

Prise en main de Symfony 4.2 Présentation des possibilités

Un grand merci à Thomas Laure (BTS SIO 2018) pour avoir suggéré de compléter mes travaux de découverte de Symfony et d'avoir réalisé la V1.0 de ce TD. Je m'en suis très largement inspiré !

Environnement :

- ✓ Serveur apache : celui de wamp (préféré à celui intégré à symfony)
- ✓ Base de données : mysql en local v>5.7
- ✓ PHP > V.7.0
- ✓ Apache > 2.4.xx
- ✓ IDE : netbeans >= 11
- ✓ virtualHost : premierProjet40.Sym
- ✓ **x-debug activé**

Partie 1 : A LIRE AVANT DE COMMENCER



Ce TP n'est pas à faire d'un coup ! Il est long et pourra vous paraître fastidieux. Il est préférable d'avancer étape par étape et comprendre ce que l'on fait plutôt que de faire un « rush », de terminer avant les autres et de penser que le professeur en sera satisfait !

Nous sommes là pour passer de l'état « consciemment incompetent » à l'état « consciemment compétent » (M. Gil).

Pour Symfony comme pour d'autres langages ou frameworks, ayez le réflexe d'aller voir la documentation dès que vous souhaitez apprendre quoi que ce soit ou dès que vous bloquez.

Celle de Symfony est très complète, et vous donne de petits tutoriels **à jour** vous permettant de visualiser le fonctionnement des formulaires, de l'ORM avec Doctrine, etc...

La documentation **officielle** sera toujours plus fiable que ce que vous pourrez trouver sur YouTube, parce qu'elle est à jour et écrite par des contributeurs du projet Symfony.

Autres sources de documentation :

Openclassroom (version 3 proposée au 21/03/2019) : cours écrit par Fabien POTENCIER, le créateur de Symfony.

symfonycasts.com : un cours vidéo en anglais, mais largement compréhensible : <https://symfonycasts.com/screencast/symfony>



Il est important de maîtriser le vocabulaire si vous voulez coder correctement. J'ajouterais même que c'est en forgeant qu'on devient forgeron !
Au terme de ce TP toutes les notions seront devenues naturelles pour vous !

N'oubliez pas non plus de bien respecter les règles d'écriture en respectant notamment les normes PSR sans quoi vous passerez votre temps à déboguer et... vous vous découragerez !!!

Partie 2 : INSTALLATION DE COMPOSER

Dans la version 4 de Symfony, l'installation se fait via Composer et pas avec la commande Symfony comme dans la version 3.



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 :

```
C:\> 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

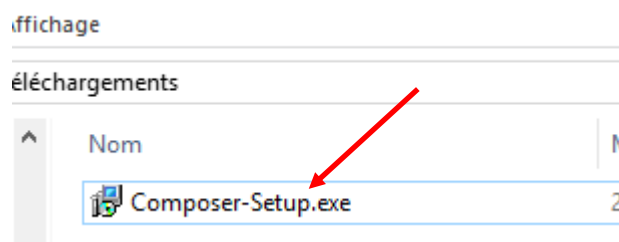
The installer will download composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

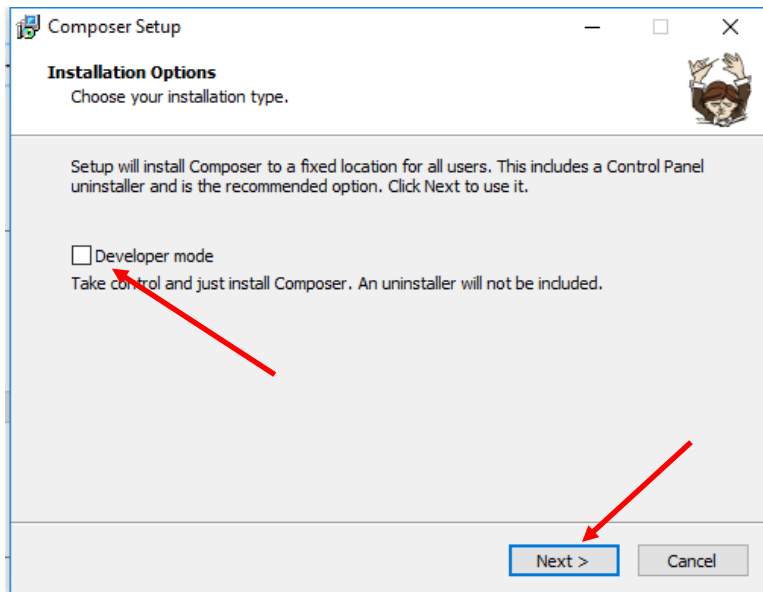
Download and run [Composer-Setup.exe](#) - it will install the latest composer version whenever it is executed.

Command-line installation

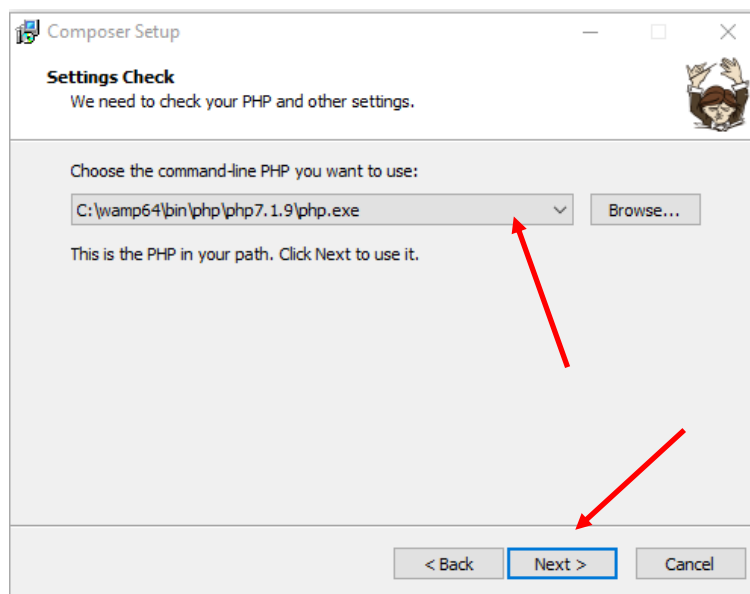
To quickly install Composer in the current directory, run the following script in your terminal. To automate the installation, use [the guide on installing Composer programmatically](#).

Téléchargez le fichier composer-setup.exe et exécutez le

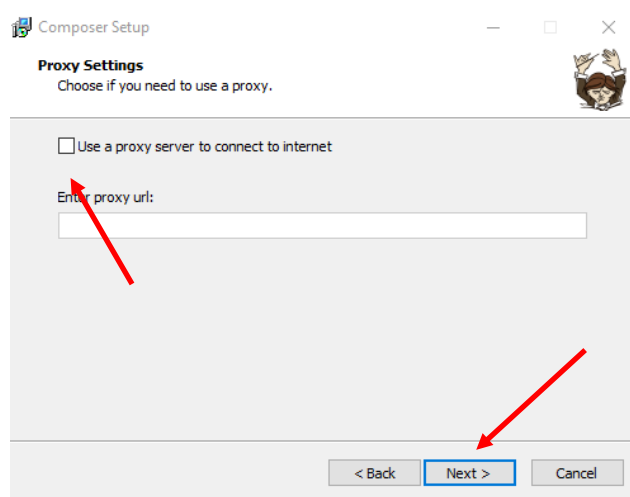




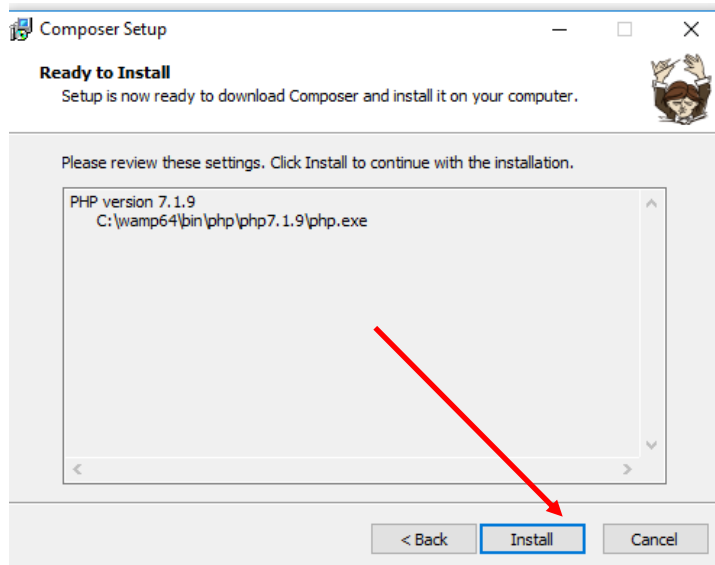
Indiquez le dossier contenant votre fichier php.exe



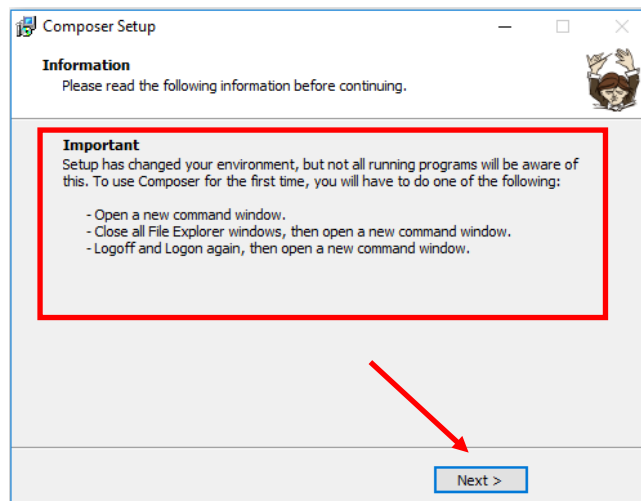
Indiquez si vous avez un proxy pour la sortie sur internet :



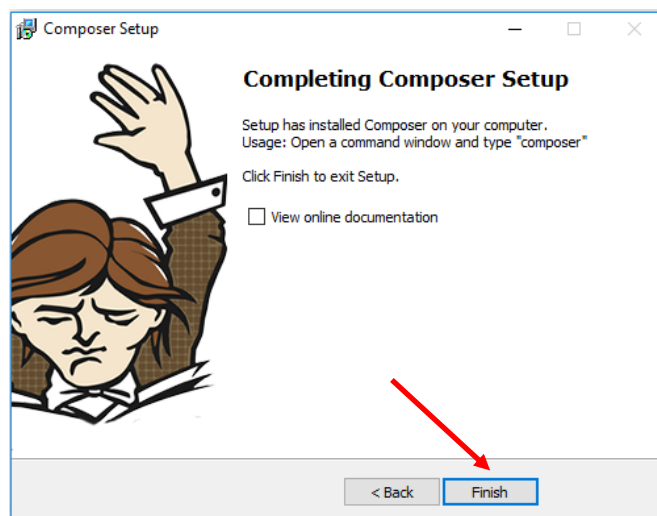
C'est parti :



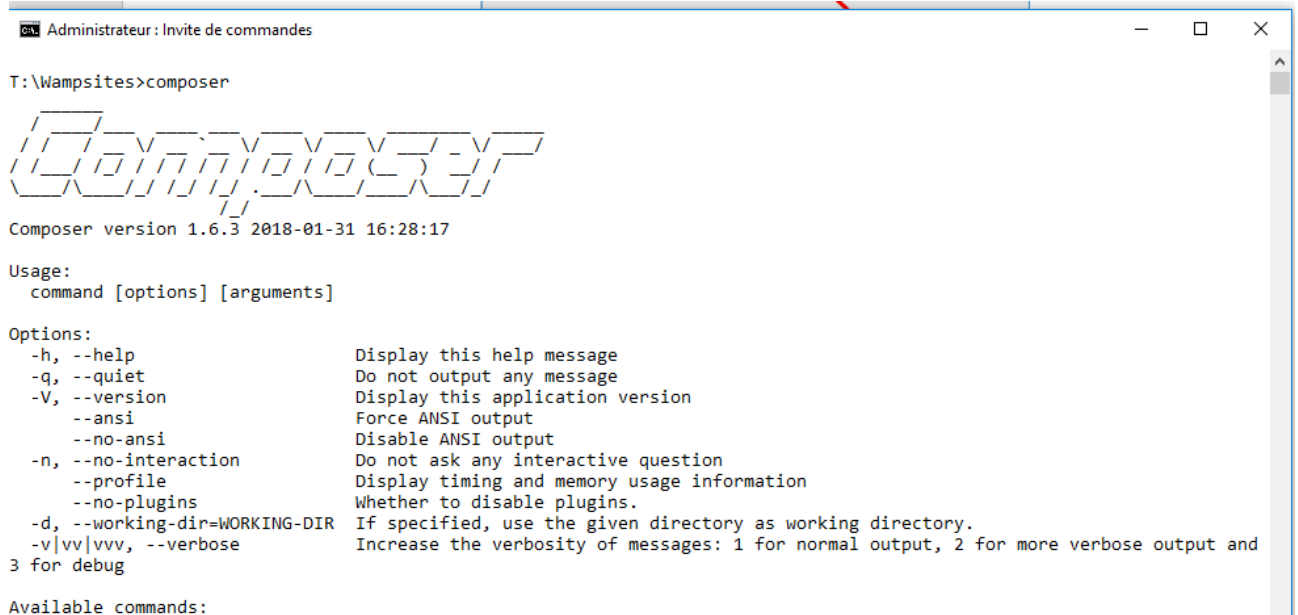
Lisez bien les recommandations :



C'est fini :



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



```

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 3 : CREATION DU PROJET PREMIERPROJET42

On va utiliser composer pour installer le framework symfony

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

Creating Symfony applications

STABLE VERSION

LTS VERSION

Symfony 4.2 is the latest stable version. Use it to get the most recent Symfony features. End of support for bug fixes: Jul 2019 (see [roadmap](#)).

