bouton droit sur le dossier src/Controller





2. Création d'une action (méthode)

Vous modifierez le code généré pour arriver à ceci :

```
<?php
namespace App\Controller;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;


class PremierController {

    /**
     * @Route("/index",name="index")
     */
    public function index() {
        return new Response($content = 'Bonjour mon premier contrôleur');
    }

}
```

3. Création d'une route

Pour pouvoir travailler sur les routes en annotation, vous allez exécuter la commande suivante dans la console, **dans le dossier racine du projet** :

 Invite de commandes

```
.\Wampsites\CoursSymfony\testdec2019\premierprojet40>composer require annotation
```

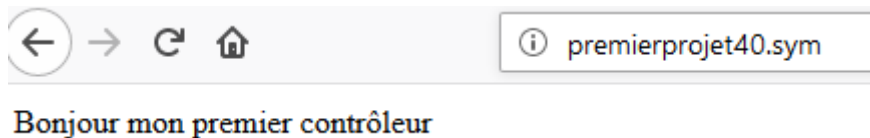
Nous verrons plus tard pourquoi.

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

C'est donc ici que l'on va associer une URL à une action (méthode)
... à condition d'avoir indiqué le bon espace de noms pour les annotations de routing :
use Symfony\Component\Routing\Annotation\Route;



Sauvegardez vos fichiers et entrez l'url : <http://premierprojet40.sym/index>
Vous obtenez ceci ... en principe



On vient de voir seulement dans cette partie comment associer une URL (route) à une action (méthode) d'une classe Controller.



On peut faire beaucoup mieux !!!

Partie 5 : CREATION D'UNE DEUXIEME PAGE

Dans cette partie, on va dans cette partie découvrir plusieurs choses :

- ✓ Utiliser composer et flex
- ✓ Charger les bundle twig, annotation (déjà fait), maker-bundle, profiler et apache-pack
- ✓ Créer un deuxième contrôleur
- ✓ Créer une nouvelle action avec routage par annotation
- ✓ Générer une page twig et l'afficher

1. Utiliser composer et flex

Flex est un plugin composer (une surcouche) qui permet de gérer plus facilement les dépendances.

Flex améliore l'installation des bundles en automatisant certaines tâches comme installer ou désinstaller les bundles et la gestion de Composer. Flex apporte notamment la possibilité d'exécuter et automatiser des tâches avant et après l'exécution d'une tâche Composer. Ainsi Flex apporte la possibilité, en plus de la gestion de l'installation des librairies et bundles, de pouvoir les configurer automatiquement.... Ceux qui ont travaillé sur les anciennes versions de symfony comprendront bien !!!

L'ensemble de ces tâches et fonctionnalités sont décrits sous forme de recette. On peut par exemple retrouver l'ensemble des **recettes symfony** pour les paquets Composer gérés par l'équipe en charge du Core de Symfony.

Vous pouvez trouver ces recettes (Recipies) ici <https://flex.symfony.com/>

All Symfony Recipes available

ad3n/ratchet-bundle <div>contrib</div> Package details Recipe	adback/adback-sdk-php-symfony <div>contrib</div> Package details Recipe	ajardin/docker-symfony <div>contrib</div> Package details Recipe	algolia/algolia-search-bundle <div>contrib</div> Package details Recipe
algolia/search-bundle <div>contrib</div> Package details Recipe	andchir/omnipay-bundle <div>contrib</div> Package details Recipe	anezi/locale-extension <div>contrib</div> Package details Recipe	api-platform/admin-pack <div>official</div> Aliases: api-admin api-platform-admin react-admin Package details Recipe
api-platform/api-pack <div>official</div> Aliases: api api-platform Package details Recipe	api-platform/core <div>official</div> Package details Recipe	artprima/prometheus-metrics-bundle <div>contrib</div> Package details Recipe	atlas/symfony <div>contrib</div> Package details Recipe



On remarque qu'il y a les recettes officielles et celles provenant de contributeurs.

Pour charger une dépendance flex, on utilise la **commande composer en ligne de commande depuis le dossier racine de l'application** :

Composer require monrecipient

```
C:\> Administrateur : Invite de commandes

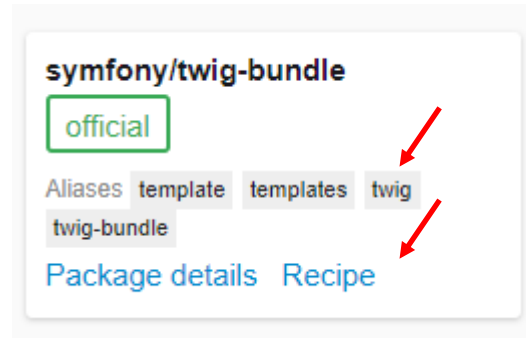
T:\Wampsites\CoursSymfony\premierprojet40>composer require monrecipient
```

2. Charger les bundle twig, annotation, maker-bundle et profiler et apache-pack

On va charger les bundles (recipies) suivants :

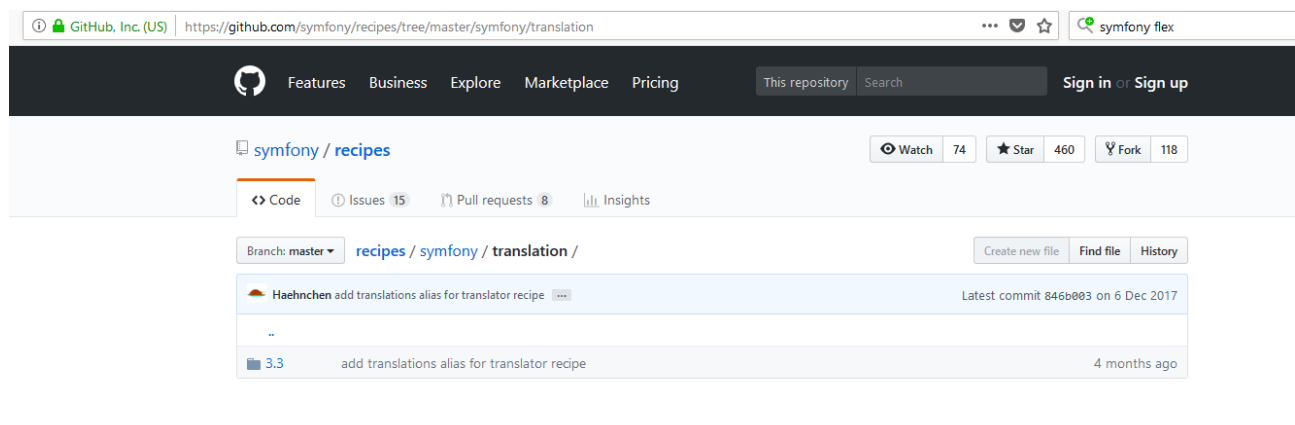
- ✓ **twig** : afin de pouvoir utiliser le moteur de template traditionnellement associé à symfony
- ✓ **annotation** : afin de pouvoir travailler sur l'ensemble de notre projet avec les annotations. Ce recipe est en principe chargé à ce stade du TD.
- ✓ **maker-bundle** : pour pouvoir nous aider à créer des contrôleurs, des classes de formulaires, des tests et bien plus encore pour nous éviter à avoir à écrire trop de code rébatif et répétitif.
- ✓ **profiler** : il s'agit de l'outil qui affichera, en mode "dev" une barre d'outil en bas de la page qui nous servira notamment au niveau du débogage.(ex app_dev.php)
- ✓ **apache-pack** : il s'agit d'un recipe qui va permettre de prendre en charge la réécriture d'URL's. A utiliser si on travaille avec Apache et non le serveur web intégré à PHP

le principe : prenons l'exemple de twig, on peut voir sur le site <https://flex.symfony.com/>