If you are building a **traditional web application**:

```
symfony new --full my_project
```

COPY

If you prefer Composer: `composer create-project symfony/website-skeleton my_project`

If you are building a **microservice, console application** or **API**:

```
symfony new my_project
```

COPY

If you prefer Composer: `composer create-project symfony/skeleton my_project`

L'installation se fera par composer



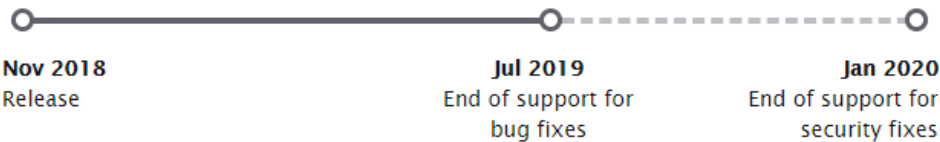
Vous allez installer la dernière version de Symfony, à savoir la version 4. On peut aller voir la roadmap de symfony pour voir que la prochaine version (4.1) sera disponible en mai 2018 :

Symfony 4.2 Roadmap

Symfony manages its releases through a time-based model. A new Symfony minor version comes out every six months, one in May and one in November. Check out the [release process details](#).

Symfony 4.2 is the **latest stable version** and it will be maintained until the end of January 2020.

Roadmap



TIP Get this information in JSON format: <https://symfony.com/roadmap/4.2.json>



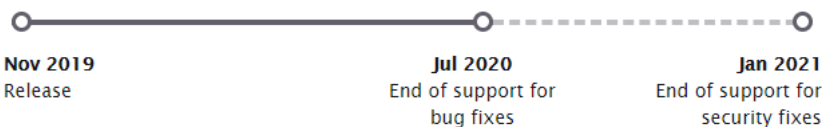
Un petit check nous montre que la version 5.0 sortira en ... novembre 2019 ...

Symfony 5.0 Roadmap

Symfony manages its releases through a time-based model. A new Symfony minor version comes out every six months, one in May and one in November. Check out the [release process details](#).

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

Roadmap



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



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

Symfony sera installé dans le dossier *premierprojet42* situé dans le dossier où vous exécuterez votre commande composer :

`composer create-project symfony/skeleton premierprojet42`

```

C:\ Invite de commandes
T:\Wampsites\CoursSymfony>composer create-project symfony/skeleton premierprojet42

```

Et voilà en quelques secondes, c'est fait :



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

```

C:\ Invite de commandes

Generating autoload files
Symfony operations: 4 recipes (1df2fbfc8e03c859ee307ea52240ee3a)
- Configuring symfony/flex (>=1.0): From github.com/symfony/recipes:master
- Configuring symfony/framework-bundle (>=4.2): From github.com/symfony/recipes:master
- Configuring symfony/console (>=3.3): From github.com/symfony/recipes:master
- Configuring symfony/routing (>=4.2): From github.com/symfony/recipes:master
Executing script cache:clear [OK]
Executing script assets:install 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. Create your code repository with the git init command
  3. Run composer require server --dev to install the development web server,
     or configure another supported web server https://symfony.com/doc/current/setup/web_server_configuration.html

* 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 :

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

premierprojet42.sym

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

T:\Wampsites\CoursSymfony\premierprojet42\public

Si vous voulez utiliser un "Listen port" autre que celui par défaut, vous devez ajouter un Listen Port à Apache par Clic-Droit Outils **Optionnel**

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

Démarrer la création du VirtualHost (Peut prendre un certain temps)

Don't FORGET!

Ajouter un VirtualHost - Retour à l'accueil

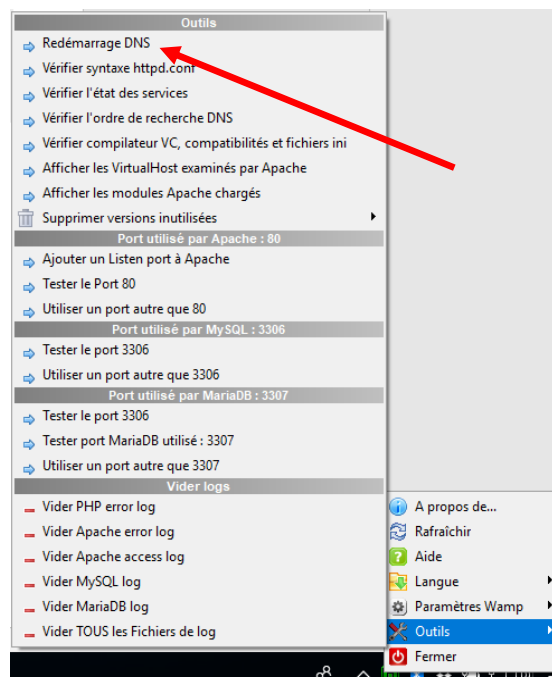
WampServer

Les fichiers ont été modifiés, le virtual host premierprojet.sym a été créé

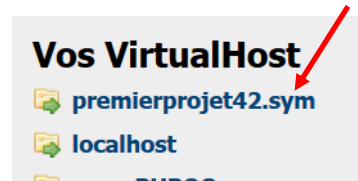
Messages de la console pour actualisation des DNS :

Vous pouvez ajouter un autre VirtualHost en validant "Ajouter un VirtualHost"
Cependant, pour que vos nouveaux VirtualHost soient pris en compte par Apache, vous devez lancer l'item Redémarrage DNS du menu Outils par Clic-Droit sur l'icône Wampmanager. (Ceci ne peut, hélas, pas être fait automatiquement)

Redémarrer le DNS.



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

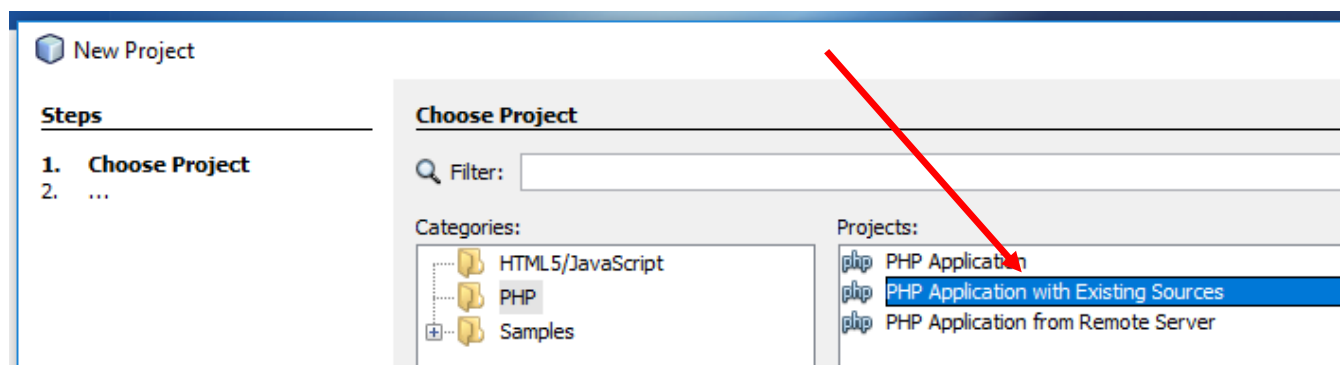
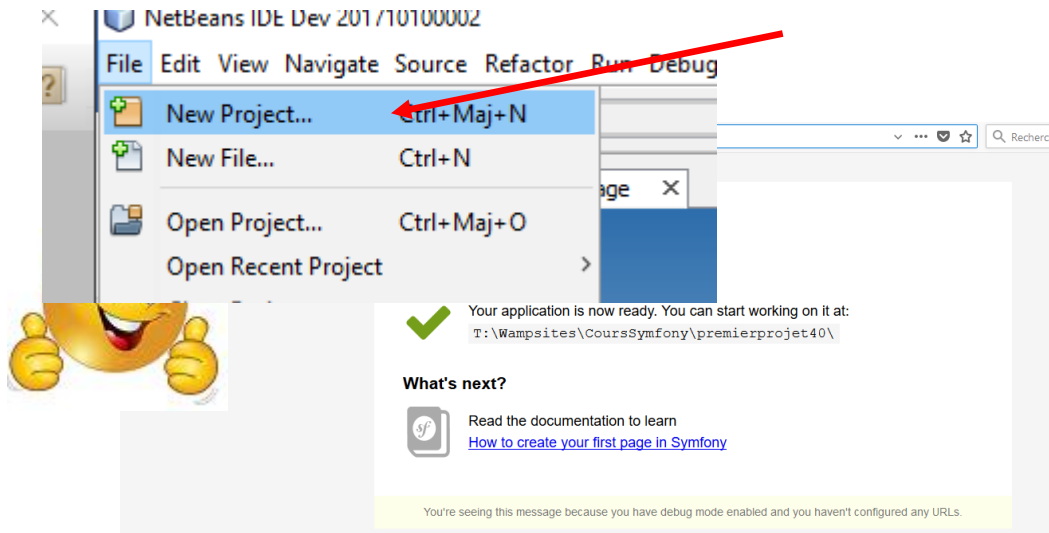


Testez :

Partie 4 : PROJET NETBEANS

L'idée est de créer un nouveau projet ... **Avec les sources existantes**

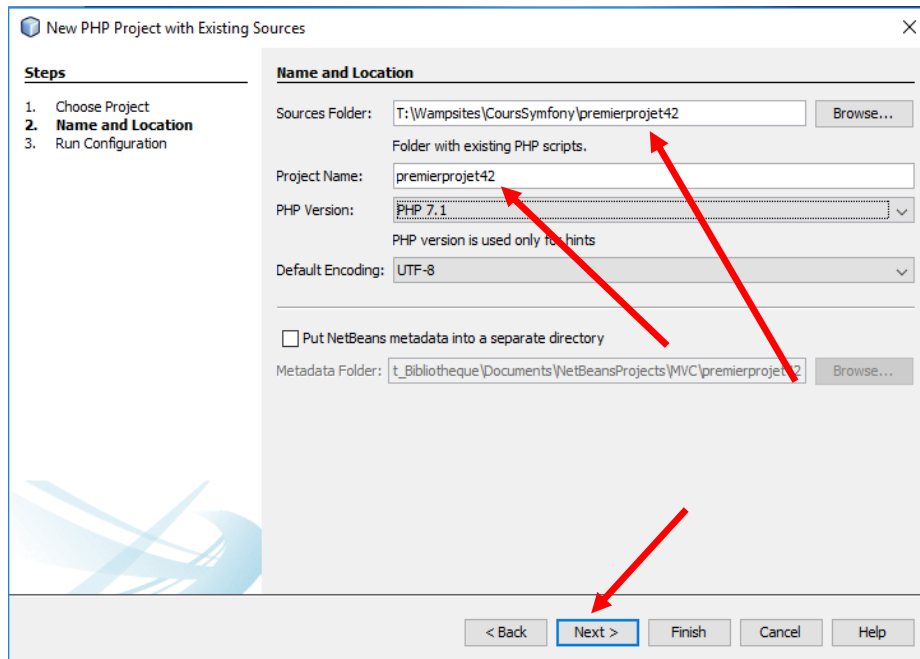
Ouvrez Netbeans et créez un nouveau projet



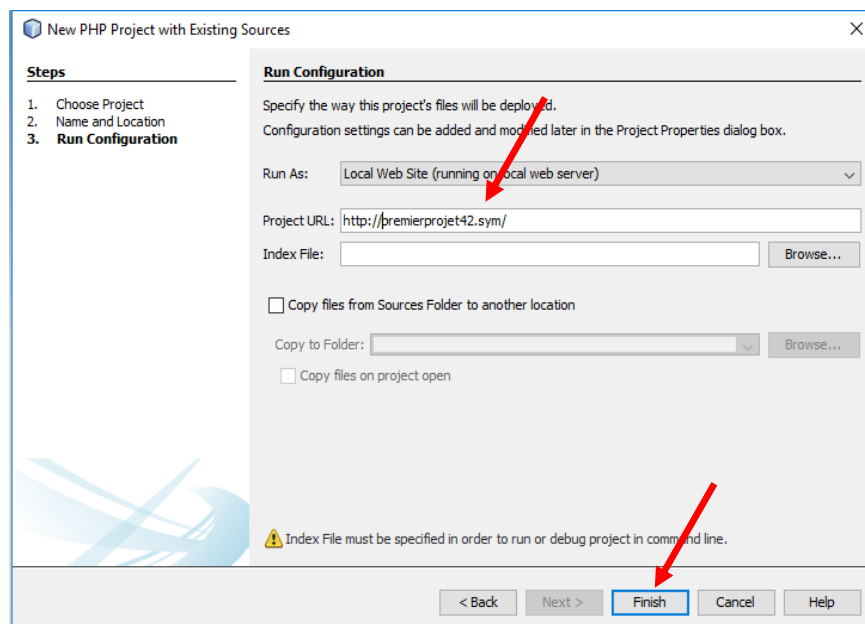
Choisir un projet php **AVEC APPLICATION EXISTING SOURCES**

Indiquez :

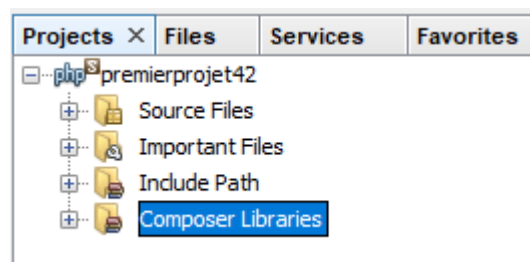
- ✓ le chemin du projet.
- ✓ le nom du projet
- ✓ la version de PHP



✓ et l'URL :



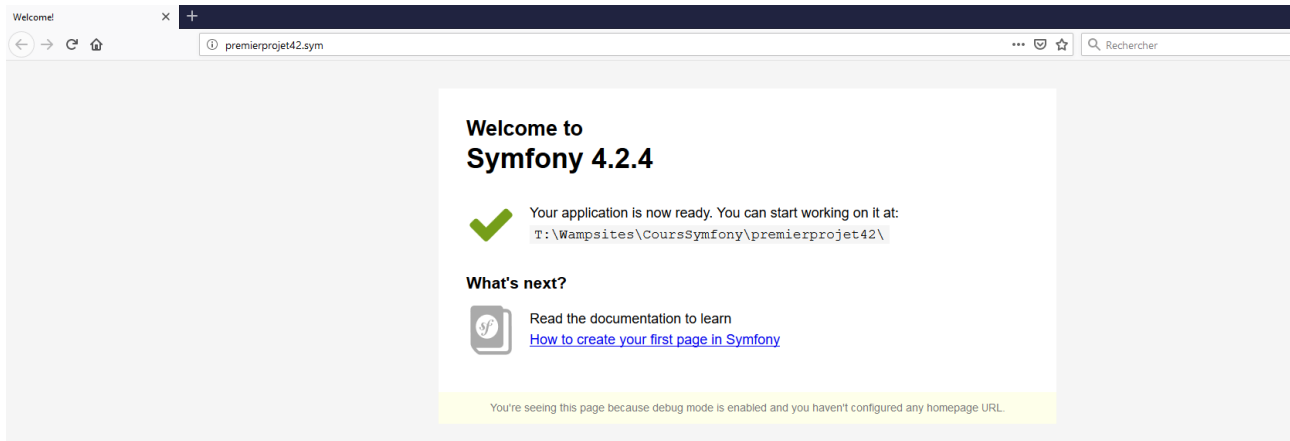
Notre projet a bien été créé :



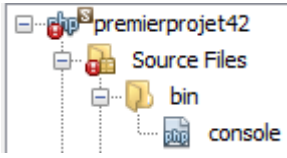
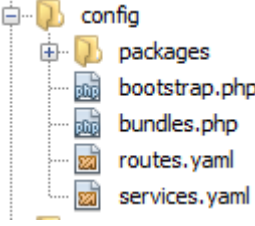
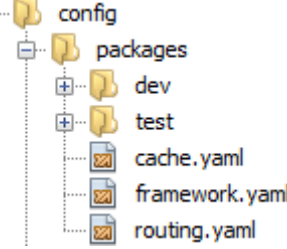
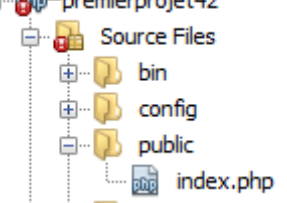

Testez :

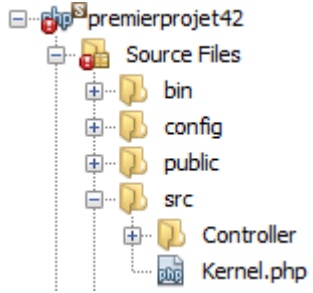


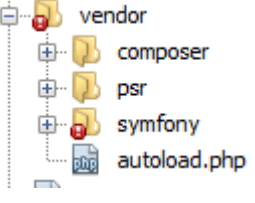


Ça fonctionne :



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> <p>Pour l'instant il contient ... le framework symfony et le fichier autoload.php</p>



Le contrôleur frontal : Si vous ouvrez le fichier public/index.php, vous verrez qu'il ne contient pas le code de la page que nous avons vu précédemment. Il s'agit du contrôleur frontal (vous entendrez souvent parler de front controller) ! C'est le point d'entrée de l'application. Toutes les demandes de page passent par ce fichier. Sa mission sera de charger le kernel (noyau) qui se chargera de trouver la bonne route pour renvoyer la bonne page.

Ne pas faire attention aux erreurs levées par NetBeans, elles n'empêchent en aucun cas le bon fonctionnement du projet)

Partie 5 : CREATION DE LA PREMIERE PAGE

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

Vous allez afficher votre nom et prénom 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

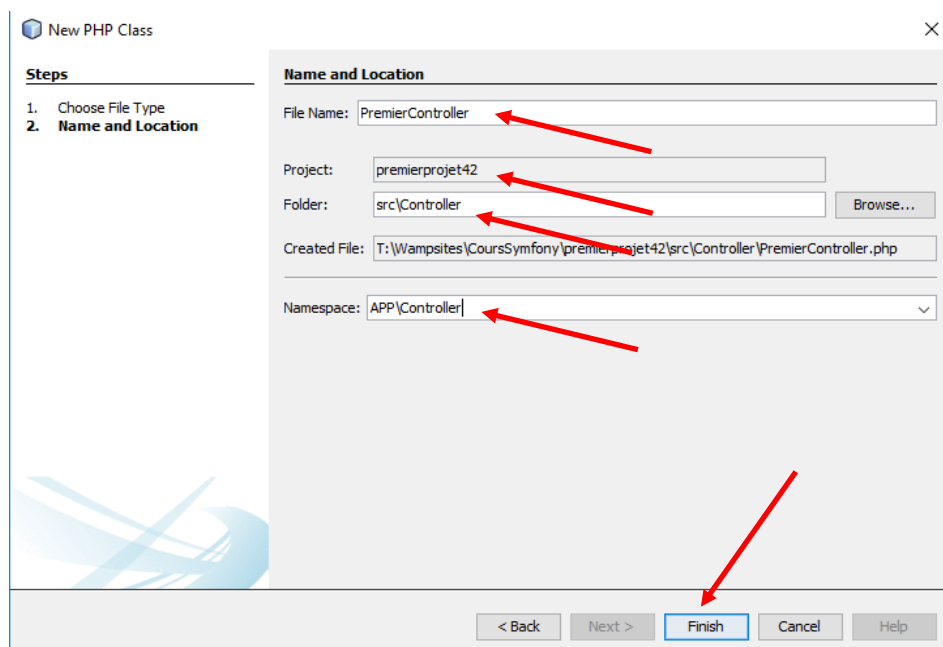
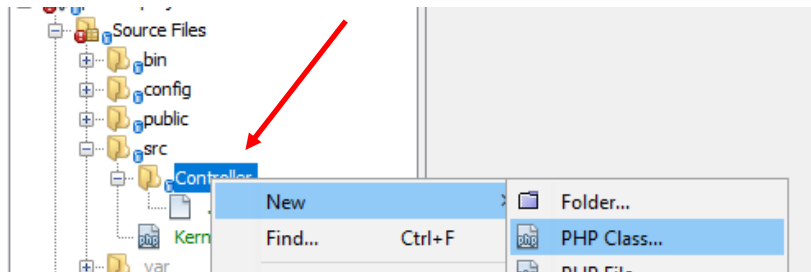
Un contrôleur est une classe dans le dossier src/Controller



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;

class PremierController {

    public function index() {
        return new Response($content='Bonjour mon premier contrôleur');
    }

}
```

3. Création d'une route

Ensuite, il faut dire à symfony que l'url : <http://premierprojet42.sym/index> va exécuter la méthode PremierController::index

Allez dans le fichier config/routes.yaml
Et entrez ceci :

```
index:
    path: /index/
    controller: App\Controller\PremierController::index
```



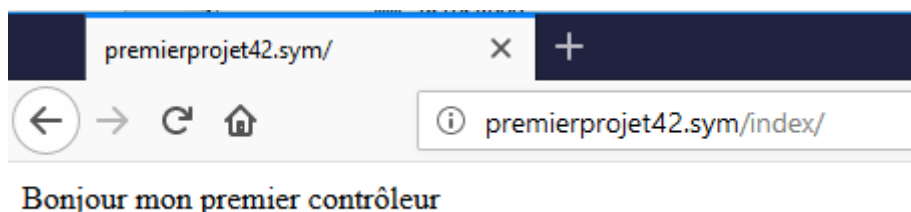
N'utilisez pas les tabulations mais les espaces et par groupes de 4 espaces

C'est donc ici que l'on va associer une URL à une action (méthode)



On verra plus loin que l'on va rapidement abandonner cette méthode qui est un peu lourde au profit des annotations qui sont largement répandues en symfony.

Sauvegardez vos fichiers et entrez l'url : <http://premierprojet42.sym/index>
Vous obtenez ceci ... en principe

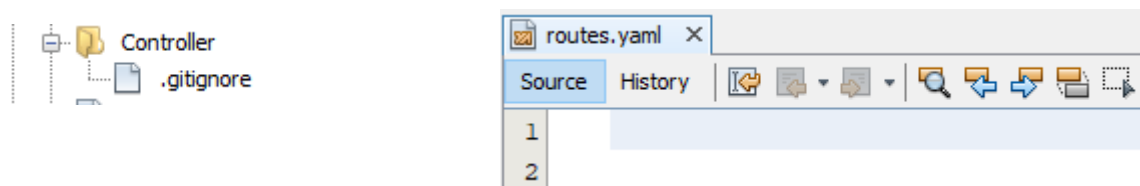


On vient de voir seulement dans cette partie comment associer une URL (route) à une action (méthode) d'une classe Controller.



On peut faire beaucoup mieux !!!

Vous pouvez maintenant supprimer votre contrôleur et sa route dans le fichier routes.yaml :



Vous n'aurez plus besoin de ce contrôleur pour la suite !!!

Mais avant de passer à la suite, nous allons nous intéresser aux dépendances ! C'est incontournable tant elles vont nous offrir d'énormes facilités de développement !!!

Partie 6 : LES DÉPENDANCES

On peut également observer dans le fichier `composer.json` que Composer a installé les dépendances supplémentaires nécessaires au projet comme `symfony/flex`.

```
"require": {
    "php": "^7.1.3",
    "ext-ctype": "*",
    "ext-iconv": "*",
    "symfony/console": "4.2.*",
    "symfony/dotenv": "4.2.*",
    "symfony/flex": "^1.1",
    "symfony/framework-bundle": "4.2.*",
    "symfony/yaml": "4.2.*"
},
```

Qu'est-ce que Flex ?

La réponse dans la documentation ! <https://symfony.com/doc/current/setup/flex.html>

C'est un plugin Composer dont les objectifs sont de faciliter la gestion des dépendances et d'avoir une configuration par défaut qui fonctionne immédiatement. Il requiert au minimum la version 7.1 de PHP (nous sommes dans la version 7.2.4).

Oui mais concrètement ?