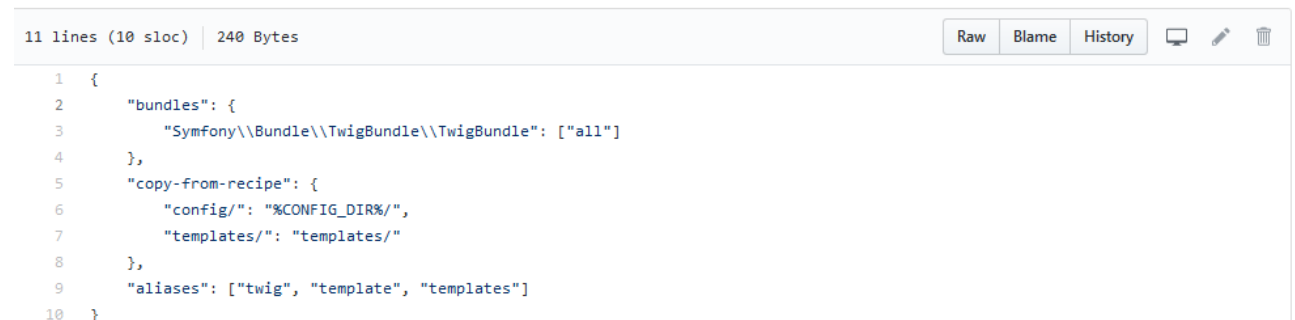


On utilisera un des alias pour le charger. (exemple : twig)

Si on clique sur *Recipe*, on se retrouve sur le dépôt



Et en cliquant sur la version (3.3 ici), on peut trouver le manifeste (manifest.json) d'installation :

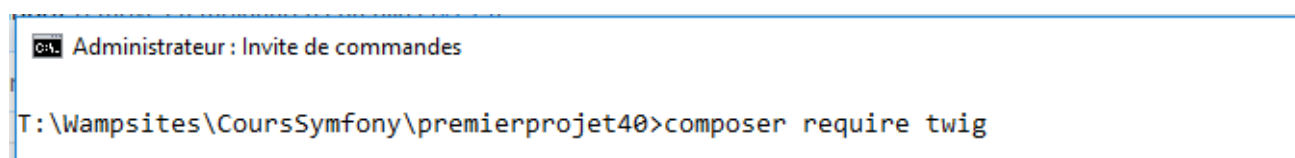


Il va nous simplifier la vie ... les "anciens de symfony 3.x comprennent !!!

Chargement du bundle twig :



Attention à lancer la commande depuis le répertoire racine de l'application



Et le bundle, ainsi que ses dépendances vont se charger, des fichiers seront ajoutés au projet.

```

C:\> Administrateur : Invite de commandes

T:\Wampsites\CoursSymfony\premierprojet40>composer require twig
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)

Prefetching 3 packages
- Downloading (100%)

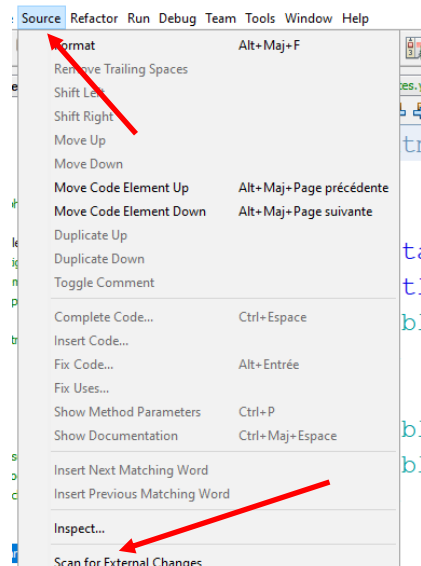
Package operations: 3 installs, 0 updates, 0 removals
- Installing twig/twig (v2.4.7): Loading from cache
- Installing symfony/twig-bridge (v4.0.6): Loading from cache
- Installing symfony/twig-bundle (v4.0.6): Loading from cache
Writing lock file
Generating autoload files
Symfony operations: 1 recipe (b247dcce8bc41359807b50dfc10a413d)
- Configuring symfony/twig-bundle (>=3.3): From github.com/symfony/recipes:master
Executing script cache:clear [OK]
Executing script assets:install --symlink --relative public [OK]

Some files may have been created or updated to configure your new packages.
Please review, edit and commit them: these files are yours.

T:\Wampsites\CoursSymfony\premierprojet40>
    
```

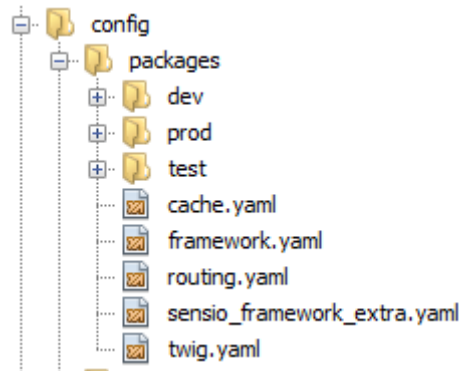


Pour afficher les dossiers et fichiers créés à l'extérieur de Netbeans, il est nécessaire de rafraîchir les sources du projet en faisant Source/Scan for External Changes :

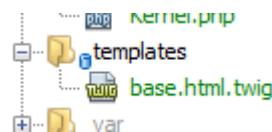


On voit

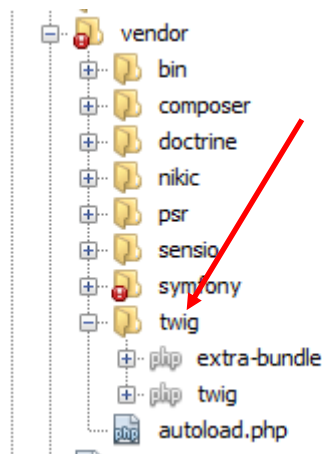
- ✓ qu'un fichier twig.yaml a été créé dans config/packages :



- ✓ qu'un dossier templates a été créé et il contient le fichier base.html.twig




- ✓ que le bundle twig a été placé dans le dossier vendor/symfony




Vous remarquerez qu'un fichier autoload a aussi été créé... vous pouvez l'ouvrir, vous apprendrez peut-être des choses....


Vous allez maintenant charger les recettes annotations maker-bundle et apache-pack

 Invite de commandes

~~T:\Wampsites\CoursSymfony\testdec2019\premierprojet40>composer require annotation~~

 Administrateur : Invite de commandes

T:\Wampsites\CoursSymfony\premierprojet40>composer require maker-bundle

 Administrateur : Invite de commandes

T:\Wampsites\CoursSymfony\premierprojet40>composer require profiler

CA: Administrateur : Invite de commandes

T:\Wampsites\CoursSymfony\premierprojet40>composer require symfony/apache-pack

CA: Administrateur : Invite de commandes - composer require symfony/apache-pack

```
T:\Wampsites\CoursSymfony\premierprojet40>composer require symfony/apache-pack
Using version ^1.0 for symfony/apache-pack
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
 - Installing symfony/apache-pack (v1.0.1): Loading from cache
Writing lock file
Generating autoload files
Symfony operations: 1 recipe (a4c074762e8c2bf76716dcf0699855f5)
 - WARNING symfony/apache-pack (>=1.0): From github.com/symfony/recipes-contrib:master
   The recipe for this package comes from the "contrib" repository, which is open to community contributi
   Review the recipe at https://github.com/symfony/recipes-contrib/tree/master/symfony/apache-pack/1.0

Do you want to execute this recipe?
[y] Yes
[n] No
[a] Yes for all packages, only for the current installation session
[p] Yes permanently, never ask again for this project
(defaults to n):
```

Répondre p

Allez voir le fichier config\bundles.php :

<?php

```
return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
    Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],
    Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
    Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],
];
```

On voit que les bundles téléchargés ont été référencés dans l'application !!!



Dans les versions précédentes de symfony, on devait ajouter ces lignes à la main !!!