- ✓ Installation d'un package sans Flex on doit faire, dans l'ordre :
- ✓ `composer require mon-package`
- ✓ Instancier le(s) package(s) dans le Kernel (un gros-mot)
- ✓ Créer la configuration dans `app/config/config.yml` (un autre)
- ✓ Importer le routing dans `app/config/routing.yml` (ô joie)

Et avec Flex :

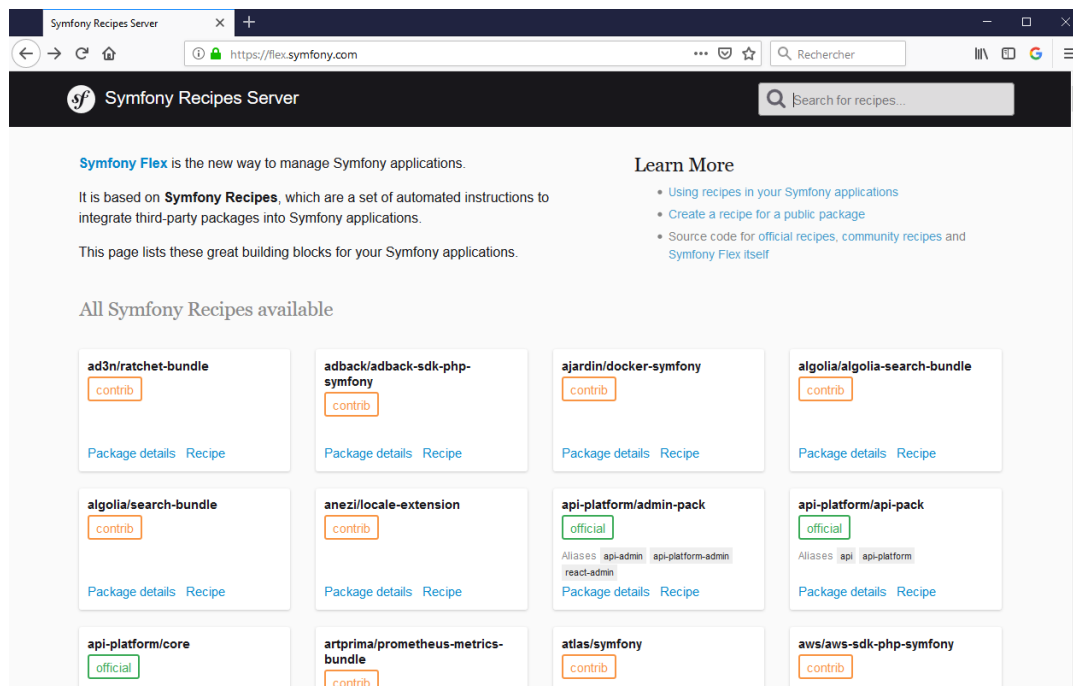
- ✓ `composer require mon-package` (plus facile)

Flex automatise donc l'installation des packages et de leurs dépendances et permet d'utiliser des alias pour les installer.

Les alias permettent d'installer des packages en utilisant un nom plus court que leur nom complet. Par exemple : l'alias de package `api-platform/api-pack` est : `api`.

L'utilisation des alias permet de gagner du temps d'écriture et en ne se trompant pas dans le nom du package. C'est tout de même plus confortable.

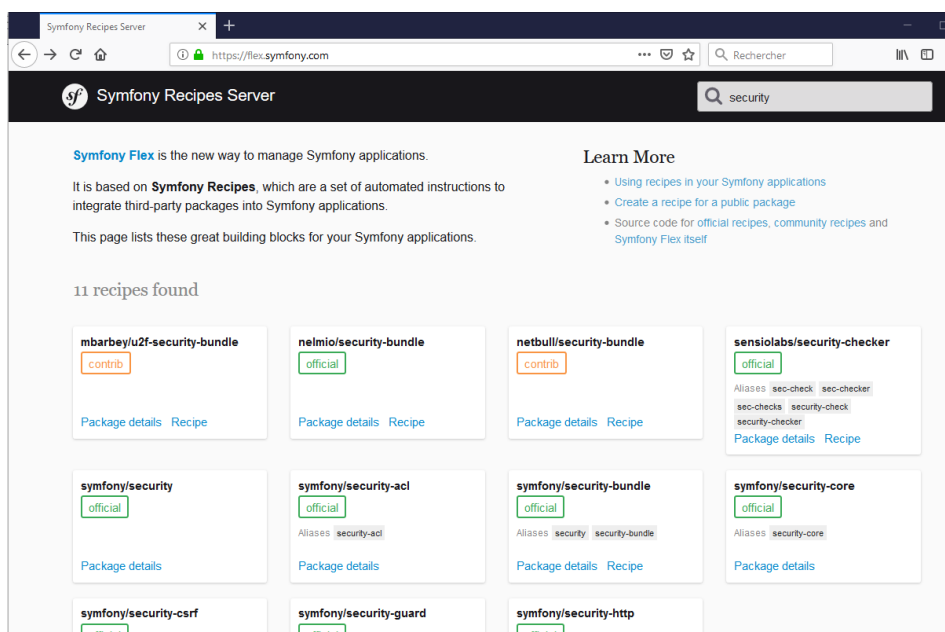
Vous trouverez ces dépendances ou recettes (Recipies) sur le site <https://flex.symfony.com/> (anciennement <https://symfony.sh/>)



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

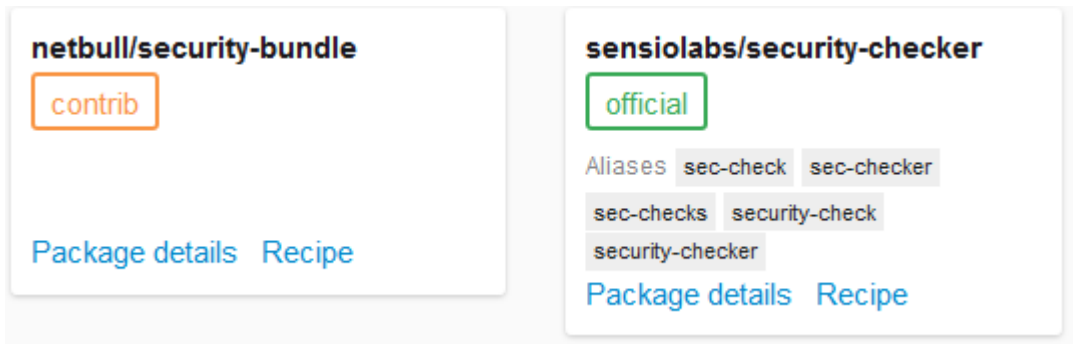
Par exemple, vous souhaitez installer dans votre projet un recipe (packages) permettant de vérifier la conformité des autres recipes que nous installerons par la suite dans notre projet.

Vous allez donc taper dans la barre de recherche du site *security*.



Vous vous retrouverez avec un certain nombre de packages... Quand nous ne sommes pas habitués, ça peut porter confusion.

La première chose à regarder, ce sont les petites étiquettes suivantes :



Le premier recipe est une contribution non officielle

Le deuxième recipe est un recipe officiel

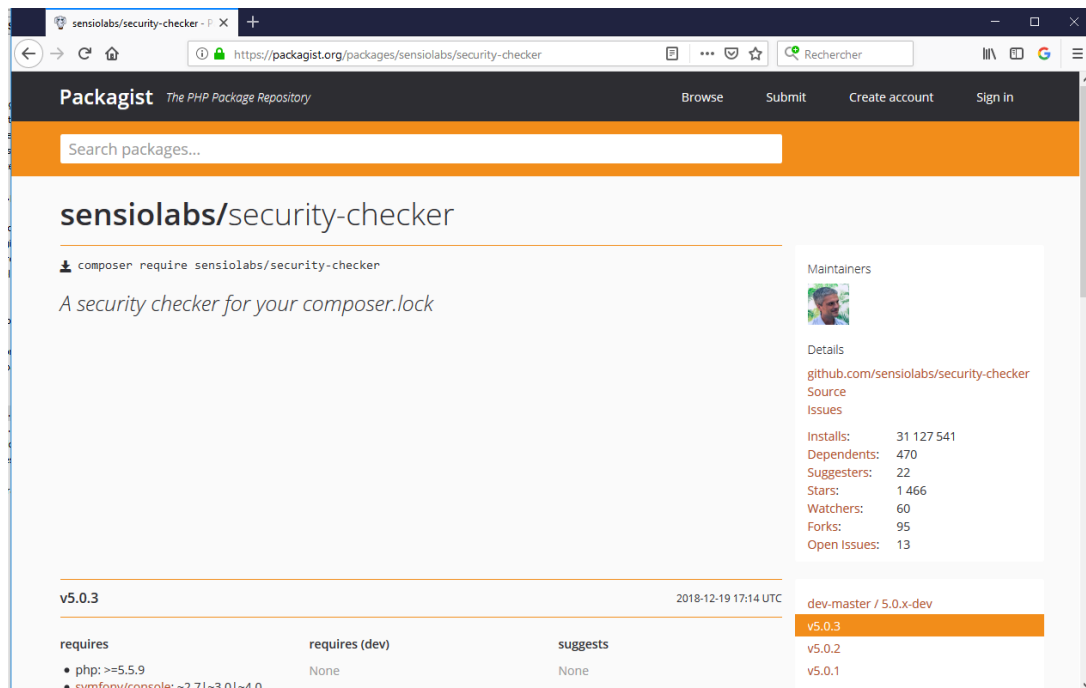


Vous privilégiez toujours privilégier les recipes officiels par soucis de fiabilité.

Les recipes sont nommées de la manière suivante : *nom-auteur/nom-recipe*.

Dans le cas des deux recipes précédents, nous avons donc deux auteurs distincts : Netbull et SensioLabs. Sachant que ce dernier correspond au projet Symfony, il serait préférable de choisir celui-ci.

Vous pouvez ensuite cliquer sur *Package details* du second recipe, nous sommes redirigés sur Packagist qui est le site où sont stockés tous les packages pour PHP que nous pouvons installer via Composer.



Cette page renseigne donc sur le nom du package, son auteur, ses dépendances, sa version, etc... et sa **description**.

A security checker for your composer.lock

Et composer dans tout ça ?

D'après la documentation de Composer (<https://getcomposer.org/doc/01-basic-usage.md#composer-lock-the-lock-file>), le fichier composer.lock permet de garder les dépendances dans un « état connu ». C'est-à-dire que lorsque vous versionnez votre projet, les packages et leurs dépendances **ne sont pas versionnés** (imaginez le temps que vous mettriez à récupérer votre projet si vous utilisez un VCS distant comme GitLab ou GitHub !).

Mais le composer.json et le composer.lock le sont eux.

Et ils permettent, lorsque vous venez de télécharger votre projet depuis un dépôt distant, de télécharger toutes les dépendances que vous aviez installées en conservant la version qu'elles avaient sur votre projet (pour éviter les problèmes liés à la compatibilité) en lançant simplement la commande `composer update`.

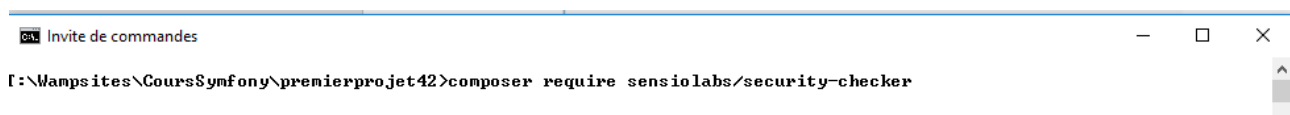
Le package que nous avons trouvé est donc celui que nous cherchions !

Supposons que ce package que est celui que vous cherchiez, vous allez l'installer.. via composer.

Vous pouvez le retrouver et le charger depuis son dépôt en cliquant sur [Recipe](#) mais cela n'a pas d'intérêt.

Essayons donc de l'installer via composer et sans l'alias avec la commande :

Composer require sensiolabs/security-checker



Voici le résultat, tout s'est bien passé :

```

C:\> Invite de commandes

T:\Wampsites\CoursSymfony\premierprojet42>composer require sensiolabs/security-checker
Using version ^5.0 for sensiolabs/security-checker
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.2.*"
Nothing to install or update
Generating autoload files
Executing script cache:clear [OK]
Executing script assets:install public [OK]
Executing script security-checker security:check [OK]

T:\Wampsites\CoursSymfony\premierprojet42>

```

Examinez la partie require du fichier composer.json, vous trouverez le recipe :

```

"require": {
    "php": "^7.1.3",
    "ext-ctype": "*",
    "ext-iconv": "*",
    "sensiolabs/security-checker": "^5.0",

```

Maintenant vous allez le désinstaller à l'aide de la commande
composer remove sensiolabs/security-checker :

```

C:\> Invite de commandes

T:\Wampsites\CoursSymfony\premierprojet42>composer remove sensiolabs/security-checker
Dependency "symfony/console" is also a root requirement, but is not explicitly whitelisted. Ignoring.
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.2.*"
Package operations: 0 installs, 0 updates, 2 removals
 - Removing sensiolabs/security-checker (v5.0.3)
 - Removing composer/ca-bundle (1.1.4)
Writing lock file
Generating autoload files
Symfony operations: 1 recipe (def018689dc81444d4afa45aaad4b59f)
 - Unconfiguring sensiolabs/security-checker (>=4.0): From github.com/symfony/recipes:master
Executing script cache:clear [OK]
Executing script assets:install public [OK]

T:\Wampsites\CoursSymfony\premierprojet42>

```

Vérifiez dans le fichier composer.json, le recipe ne doit plus y figurer .

Maintenant installons-le avec un de ses alias, par exemple *sec-check*:

```

C:\> Invite de commandes

T:\Wampsites\CoursSymfony\premierprojet42>composer require sec-check

```

Nous pouvons même observer qu'il s'est lancé à la fin de l'installation :

```

Executing script security-checker security:check [OK]

```

Vous l'avez compris, pour charger une dépendance fles, on utilise la commande composer en ligne de commande depuis le dossier racine de l'application :

Composer require monrecipient

Partie 7 : MISE EN PLACE DE L'APPLICATION EXEMPLE

Vous allez maintenant commencer à écrire du code pour réaliser une petite application où vous créerez :

- ✓ une page où vous pourrez créer des utilisateurs
- ✓ une page permettant de se connecter
- ✓ une page permettant d'afficher les informations de l'utilisateur connecté



N'oubliez pas Symfony est là pour vous simplifier la vie... mais avant de commencer il faut passer un peu de temps dans sa documentation, ainsi que celle de Twig, de Composer, de Doctrine, etc... ;-)

Mais après, tout sera vraiment plus simple pour vous....

1. Chargement des packages (recipes)

Vous aurez besoin des recipes suivants :

- ✓ **symfony/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.
- ✓ **sensio/framework-extra-bundle**: afin de pouvoir travailler sur l'ensemble de notre projet avec les annotations
- ✓ **symfony/profiler-pack**: 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)
- ✓ **symfony/twig-bundle**: afin de pouvoir utiliser le moteur de template traditionnellement associé à symfony
- ✓ **symfony/asset**: il s'agit d'un recipe qui va permettre de gérer les url's au sein de notre application et notamment quand on va travailler avec les fichiers css et javascript



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

Pour ne pas avoir à systématiquement saisir toutes les commandes, vous pouvez vous créer un fichier

loadrecipes.bat

à la racine de votre application. Il contiendra les commandes suivantes :

```
loadrecipes.bat x composer.json x
Source History | [Icons]
1 call composer require symfony/maker-bundle
2 call composer require sensio/framework-extra-bundle
3 call composer require symfony/profiler-pack
4 call composer require symfony/twig-bundle
5 call composer require symfony/asset
```

Il ne vous reste plus qu'à l'exécuter :

```

C:\> Invite de commandes

I:\Wampsites\CoursSymfony\premierprojet42>loadrecipes.bat
    
```

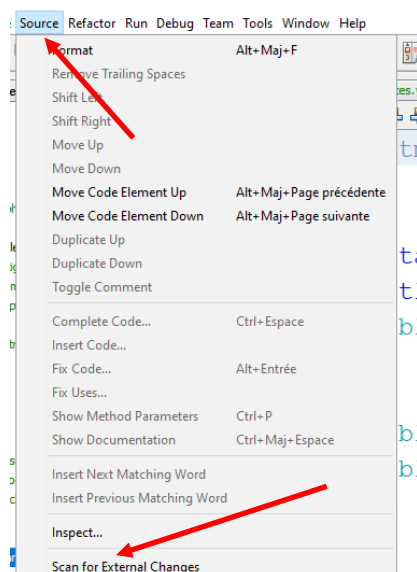
Vérifiez ensuite que vos packages sont bien installés (fichier composer.json)

```

"require": {
    "php": "^7.1.3",
    "ext-ctype": "*",
    "ext-iconv": "*",
    "sensio/framework-extra-bundle": "^5.2",
    "sensiolabs/security-checker": "^5.0",
    "symfony/asset": "4.2.*",
    "symfony/console": "4.2.*",
    "symfony/dotenv": "4.2.*",
    "symfony/flex": "^1.1",
    "symfony/framework-bundle": "4.2.*",
    "symfony/maker-bundle": "^1.11",
    "symfony/profiler-pack": "^1.0",
    "symfony/twig-bundle": "4.2.*",
    "symfony/yaml": "4.2.*"
},
    
```

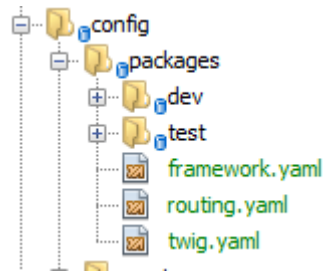


Pour afficher les dossiers et fichiers créés à l'extérieur de Netbeans, il est nécessaire de rafraîchir les sources du projet en faisant Source/Scan for External Changes :

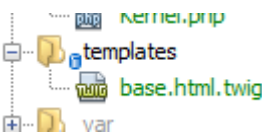


On voit

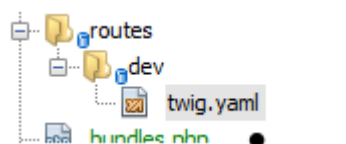
- ✓ qu'un fichier twig.yaml a été créé dans config/packages :



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




- ✓ qu'un fichier twig été rajouté dans le dossier routes/dev :




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

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


 Administrateur : Invite de commandes

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


Et

 Administrateur : Invite de commandes

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

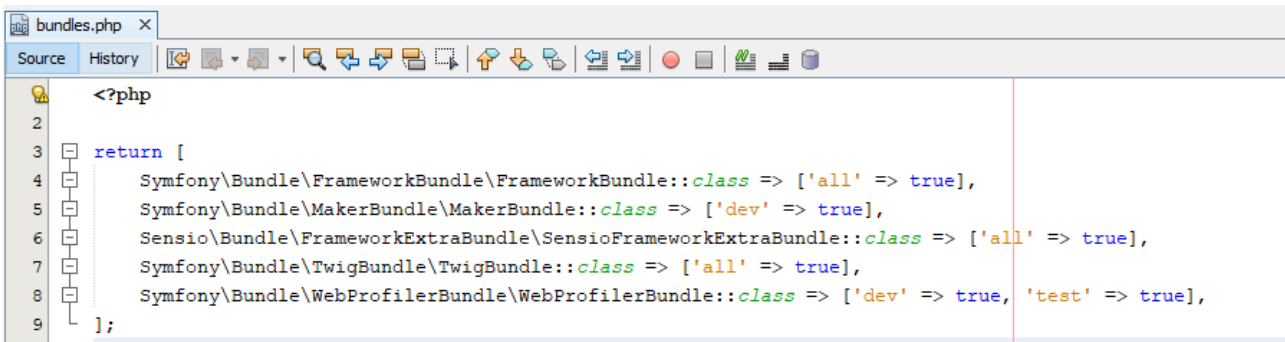
 Administrateur : Invite de commandes

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

 Administrateur : Invite de commandes

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

Allez voir le fichier config/bundles.php :



```
<?php
2
3 return [
4     Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
5     Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
6     Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],
7     Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
8     Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],
9 ];
```

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

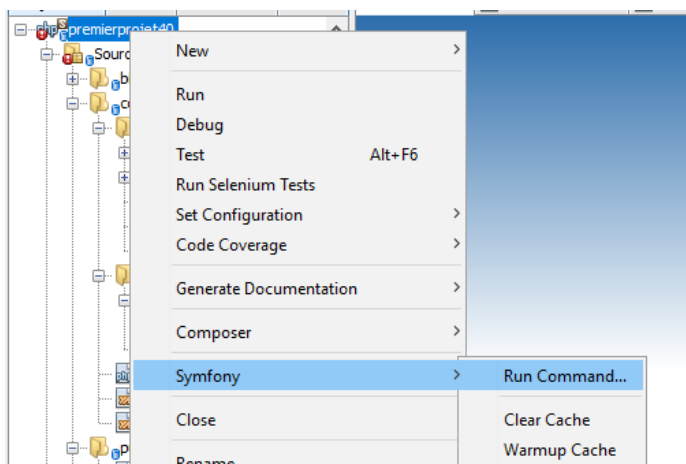
Dans les versions précédentes de symfony, on devait ajoutés ces lignes à la main !!!



2. Créer le contrôleur de notre application

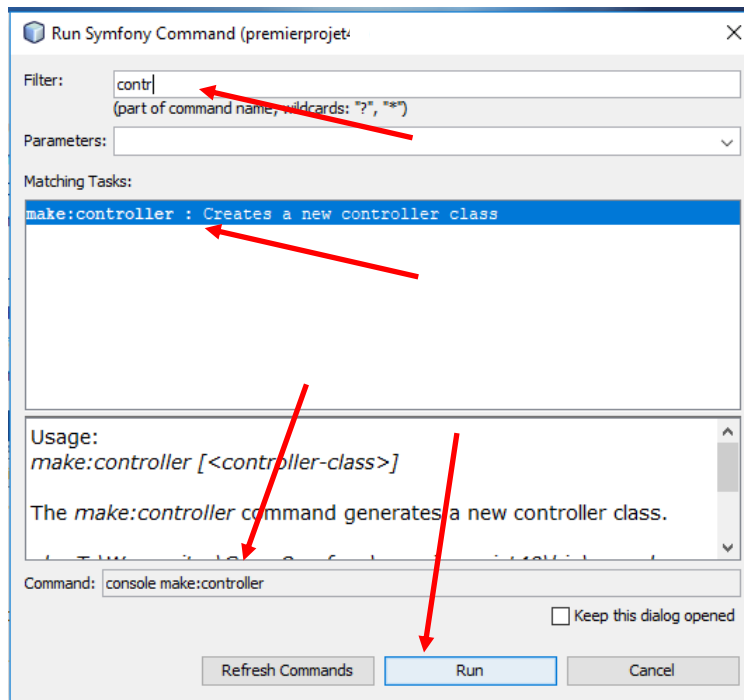
On va créer un deuxième contrôleur que l'on va appeler **PrincipalController**. Mais au lieu de le créer comme dans la partie précédente, on va le créer au moyen d'une commande Symfony que l'on peut désormais utiliser grâce au bundle Maker-Bundle que l'on vient d'installer.

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.

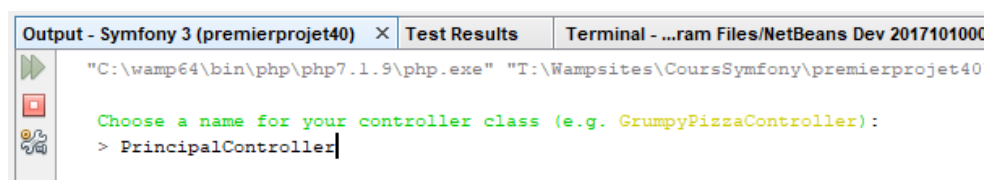
Je continue sur la fenêtre Netbeans

Vous aurez à saisir le nom du contrôleur

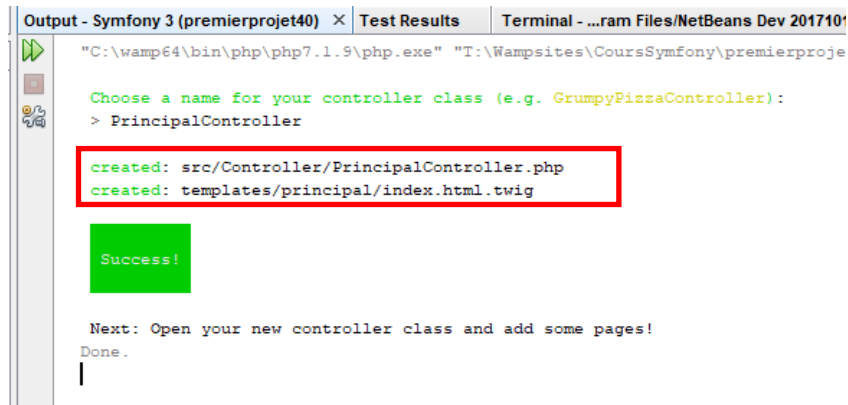
Attention ce sera une classe, donc il doit être en UpperCamelCase



PrincipalController



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



```

Output - Symfony 3 (premierprojet40) x Test Results Terminal - ...ram Files/NetBeans Dev 201710
"C:\wamp64\bin\php\php7.1.9\php.exe" "T:\Wampsites\CoursSymfony\premierproje

Choose a name for your controller class (e.g. GrumpyPizzaController):
> PrincipalController

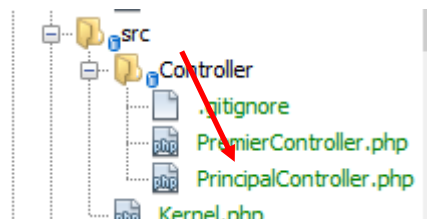
created: src/Controller/PrincipalController.php
created: templates/principal/index.html.twig

Success!

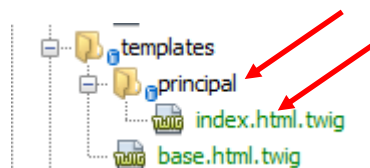
Next: Open your new controller class and add some pages!
Done.
|
    
```

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\AbstractController`
- ✓ 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`...

annotation ???