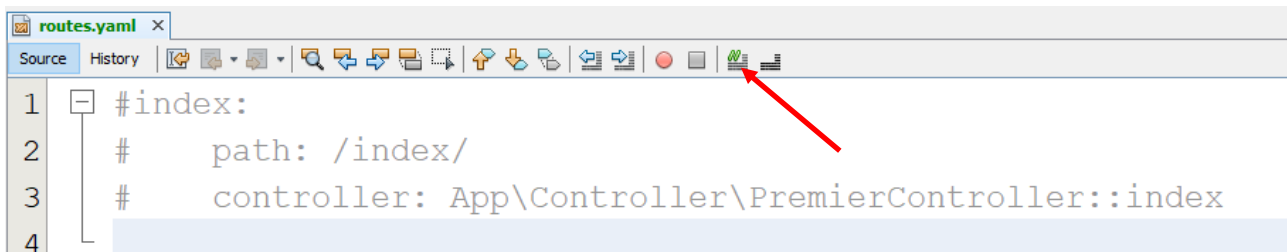


3. Créer un deuxième contrôleur



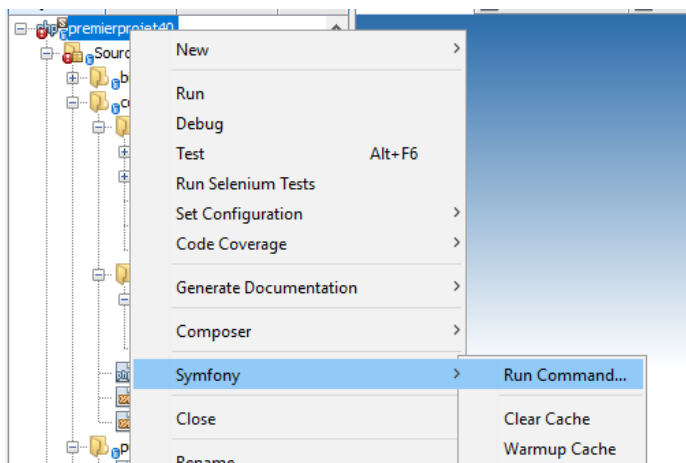
Cet exercice est purement pédagogique et pas fonctionnel, il nous permet seulement de faire le tour des principales fonctionnalités de Symfony.

Avant de commencer, on va commenter l'ensemble du contenu du fichier config/routes.yaml :



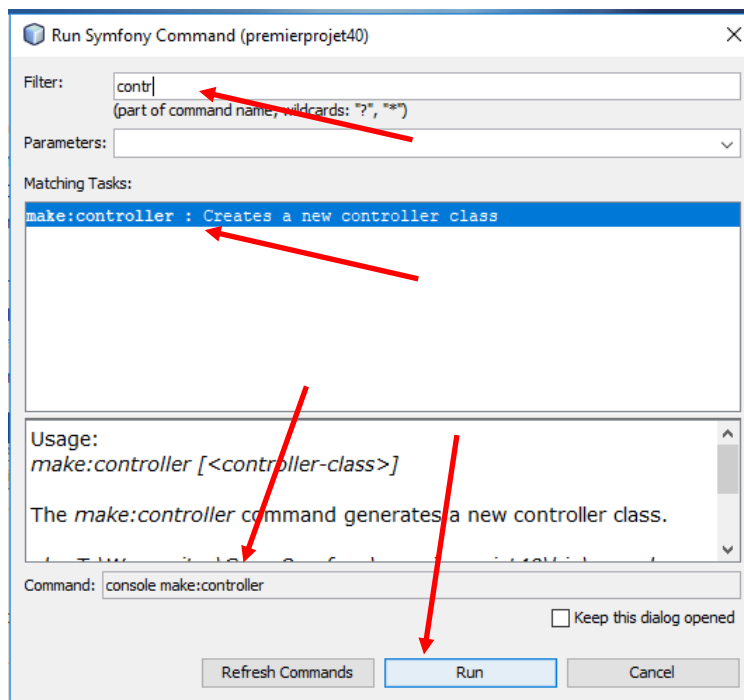
On va créer un deuxième contrôleur que l'on va appeler **PrincipalController**. Mais au lieu de le créer comme dans la partie précédente, on va le créer au moyen d'une commande Symfony que l'on peut désormais utiliser grâce au bundle Maker-Bundle que l'on vient d'installer.

Vous allez utiliser les commandes Symfony accessibles directement depuis Netbeans :



Dans le filtre de votre interface, commencez à saisir le mot *controller* : il vous sera proposé quelques commandes. Celle qui nous intéresse est

Make:controller

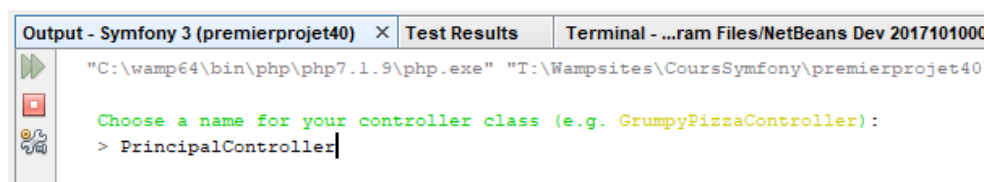


On aurait aussi pu rentrer cette commande par la console en étant positionné sur le dossier racine de notre application :

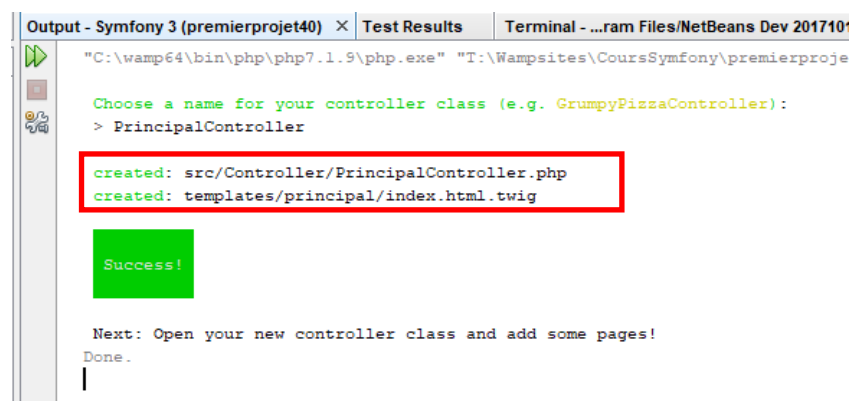
```
C:\> Administrateur : Invite de commandes
T:\Wampsites\CoursSymfony\premierprojet40>php bin\console make:controller
```

Vous choisirez la méthode qui vous convient le mieux.
Je continue sur la fenêtre Netbeans

Vous aurez à saisir le nom du controller

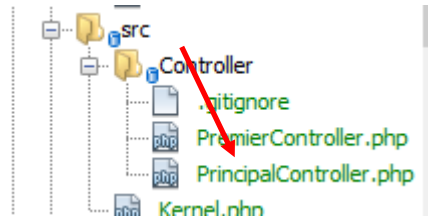


Après avoir cliqué sur Enter, votre controller sera créé...

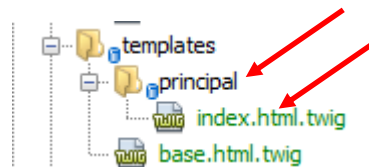


Mais pas QUE

Voyez dans le dossier src/Controller, vous voyez la classe Controller qui a été créée :



Mais aussi dans le dossier templates, il y a un dossier principal (nom du contrôleur créé) et un fichier index.html.twig :



On verra ceci un peu plus loin.

Ouvrez le fichier PrincipalController.php .