Une annotation est un bloc commenté. En fait il s'agit d'un commentaire enrichi qui peut contenir des informations sur les scripts ou des fonctions sans en modifier leur comportement.

Ici par exemple l'annotation `@Route` permettra au kernel de comparer la route saisie dans l'url (<http://premierprojet42.sym/principal>) et la route de l'annotation ("**/principal**").

S'il *matche* alors ce sera la méthode qui suit qui sera exécutée.

Testez cette URL : <http://premierprojet42.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`

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 :

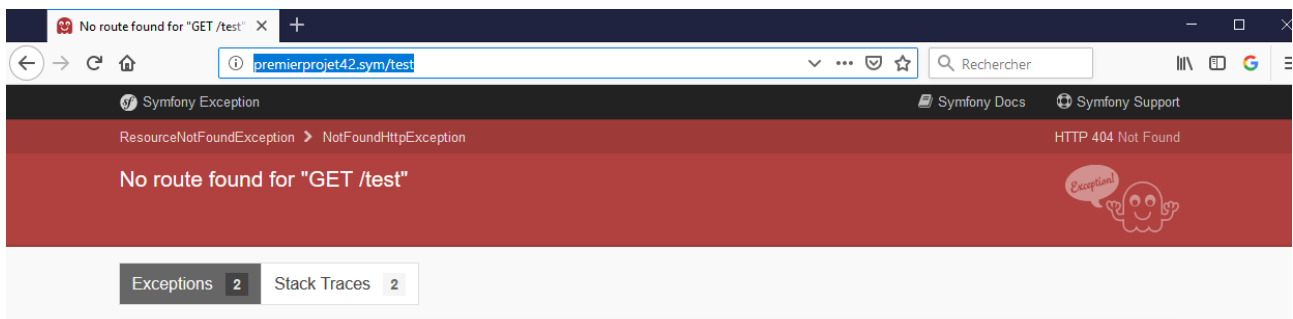
Hello Symfony c'est super!

This friendly message is coming from:

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

Essayez avec l'URL : <http://premierprojet42.sym/test>

Vous devriez obtenir ceci :



Pourquoi ?

Corrigez le code pour afficher 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`

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

Exemple d'URL :

<http://premierprojet42.sym/welcome/Benoît>

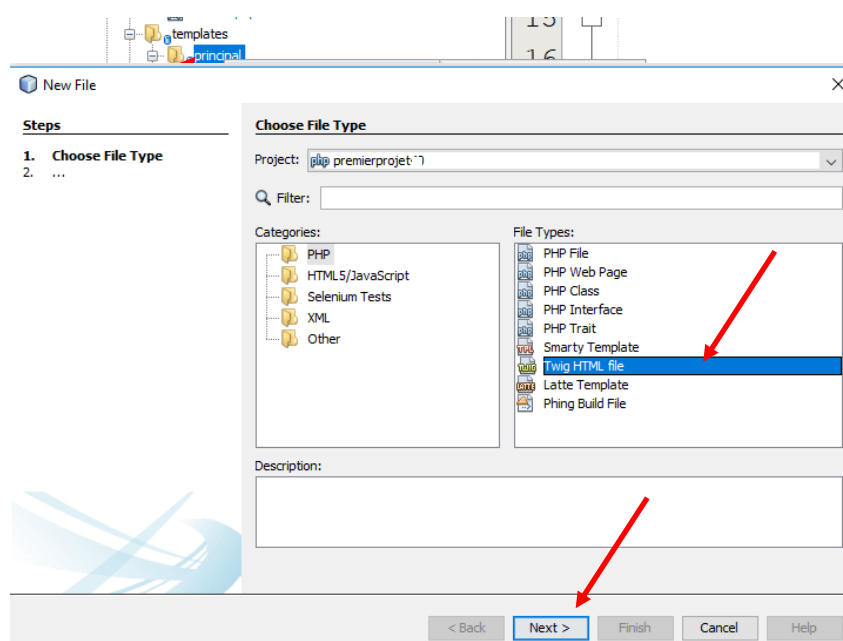
Dans le contrôleur PrincipalController, créez une nouvelle action appelée welcome et qui admet une chaîne de caractères en paramètre.

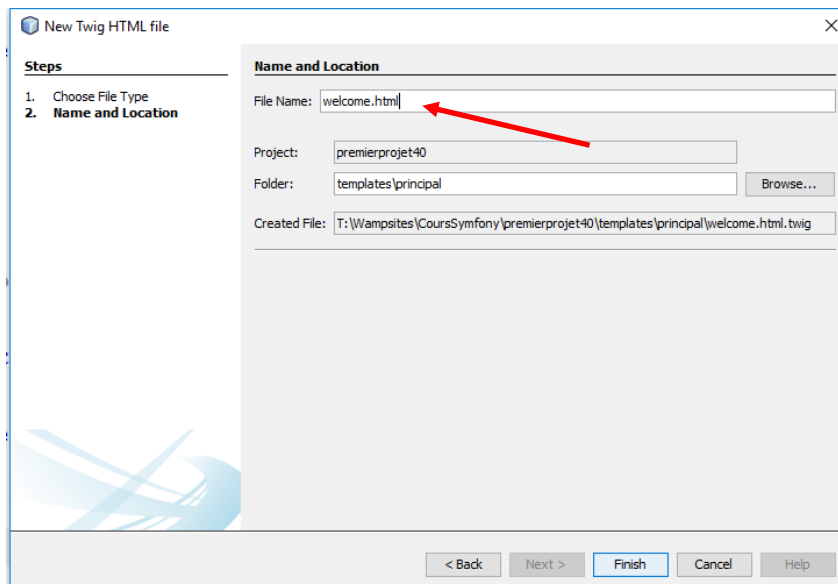
```
/**
 * @Route("/welcome/{nom}")
 */
public function welcome($nom) {
    return $this->render('principal/welcome.html.twig', array(
        "nom" => $nom
    ));
}
```

← Route avec un paramètre appelé nom

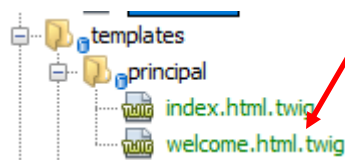
← Paramètre nom passé à la méthode

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





Voici :



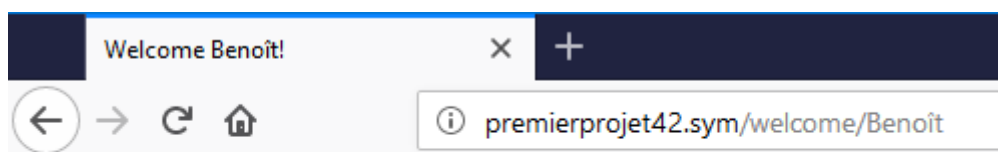
Ouvrez ce fichier et complétez avec ce code :

```
{% extends 'base.html.twig' %}
{% block title %}Welcome {{ nom }}!{% endblock %}

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

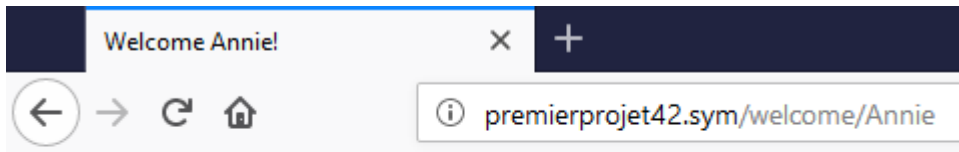
Formulaire hérité du
formulaire principal

Exécutez : <http://premierprojet42.sym/welcome/Benoît>



welcome Benoît!

Avec cette URL: <http://premierprojet42.sym/welcome/Annie>
on obtient :



welcome Annie!

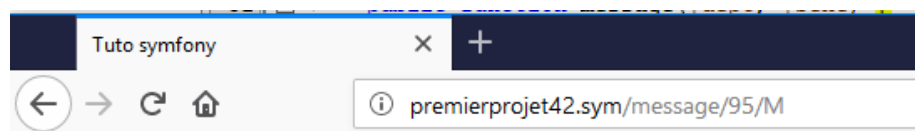
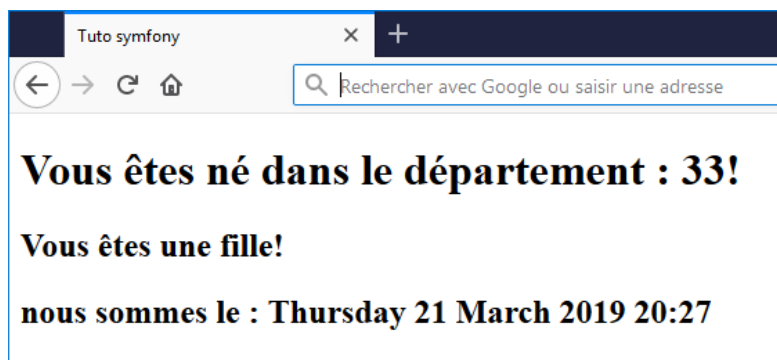
Vous remarquerez aussi en bas de la page, la barre d'outils qui s'affiche grâce au bundle profiler



Elle vous aidera bien par la suite

Partie 8 : EXERCICE

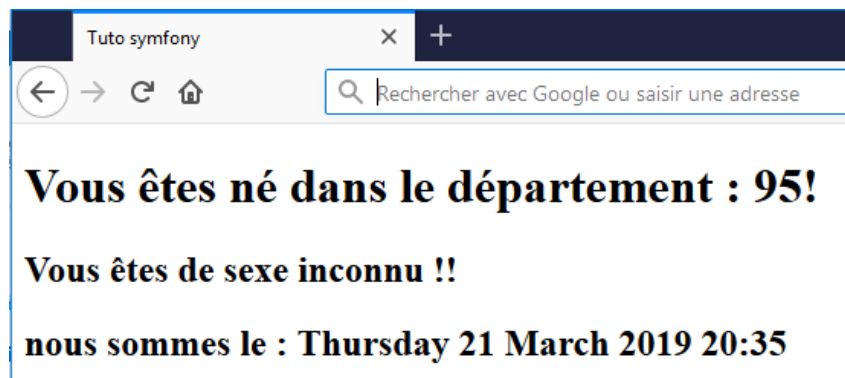
Créer une nouvelle action dans le contrôleur PrincipalController qui réponde à l'URL <http://premierprojet42.sym/??> et qui affiche ce que vous voulez ... vous passerez au moins 2 paramètres à la vue !



Vous êtes né dans le département : 95!

Vous êtes un garçon!

nous sommes le : Thursday 21 March 2019 20:28

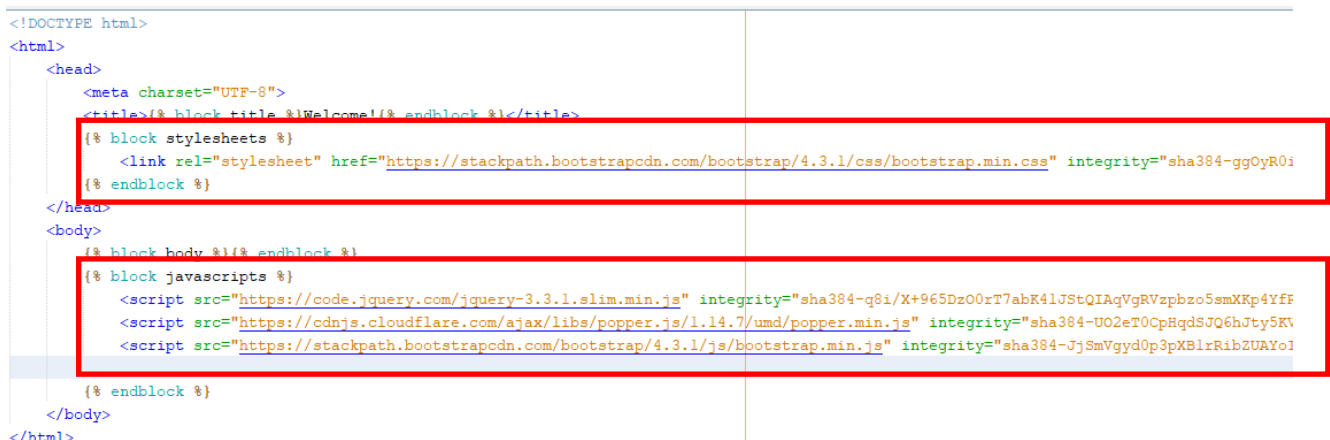


si on mettait un peu de bootstrap !!!

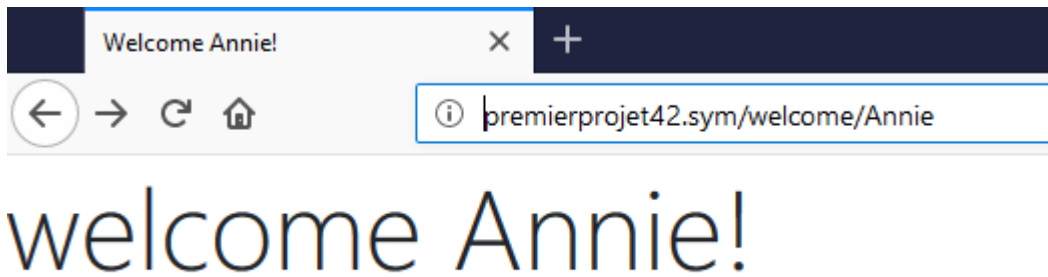
Et Pour cela, rendez-vous sur le site de Bootstrap (<https://getbootstrap.com/>) et cliquez sur « Get started », et nous allons faire un petit copié-collé pour importer les CDN dans le base.html.twig



Dans le fichier base.html.twig :



Modifiez votre vue welcome.html.twig pour y mettre du style :



Vous allez rajouter un bloc footer dans le fichier base.html.twig :