Vous verrez :

- ✓ Que cette classe hérite de la classe `Symfony\Bundle\FrameworkBundle\Controller\Controller`
- ✓ qu'il a créé une action (méthode) `index`
- ✓ que la route est faite automatiquement par annotation
- ✓ que cette fonction `index` est appelée par la route `/principal`
- ✓ et qu'elle appelle la vue `template/principal/index.html.twig`...

```
<?php
```

```
namespace App\Controller;
```

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

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

```
class PrincipalController extends AbstractController
```

```
{
```

```
    /**
```

```
     * @Route("/principal", name="principal")
```

```
     */
```

```
    public function index()
```

```
    {
```

```
        return $this->render('principal/index.html.twig', [
```

```
            'controller_name' => 'PrincipalController',
```

```
        ]);
```

```
    }
```

```
}
```

Testez cette URL : <http://premierprojet40.sym/principal> :

Et voici le résultat :



Que s'est-il passé ?

- ✓ Cette action a été "matchée" par l'URL et s'est donc exécutée
- ✓ Elle a appelé la vue twig *principal/index.html.twig*
- ✓ En lui passant un tableau de variables ne contenant qu'une seule variable :

Clé : `controller_name` ; valeur : `PrincipalController`

```
class PrincipalController extends AbstractController
{
    /**
     * @Route("/principal", name="principal")
     */
    public function index()
    {
        return $this->render('principal/index.html.twig', [
            'controller_name' => "Symfony, c'est super",
        ]);
    }
}
```

Le code de cette vue twig :

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

{% block title %}Hello {{ controller_name }}!{% 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>src/Controller/PrincipalController.php</code></li>
    <li>Your template at <code>templates/principal/index.html.twig</code></li>
  </ul>
</div>
{% endblock %}
```

Ce bloc va

- ✓ Hériter de la vue base.html.twig
- ✓ Redéfinir les balises title et body

Le code de la vue base.html.twig :

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



Exercice : Modifier le code pour obtenir ceci :

Hello Symfony c'est super!

This friendly message is coming from:

- Your controller at `src/Controller/PrincipalController.php`
- Your template at `templates/principal/index.html.twig`

Partie 6 : NOUVELLE PAGE DE BIENVENUE

Vous allez maintenant créer une page qui affiche une valeur récupérée dans l'URL

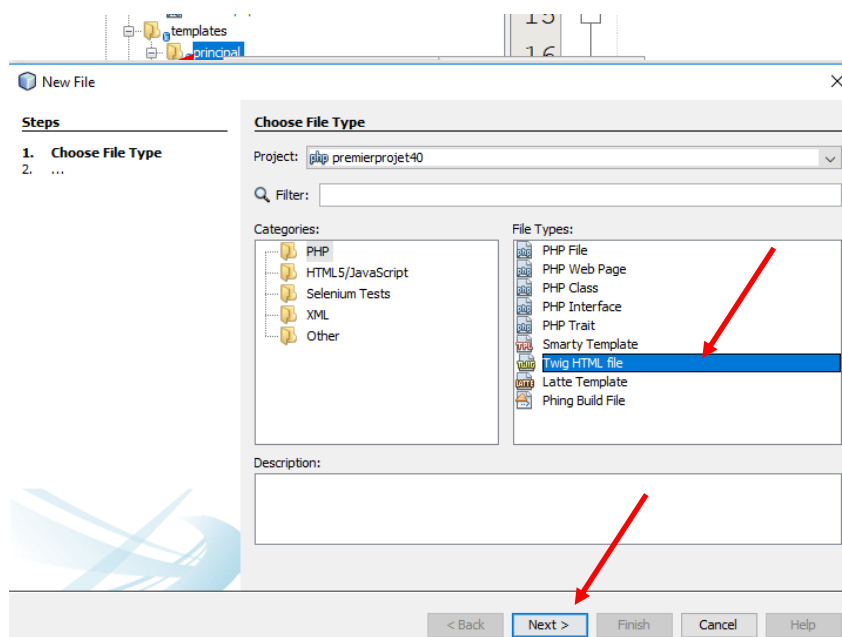
Exemple d'URL :

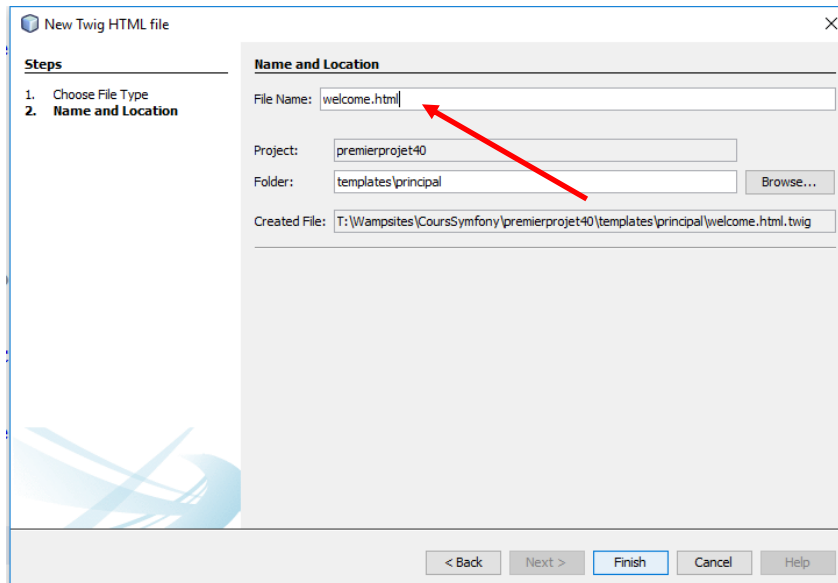
<http://premierprojet40.sym/welcome/Benoît>

Dans le contrôleur `PrincipalController`, créez une nouvelle action appelée `welcome` et qui admet une chaîne de caractères en paramètre.

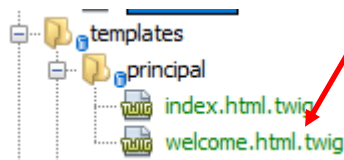
```
/**
 * @Route("/welcome/{nom}")
 */
public function welcome($nom) {
    return $this->render('principal/welcome.html.twig', array(
        "nom" => $nom
    ));
}
```

Dans le dossier `templates/principal`, créez une vue `welcome.html.twig` :





Voici :



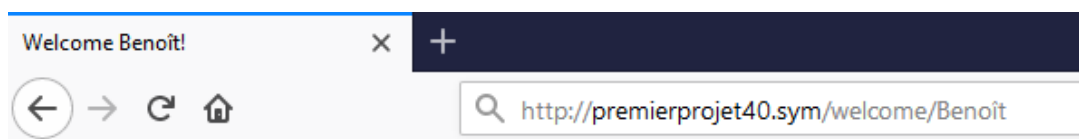
Ouvrez ce fichier et complétez avec ce code :

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

{% block title %}Welcome {{ nom }}!{% endblock %}

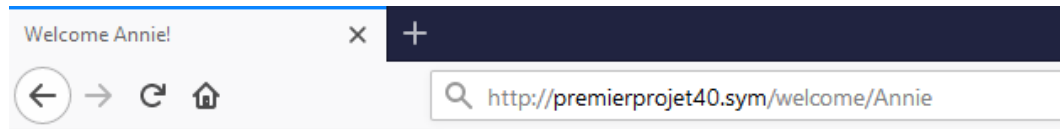
{% block body %}
    <h1>welcome {{ nom }}! </h1>
{% endblock %}
```

Exécutez :



welcome Benoît!

Avec cette URL: <http://premierprojet40.sym/welcome/Annie>
on obtient :



welcome Annie!

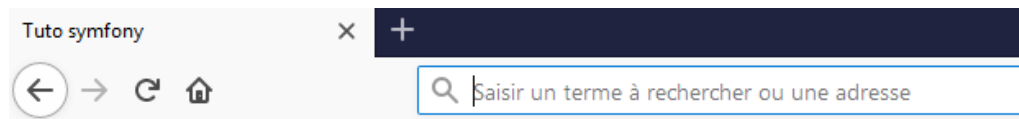
Vous remarquerez aussi en bas de la page, la barre d'outils qui s'affiche grâce au bundle profiler



Elle vous aidera bien par la suite

Partie 7 : EXERCICE

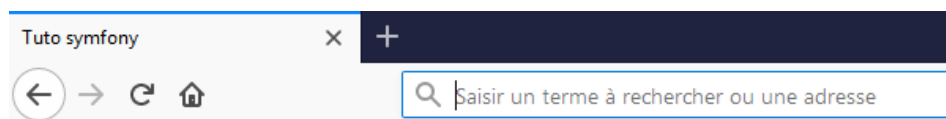
Créer une nouvelle action dans le contrôleur PrincipalController qui réponde à l'URL <http://premierprojet40.sym/??> et qui affiche ce que vous voulez ... vous passerez au moins 2 paramètres à la vue !



Vous êtes né dans le département : 33!

Vous êtes un garçon!

nous sommes le : Thursday 22 March 2018 23:45



Vous êtes né dans le département : 95!

Vous êtes une fille!

nous sommes le : Thursday 22 March 2018 23:46

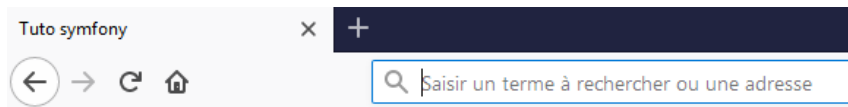
Partie 8 : AFFICHER DES DONNEES



Maintenant on va faire une liaison avec les bases de données.

Et comme on va relier nos données avec des formulaires, on va installer les bundles suivants (ainsi que leurs dépendances) :

- ✓ Form : pour générer les formulaires,
- ✓ Validator : pour valider les formulaires... sans écrire trop de code



Vous êtes né dans le département : 33!

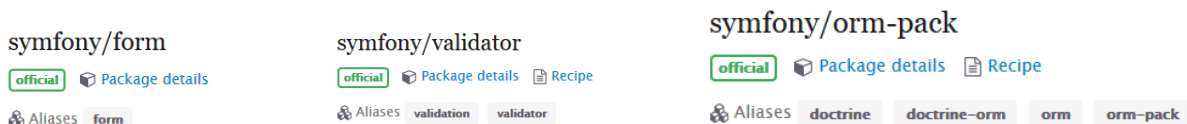
Vous êtes de sexe inconnu !!

nous sommes le : Thursday 22 March 2018 23:46

- ✓ Orm : pour la persistance des objets (doctrine)

Allez voir sur <https://flex.symfony.com/>

Et vous trouverez tous les bundles voulus :

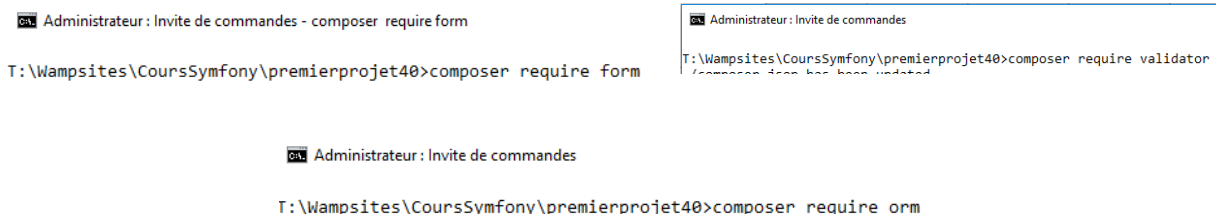


Donc vous pouvez lancer les commandes :

- ✓ `Composer require form`
- ✓ `Composer require validator`
- ✓ `Composer require orm`



Dans la fenêtre de commandes et dans le dossier source de l'application



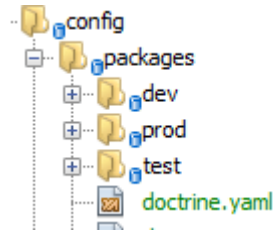
Vérifiez ensuite que vos bundles sont installés :

- ✓ Form et validator dans le dossier vendor/symfony
- ✓ Orm (doctrine) dans le dossier vendor

Vous allez maintenant configurer la base de données.

Chaque bundle fournit son fichier de configuration présent dans config/packages/unbundle.yaml

Pour doctrine, ouvrez le fichier config/packages/doctrine.yaml :



Si vous ne voyez pas ces fichiers après l'installation des bundles, n'oubliez pas de faire dans votre IDE source\Scan for external changes...

Commentez cette ligne :

With Symfony 3.3, remove the `resolve:` prefix

```
#url: '
    # With Symfony 3.3, remove the `resolve:` prefix
    #url: '%env(resolve:DATABASE_URL)%'
    dbname: '%env(DATABASE_NAME)%'
    host: 'localhost'
    user: '%env(DATABASE_USER)%'
    password: '%env(DATABASE_PWD)%'
```

orm:

Et rajoutez ces paramètres dans la partie DBAL :

Symfony stocke ses paramètres dans le fichiers : .env

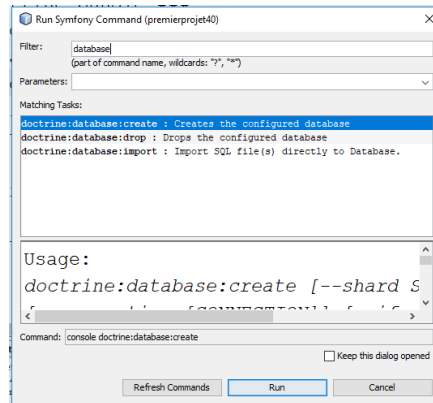
Rajoutez les paramètres suivants au fichier .env:

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connec
# For an SQLite database, use: "sqlite:///kernel.project_dir/var/data.db"
# For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7
DATABASE_USER=root
DATABASE_PWD=""
DATABASE_NAME=dbpremiersymfo40
###< doctrine/doctrine-bundle ###
```

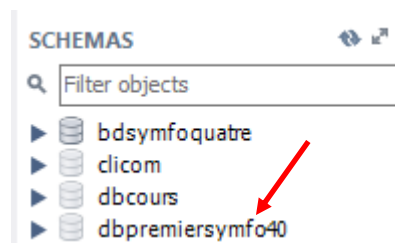
Les paramètres des fichiers .yaml iront chercher leurs valeurs dans ces fichiers :

```
DATABASE_USER=root
DATABASE_PWD=""
DATABASE_NAME=dbpremiersymfo40
```

Il ne reste plus qu'à créer la base avec la commande `doctrine:database:create`

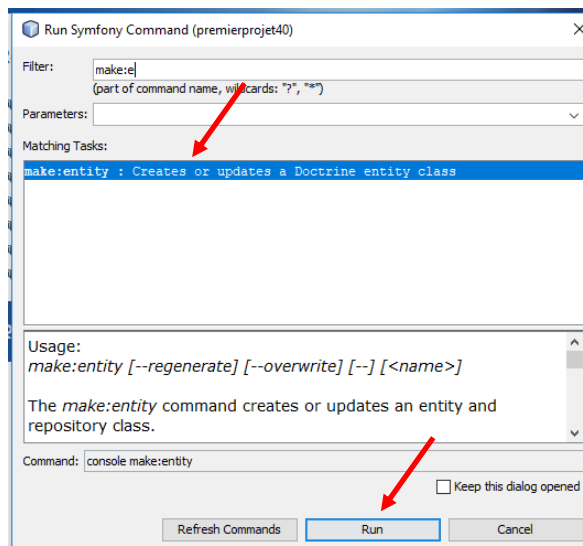


La base de données a été créée :



Vous allez maintenant créer une entity.
Une Entity est une classe qui sera mappée sur une table de la BD.

Vous allez utiliser la commande `make:entity`



Cette commande va créer une classe métier, il vous sera demandé de créer les attributs de ces classes **SAUF l'id qui sera créé automatiquement**

Respectez aussi la casse !!! Norme PSR-4

Vous créerez 2 attributs :

- ✓ Nom : chaîne de 50 caractères not null
- ✓ Salaire de type décimal 9 chiffres dont 2 décimales not null

Renseignez le nom :

```
"C:\wamp64\bin\php\php7.1.9\php.exe" "T:\Wampsites\CoursSymfony\premierprojet40\bin\console" "--ansi" "make:entity"

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

created: src/Entity/Employe.php
created: src/Repository/EmployeRepository.php

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

New property name (press <return> to stop adding fields):
> |
Field type (enter ? to see all types) [string]:
>

Field length [255]:
> 50

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

updated: src/Entity/Employe.php

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

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

Precision (total number of digits stored: 100.00 would be 5) [10]:
> 9

Scale (number of decimals to store: 100.00 would be 2) [0]:
> 2

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

updated: src/Entity/Employe.php

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



Ça doit vous rappeler des choses ...