```
<footer>
    {% block footer %}
        mentions légales
    {% endblock %}
</footer>
</body>
```

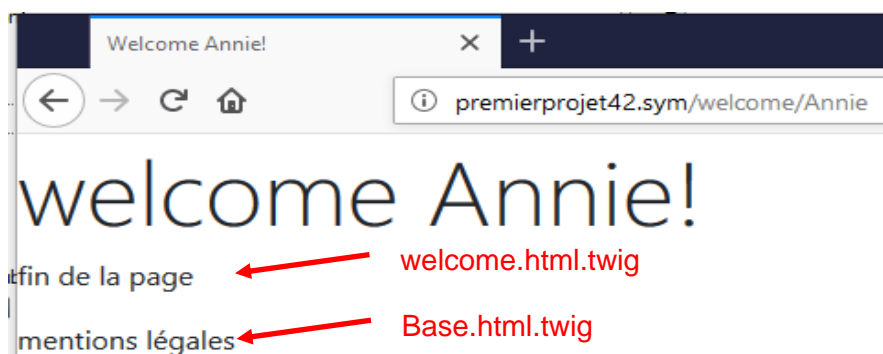
Puis à la fin du template welcome.html.twig :

```
{% endblock %}
{% block footer %}
    <p>fin de la page <p>
    {{ parent() }}
{% endblock %}
```

Surcharge du block parent footer

Vous venez de surcharger le bloc footer du template principal

<http://premierprojet42.sym/welcome/Annie>

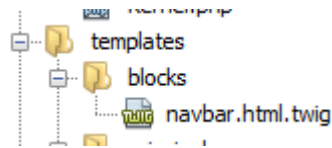


3. Création d'une barre de navigation

Dans l'application que vous allez développer, vous aurez sans doute besoin d'une barre de navigation (navbar) dans laquelle vous aurez différents liens.

Il est évident que ce sera la même barre de navigation sur toutes les pages de votre application. Vous allez donc créer une fois pour toutes cette navbar.

Vous créez donc un fichier navbar.html.twig dans le dossier templates\blocks :



Ce template contiendra donc le code de votre navbar, et vous l'appellerez ensuite dans toutes vos pages !

Bilan : un seul fichier à modifier, pas d'oubli(s) d'où un gain de temps appréciable. Pour l'instant le code à écrire est simple, il évoluera au fur et à mesure du TD :

```
{% extends 'base.html.twig' %}
{% block body %}
    <header>
        <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <a class="navbar-brand" href="#">TP Symfony</a>
        </nav>
    </header>
{% endblock %}
```



Vous remarquez que ce formulaire hérite du formulaire principal... important pour la suite !

Mais vous aurez à effectuer quelques modifications :

- ✓ Ajoutez le CDN de Font Awesome dans le block stylesheets de votre base.html.twig :

(copiez/collez ce lien)

<https://fontawesome.com/how-to-use/on-the-web/setup/getting-started?using=web-fonts-with-css>

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css">
% endblock %}
```

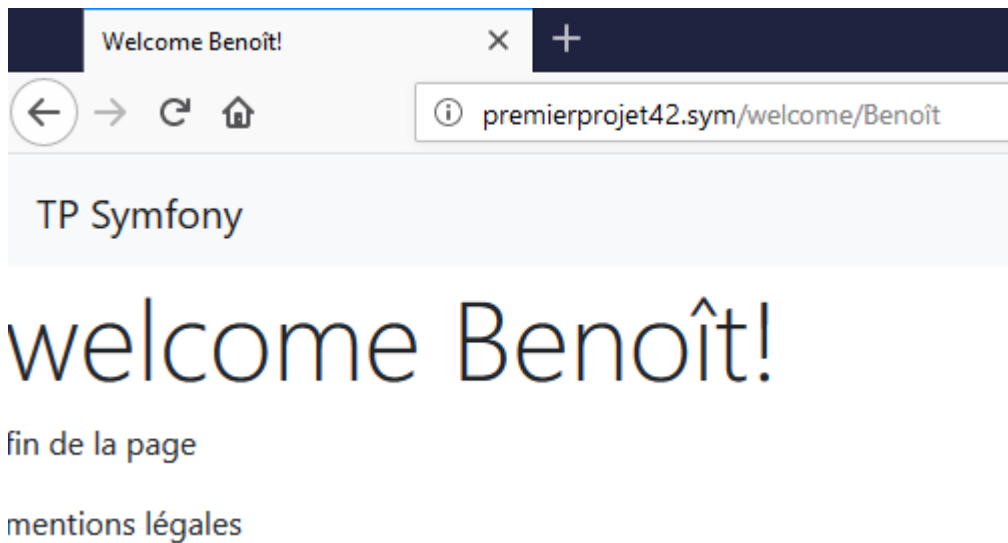
modifiez le template welcome.html.twig en le faisant hériter de navbar.html.twig et en surchargeant le bloc body :

```
{% extends 'blocks/navbar.html.twig' %}

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

{% block body %}
    {{ parent() }}
    <h1 class="display-4">welcome {{ nom }}! </h1
    </block>
```

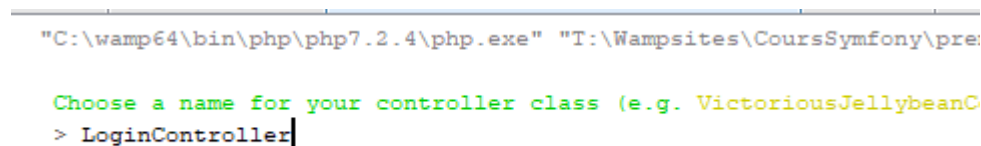
Relancez votre lien : <http://premierprojet42.sym/welcome/Benoît>



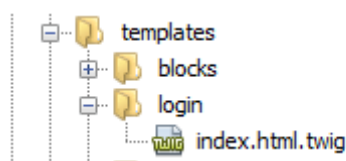
4. Formulaire de login

Maintenant vous allez créer un nouveau contrôleur que vous appellerez *LoginController*. N'oubliez pas, c'est une commande symfony :

make:controller



Dans l'arborescence du projet on peut voir qu'un nouveau dossier a été créé dans le dossier *templates*. Il s'agit du dossier contenant la page liée au contrôleur créé :



On y travaillera dessus un peu plus tard.

Vous aurez besoin dans cette partie de plusieurs packages. En effet, on ne crée pas sous symfony des formulaires comme dans une application php classique.

Vous allez donc installer les packages suivants (ainsi que leurs dépendances) :

- ✓ Form : pour générer les formulaires,
- ✓ Validator : pour valider les formulaires
- ✓ Orm : pour la persistance des objets (doctrine) puisqu'on va travailler avec une BD

Allez voir sur <https://symfony.sh/>

Et vous trouverez tous les bundles voulus :

symfony/form

official

Aliases **form**

[Package details](#)

symfony/validator

official

Aliases **validation** **validator**

[Package details](#) [Recipe](#)

symfony/orm-pack

official

Aliases **doctrine** **doctrine-orm** **orm**

orm-pack

[Package details](#) [Recipe](#)

Donc vous pouvez lancer les commandes :

- ✓ `Composer require symfony/form`
- ✓ `Composer require symfony/validator`
- ✓ `Composer require symfony/orm-pack`



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

Administrateur : Invite de commandes - composer require form

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

Administrateur : Invite de commandes

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

Administrateur : Invite de commandes

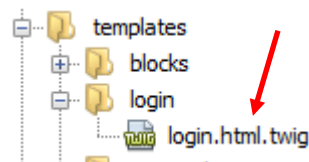
T:\Wampsites\CoursSymfony\premierprojet40>composer require orm

Vous pourrez rajouter pour les projets à venir ces packages dans le fichier loadrecipes.bat

Vérifiez ensuite que vos bundles sont installés :

- ✓ Form et validator dans le dossier vendor\symfony
- ✓ symfony/orm-pack dans le dossier vendor

Vous renommerez le fichier index.html.twig du dossier templates/login en login.html.twig :



Vous allez renommer la méthode *index* de la classe LoginController. Vous l'appellerez *login*.
Et vous modifierez également les paramètres de la route.
Vous obtiendrez donc ceci

```
class LoginController extends AbstractController
{
    /**
     * @Route("/login", name="login")
     */
    public function login()
    {
```

La fenêtre de login aura 2 zones de saisie (adresse mail, mot de passe), et un bouton pour soumettre les informations.

Vous utiliserez la méthode `createFormBuilder` de la classe `AbstractController` pour créer le formulaire.

Vous placerez le code de construction de formulaire dans la méthode privée `loginController` du `LoginController`

Vous aurez donc ce code au final dans votre classe `LoginController` :

```
class LoginController extends AbstractController {

    /**
     * @Route("/login", name="login")
     */
    public function login() {
        $formLogin = $this->createFormLogin();
        return $this->render('login/login.html.twig',
            array(
                'formLogin' => $formLogin->createView(),
                'title' => 'Login'
            )
        );
    }

    private function createFormLogin() {
        return $this->createFormBuilder()
            ->add('mailAddress', EmailType::class)
            ->add('password', PasswordType::class)
            ->add('submit', SubmitType::class)
            ->getForm();
    }
}
```

Sans oublier les références aux espaces de noms :

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type>PasswordType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class LoginController extends AbstractController {
```

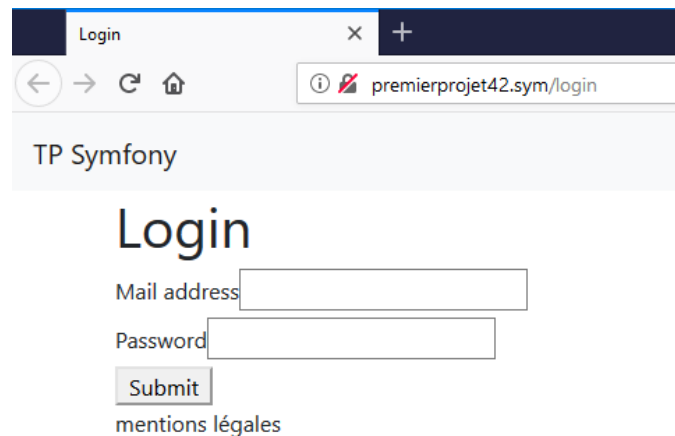
Puis vous modifierez le template login.html.twig pour inclure votre formulaire :

```
{% extends 'blocks/navbar.html.twig' %}

{% block title %}{{ title }}{% endblock %}

{% block body %}
    {{ parent() }}
    <div class="container">
        <section>
            <h1 class="display4">Login</h1>
            <article>
                {{ form_start(formLogin) }}
                {{ form_end(formLogin) }}
            </article>
        </section>
    </div>
{% endblock %}
```

Testez : <http://premierprojet42.sym/login>



Ce n'est pas joli, mais on va faire mieux

Il suffit de travailler un peu le formulaire dans le template, et de rajouter des paramètres à chacun des éléments :