Faites Enter pour sortir du mode édition.

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

Success!

Next: When you're ready, create a migration with `make:migration`

```
Done.
|
```

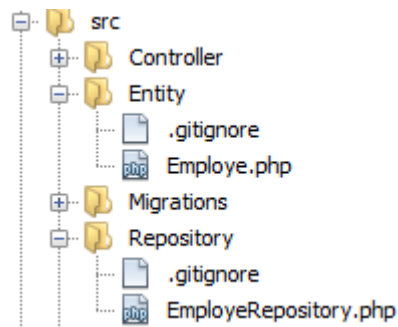
Dans les messages vous avez vu que les fichiers `Employe.php` et `EmployeRepository.php` ont été créés.

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

created: src/Entity/Employe.php
created: src/Repository/EmployeRepository.php

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

2 nouveaux dossiers créés pour stocker les fichiers créés :



- ✓ **Employe.php** : contient la classe `Employe` avec les annotations permettant de faire le mapping sur la future table `Employe`
- ✓ **EmployeRepository.php** : qui contiendra la partie traitements correspondants à la classe `Employe`.

Les attributs et les méthodes se trouvent bien dans 2 classes distinctes... On en reparlera.
Allez voir le fichier `Employe.php` :

```
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

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

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $nom;
```

Généré automatiquement

On y voit les annotations et les annotations de mapping ainsi que l'annotation indiquant le fichier de repository correspondant :

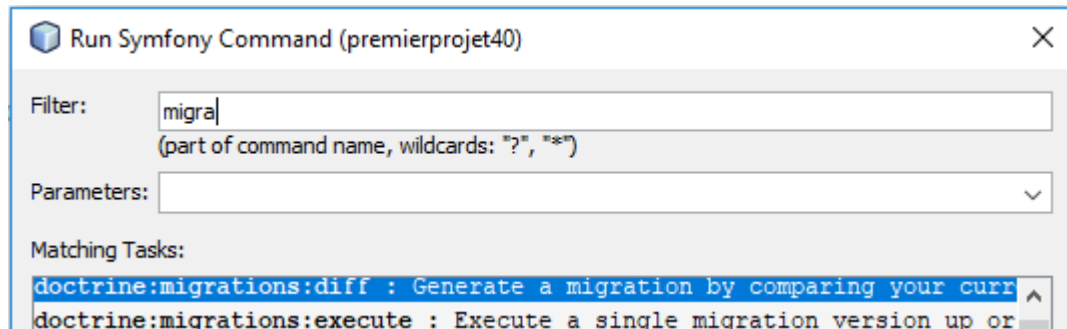
```
/**
 * @ORM\Entity(repositoryClass="App\Repository\EmployeeRepository")
 */
```

On pourra ainsi créer d'autres Entities.

Vous allez maintenant mettre à jour la BD en deux étapes :

- ✓ générer les ordres SQL de synchronisation des tables par rapport aux entities.
- ✓ commande doctrine:migrations:diff
- ✓ Exécuter ces ordres
- ✓ commande doctrine:migrations:migrate

doctrine:migrations:diff



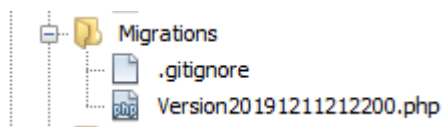
La génération du script de mise à jour s'est bien passée :

```
"C:\wamp64\bin\php\php7.3.5\php.exe" "T:\Wampsites\CoursSymfony\testdec2019\premierprojet40\bin\console" "--ansi" "doctrine:migrations:diff"
Generated new migration class to "T:\Wampsites\CoursSymfony\testdec2019\premierprojet40\src\Migrations\Version20191211212200.php"

To run just this migration for testing purposes, you can use migrations:execute --up 20191211212200

To revert the migration you can use migrations:execute --down 20191211212200
Done.
```

Et on peut voir le fichier généré des modifications :



Vous pouvez aller voir le contenu de ce fichier :

Il contient les ordres de suppression de la table et de création.

En effet, la table n'existant pas encore, on peut la supprimer et la recréer ... sans incidence sur son contenu !!!

```
<?php

declare(strict_types=1);

namespace Doctrine\Migrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20191211212200 extends AbstractMigration
{
    public function getDescription() : string
    {
        return '';
    }

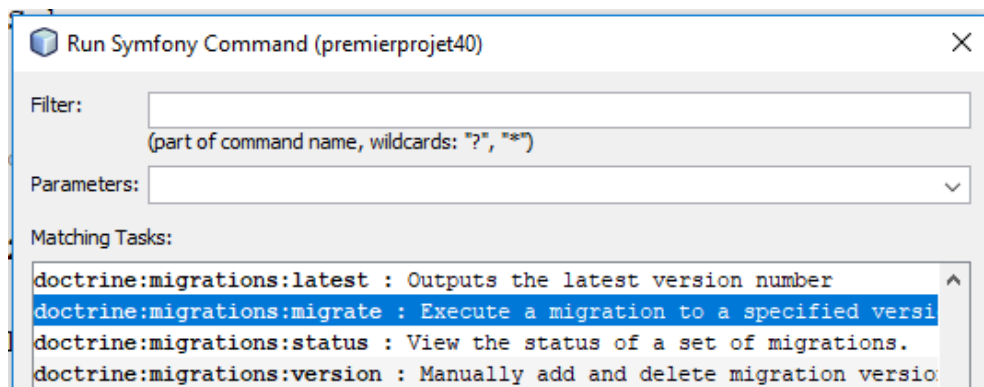
    public function up(Schema $schema) : void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() !== 'mysql', 'Migration can only be executed safely on \'mysql\'');

        $this->addSql('CREATE TABLE employe (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(50) NOT NULL, salaire NUMERIC(9, 2) NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci');
    }

    public function down(Schema $schema) : void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() !== 'mysql', 'Migration can only be executed safely on \'mysql\'');

        $this->addSql('DROP TABLE employe');
    }
}
```

Exécutez maintenant la commande doctrine:migrations:migrate :



N'oubliez pas de répondre par "y" à la demande de confirmation d'exécution du script :

```
Y
Migrating up to 20191211212200 from 0
```

On voit que le fichier de migration a été exécuté et donc l'ordre SQL de création de la table :

```
"C:\wamp64\bin\php\php7.3.5\php.exe" "T:\Wampsites\CoursSymfony\testdec2019\premierprojet40\bin\console" "--ansi" "doctrine:migrations:migrate"

Application Migrations

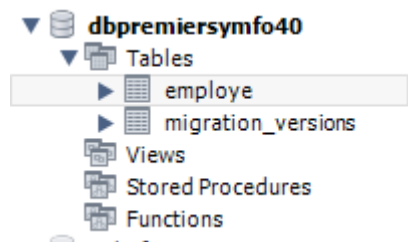
WARNING! You are about to execute a database migration that could result in schema changes and data loss. Are you sure you wish to continue? (y/n)y
Migrating up to 20191211212200 from 0

++ migrating 20191211212200
--> CREATE TABLE employe (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(50) NOT NULL, salaire NUMERIC(9, 2) NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci ENGINE = InnoDB
++ migrated (took 94.6ms, used 14M memory)

-----

++ finished in 99.7ms
++ used 14M memory
++ 1 migrations executed
++ 1 sql queries
Done.
```

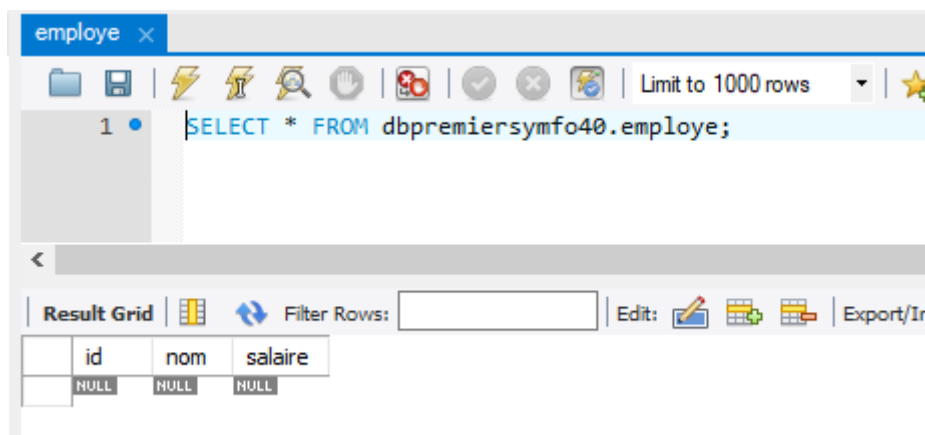
Ouvrez votre client mysql : MysqlWorkbench ou phpMyAdmin et constatez que la table a été créée dans la bonne BD :



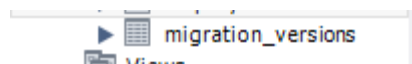
Exécutez l'ordre SQL

```
select * from dbpremiersymfo40.employe
```

et constatez que la table ... est vide !!!



Notez aussi la création de la table des migrations :



Exécutez l'ordre SQL :

```
SELECT * FROM dbpremiersymfo40.migration_versions;
```

Vous voyez le contenu :

	version	executed_at
	20191211212200	2019-12-11 21:26:16
*	NULL	NULL

La première ligne correspond au nom du fichier des migrations généré par la commande doctrine:migrations:diff

On y reviendra.



On va maintenant s'amuser un peu

Commencez par insérer quelques enregistrements dans la table employe :

```
insert into employe (nom, salaire) values
("Martin", 10000.00),
("Dupont", 23000.00),
("Russot", 45600.00),
("Delevoy", 12000.00),
("Dumans", 34000.00);
```

L'ordre a bien été exécuté :

6 22:38:53 select * from dbpremiersymfo40.employe LIMIT 0, 1000 5 row(s) returned

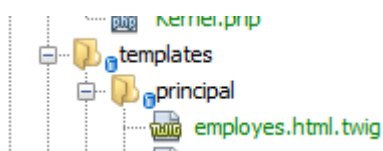
Vous allez maintenant créer la méthode suivante dans votre contrôleur PrincipalController :

```
/**
 * @Route("/employees", name="employees")
 * @param RegistryInterface $doctrine
 */
public function afficheEmployes(ManagerRegistry $doctrine) {
    $employees = $doctrine->getRepository(Employe::class)->findAll();
    $titre = "Liste des employés";
    return $this->render('principal/employees.html.twig', compact('titre', 'employees'));
}
```



Attention, peut-être qu'il vous manquera des clauses use !!!

Et vous créerez la vue :



Qui contient le code suivant :

```
{% extends "base.html.twig" %}
{% block title %}
    {{ titre }}
{% endblock %}
{% block body %}
<table>
    <thead>
        <tr>
            <th>id</th>
            <th>nom</th>
            <th>salaire</th>
        </tr>
    </thead>
    <tbody>
        {% for employe in employes %}







        <tr>
            <td>{{ employe.id }}</td>
            <td> {{ employe.nom }}</td>
            <td> {{ employe.salaire }}</td>
        </tr>
        {% endfor %}
    </tbody>

</table>

{% endblock %}
```

Essayez l'URL suivante : <http://premierprojet40.sym/employes>

Vous devriez obtenir ceci :

Liste des employés	x	+
   		 premierprojet40.sym/employes
id	nom	salaire
1	Martin	10000.00
2	Dupont	23000.00
3	Russot	45600.00
4	Delevoy	12000.00
5	Dumans	34000.00

Pas mal non ?



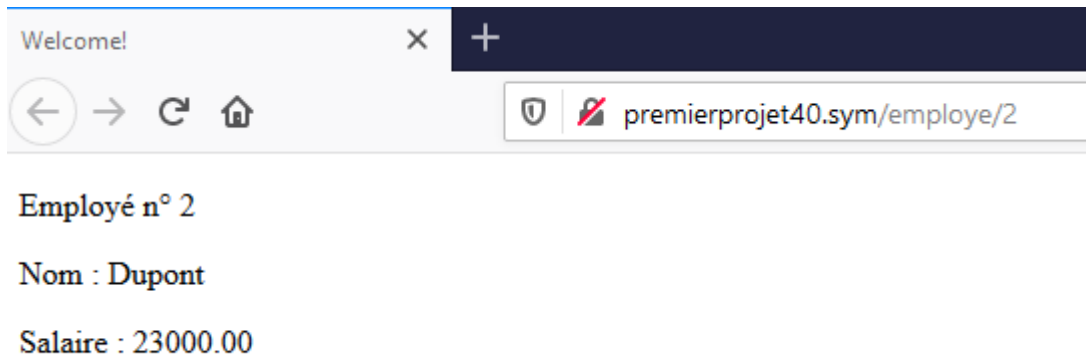


Exercice:

En saisissant cette URL : <http://premierprojet40.sym/employe/2>

Affichez les attributs de l'employé numéro 2

Vous utiliserez la méthode `find($id)` pour récupérer un enregistrement de la table par son id.



Solution (à vous de faire la bonne vue twig):

```
/**
 * @Route("/employe/{id}", name="unemploye" , requirements={"id":"\d+"})
 * @param ManagerRegistry $doctrine
 */
public function afficheUnEmploye(ManagerRegistry $doctrine, int $id) {
    $employe = $doctrine->getRepository(Employe::class)->find($id);
    $titre = "Employé n° " . $id;
    return $this->render('principal/unemploye.html.twig', compact('titre', 'employe'));
}
```

Partie 9 : LIER DES CLASSES

L'idée est de lier des classes... c'est-à-dire de mettre des objets d'une entity dans une autre entity.

On va considérer qu'un employé est affecté sur un lieu de travail.

Un lieu de travail est caractérisé par un id séquentiel, un nom, une ville.

La relation entre les deux Entities sera bidirectionnelle, c'est-à-dire que l'on pourra retrouver :

- ✓ le lieu d'affectation d'un employé
- et
- ✓ les employés affectés sur un lieu de travail

Pour cela, vous utiliserez les annotations doctrine.

Vous allez commencer par créer une Entity *Lieu* dans l'espace de nom `App\Entity` avec la commande `symfony Make:Entity`

```

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

created: src/Entity/Lieu.php
created: src/Repository/LieuRepository.php

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

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

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

Field length [255]:
> 60

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

updated: src/Entity/Lieu.php

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

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

Field length [255]:
> 60

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

updated: src/Entity/Lieu.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 `make:migration`

Done.

Dans l'entity Employé, rajoutez un objet \$lieu de la classe Lieu. (1 employe = 1 lieu)

```

/**
 * @ORM\ManyToOne(targetEntity="Lieu")
 */
private $lieu;

```

L'annotation indique le type de relation : 1 employe -> 1 lieu, 1 lieu ->n employes

Vous allez également générer le getter de l'attribut \$lieu :

```
}

function getLieu() {
    return $this->lieu;
}
```

On pourrait s'arrêter ici et mettre à jour la base de données.

On va toutefois construire une relation bi-directionnelle c'est-à-dire navigable dans les 2 sens.

Vous allez donc créer l'attribut \$lesEmployes dans la classe lieu. Ce sera la liste des employés travaillant sur un lieu :

Classe Lieu :

```
/**
 *
 * @ORM\OneToMany(targetEntity="Employe", mappedBy = "lieu")
 */
private $lesEmployes;
```

Vous allez générer le getter de cet attribut :

```
function getLesEmployes() {
    return $this->lesEmployes;
}
```

Et pour assurer le lien bi-directionnel, vous modifierez la relation ManyToOne de la classe Employe :

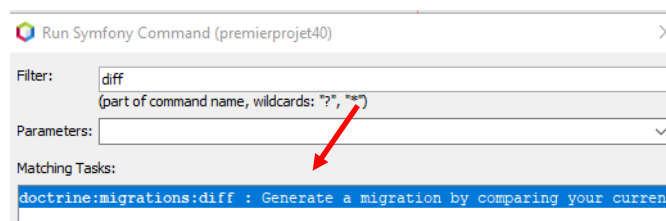
```
/**
 * @ORM\ManyToOne(targetEntity="Lieu", inversedBy="lesEmployes")
 */
private $lieu;
```

Les Use sur les classes Employe et Lieu dans les classe Lieu et Employe

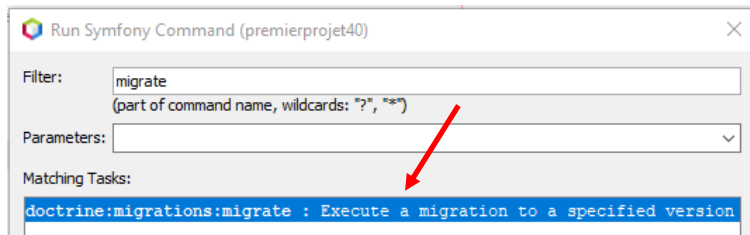


Il ne reste plus qu'à mettre à jour la base de données par les commandes symfony

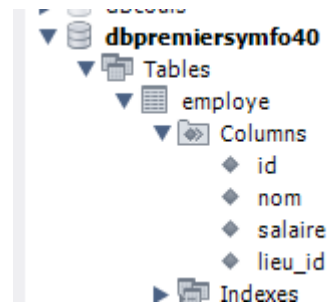
✓ doctrine:migration:diff



✓ doctrine:migration:migrate



Vous pouvez aller voir dans la table Employe, vous remarquerez la colonne clé étrangère lieu_id :



Et le contenu de la table :

	id	nom	salaire	lieu_id
▶	1	Martin	10000.00	NULL
	2	Dupont	23000.00	NULL
	3	Russot	45600.00	NULL
	4	Delevoy	12000.00	NULL
	5	Dumans	34000.00	NULL
*	NULL	NULL	NULL	NULL



Vous remarquez que toutes les valeurs de la clé primaire sont à NULL, ce n'est conforme au cahier des charges. Vous allez donc créer dans MysqlWorkbench des enregistrements dans la table Lieu :

	id	nom	ville
▶	1	Ets Blanchard	Toulon
	2	Société Dipassa	Paris
*	NULL	NULL	NULL

Et modifier la table Employe en affectant les employés sur un lieu :

	id	nom	salaire	lieu_id
	1	Martin	10000.00	1
	2	Dupont	23000.00	1
	3	Russot	45600.00	2
	4	Delevoy	12000.00	1
▶	5	Dumans	34000.00	2
*	NULL	NULL	NULL	NULL

Enfin, on va :

- ✓ rendre la colonne clé étrangère NOT
- ✓ renommer la clé étrangère

```
/**
 * @ORM\ManyToOne(targetEntity="Lieu", inversedBy="lesEmployes")
 * @ORM\JoinColumn(name = "idLieu" , nullable=false)
 */
private $lieu;
```

Il ne vous reste plus qu'à mettre à jour la BD

- ✓ doctrine:migration:diff
- ✓ doctrine:migration:migrate

Vous allez maintenant créer une nouvelle action dans le contrôleur PrincipalController qui va vous permettre d'afficher un employé avec son lieu de travail :

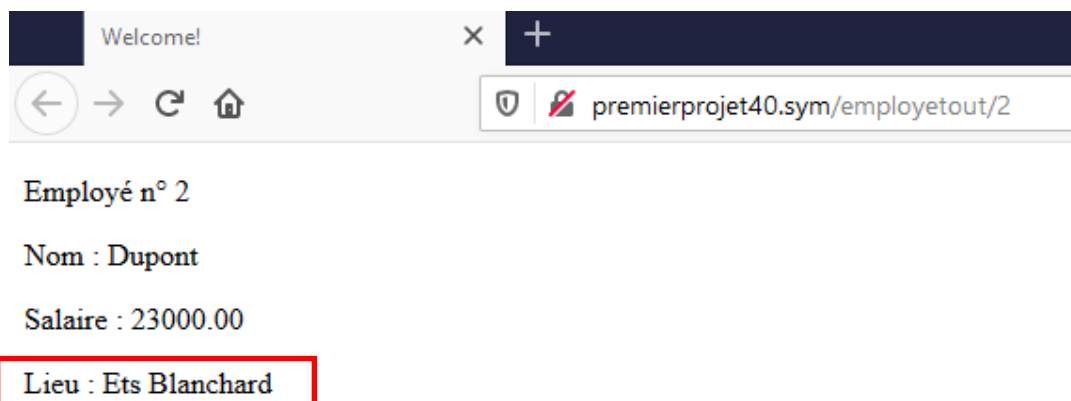
```
/**
 * @Route("/employetout/{id}", name="employetout" , requirements={"id":"\d+"})
 * @param ManagerRegistry $doctrine
 */
public function afficheUnEmployeTout(ManagerRegistry $doctrine, int $id) {
    $employe = $doctrine->getRepository(Employe::class)->find($id);
    $titre = "Employé n° " . $id;
    return $this->render('principal/unemployetout.html.twig', compact('titre', 'employe'));
}
```

Puis vous créez la vue (le template) qui correspond : unemployetout.html.twig

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

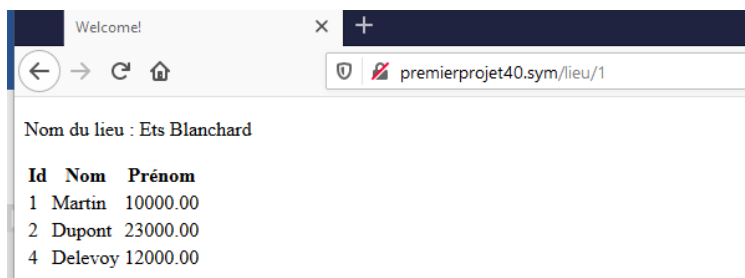
{% block title %}{{ parent() }} {% endblock %}
{% block body %}
    <p> {{ titre }}
    <p> Nom : {{ employe.nom }} </p>
    <p> Salaire : {{ employe.salaire }} </p>
    <p> Lieu : {{ employe.lieu.nom }} </p>
{% endblock %}
```

Exécutez : <http://premierprojet40.sym/employetout/2>



Partie 10 : TRAVAIL A FAIRE

Modifiez l'application pour arriver à afficher ceci ... en plus ergonomique si vous le souhaitez



Bravo pour votre travail, il ne reste plus maintenant qu'à tout revoir dans les détails