```
<div class="container">
    <section>
        <h1 class="display4">Login</h1>
        <article>
            {{ form_start(formLogin) }}
            <div class='form-group'>
                {{ form_widget(formLogin.mailAddress, {'attr' : {'class': 'form-control'}}) }}
            </div>
            <div class='form-group'>
                {{ form_widget(formLogin.password, {'attr' : {'class': 'form-control'}}) }}
            </div>
            {{ form_widget(formLogin.submit, {'attr' : {'class': 'btn btn-primary'}}) }}
            {{ form_end(formLogin) }}
        </article>
    </section>
</div>
```



Pour plus de renseignements : <https://symfony.com/doc/current/forms.html>

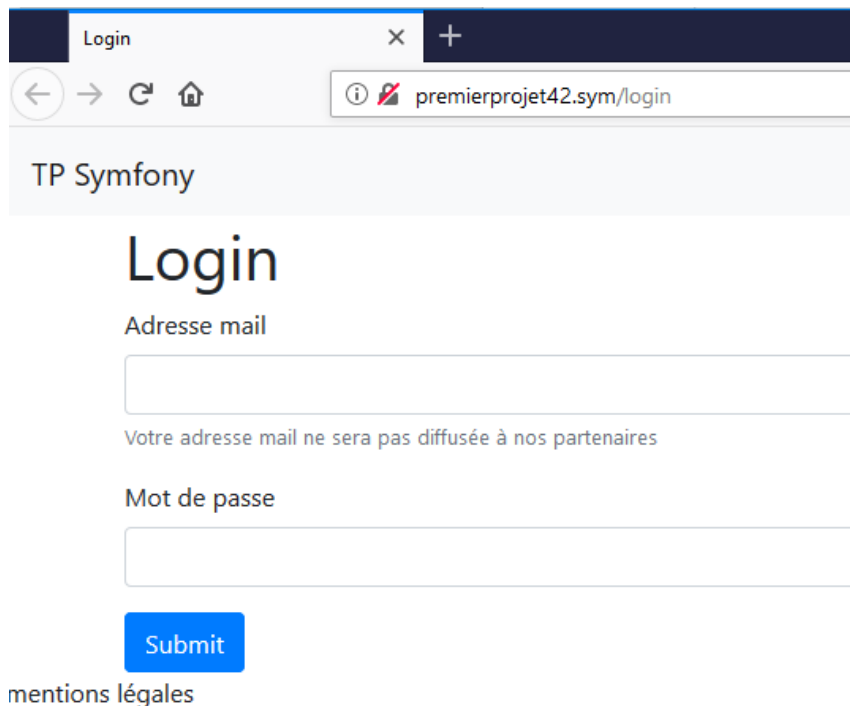
C'est déjà plus propre. Mais il nous manque les labels...

Encore une fois la documentation est d'une grande aide :

<https://symfony.com/doc/current/reference/forms/types/form.html>

```
<div class='form-group'>
    {{ form_label(formLogin.mailAddress, 'Adresse mail') }}
    {{ form_widget(formLogin.mailAddress, {'attr' : {'class': 'form-control'}}) }}
    <small class="form-text text-muted">Votre adresse mail ne sera pas diffusée à nos partenaires
</div>
<div class='form-group'>
    {{ form_label(formLogin.password, 'Mot de passe') }}
    {{ form_widget(formLogin.password, {'attr' : {'class': 'form-control'}}) }}
</div>
```

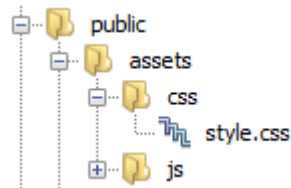
Résultat : <http://premierprojet42.sym/login>




Oups Nous avons oublié le javascript ... Et donc pas défini la classe display-4

```
<section>
  <h1 class="display-4">Login</h1>
  <article>
    {{ form_start(formLogin) }}
    <div class='form-group'>
```

Dans le dossier public, créez un dossier assets qui contiendra les dossiers css et js puis dans le dossier css un fichier style.css :



Et dans le fichier css:

```
form {
  width: 50%;
  margin: auto;
}

.display-4 {
  text-align: center;
}
```

Pour que ce fichier puisse être chargé, il faut l'indiquer au formulaire de base. Vous modifierez donc le fichier base.html.twig :

```
<link rel="stylesheet" href="{{ asset('css/style.css') }}">
```



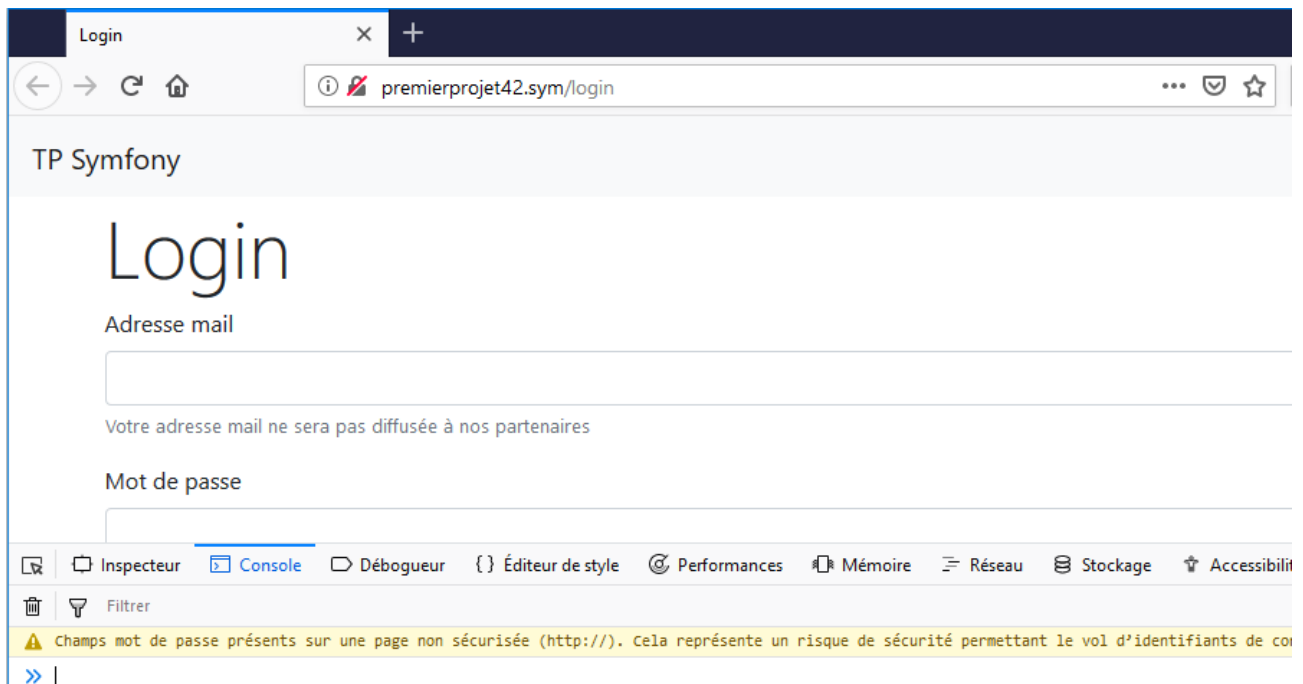
La fonction asset va se charger de résoudre l'url du fichier css et de le charger dans la page....

Avec un peu plus d'expérience, vous placerez vos fichiers css et js en dehors du dossier public et vous travaillerez avec le package symfony/webpack-encore-pack

Résultat : <http://premierprojet42.sym/login>

Vous irez voir dans la console si le fichier a bien été chargé.

Le navigateur râle parce que vous n'êtes pas en https, mais on verra ça plus loin.



Il vous suffit maintenant de compléter la barre de navigation :

```
<a class="nav-item" href="#"></a>
<li class="nav-item">
  <a class="nav-link" href="{ path('login') }"><i class="fas fa-sign-in-alt"></i> Login</a>
</li>
```

Le path contient le nom de la route correspondante à la page que nous souhaitons (voir l'annotation de la méthode login du contrôleur).

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

Et vous avez une belle barre de navigation !!!



C'est super, vous venez de terminer votre premier formulaire symfony !!!

Partie 9 : FORMULAIRE DE CREATION D'UN UTILISATEUR

Vous allez maintenant travailler avec les entités et la base de données.

Assurez-vous que le package symfony/orm-pack a bien été importé (composer.json) :

```
"symfony/orm-pack": "^1.0",
```


1. Création de l'entity User

Vous allez créer votre première entity.

Une Entity est une classe qui sera mappée sur une table de la BD.

Vous créerez l'entity User (au singulier) qui sera composée de 4 attributs :

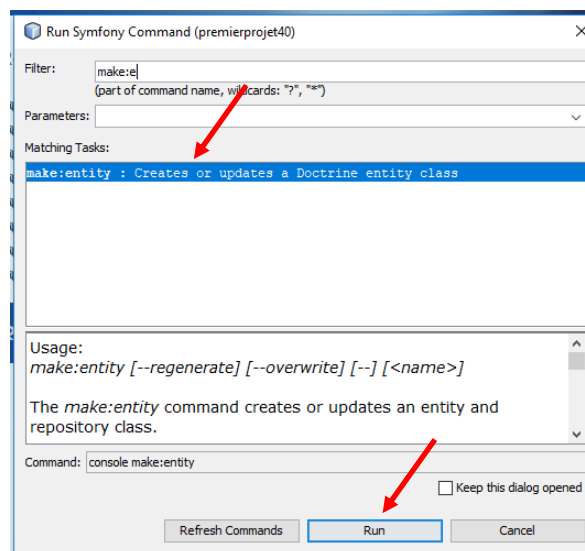
- ✓ nom (string, 60, not null)
- ✓ prenom (string, 60, not null)
- ✓ mail (string, 60, not null)
- ✓ password (string, 255, not null)

Vous allez utiliser la commande make:entity



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

Respectez aussi la casse !!! Norme PSR-4



Saisie du nom de l'entity et de la propriété nom :

```
"C:\wamp64\bin\php\php7.2.4\php.exe" "T:\Wampsites\CoursSymfony\premierprojet42\bin\console" "--ansi" "make:entity"

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

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

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/User.php

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

Vous continuerez pour les prénom, mail et password

A la fin vous devriez avoir ceci :

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

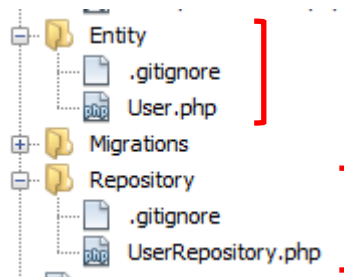
```
Add another property? Enter the property name (or press <return> to stop adding fields):
>
```

Success!

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

```
Done.
!
```

Si vous retournez dans NetBeans, vous verrez qu'un dossier Entity a été créé, ainsi qu'un dossier Repository:



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

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

On y voit des annotations sur :

La classe : elle indique où se trouve le

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

Les champs : qui fournissent les éléments du mapping avec la BD

```
/**
 * @ORM\Id()
 * @ORM\GeneratedValue()
 * @ORM\Column(type="integer")
 */
private $id;

/**
 * @ORM\Column(type="string", length=60)
 */
private $nom;
```

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

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

    /**
     * @ORM\Column(type="string", length=60)
     */
    private $nom;

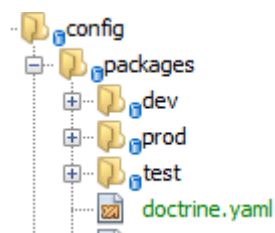
    /**
     * @ORM\Column(type="string", length=60)
     */
    private $prenom;

    /**
     * @ORM\Column(type="string", length=60)
     */
}
```

Vous pourrez ainsi créer d'autres Entities.

2. Configuration de la BD

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: '%env(resolve:DATABASE_URL) %'
```

Et rajoutez ces paramètres dans la partie DBAL :

```
# 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:

Symfony stocke ses paramètres dans 2 fichiers : .env et .env.dist

Rajoutez les paramètres suivants au fichier .env:

```
# Configure your db driver and server_version in config/packages/doctrine.yaml
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
DATABASE_USER=benoit
DATABASE_PWD=benoit
DATABASE_NAME=dbcourssymfony
###< doctrine/doctrine-bundle ###
```

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

```
DATABASE_USER=benoit
DATABASE_PWD=benoit
DATABASE_NAME=dbcourssymfony
```



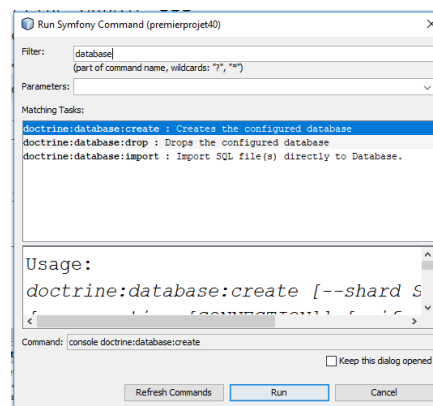
Assurez-vous que l'utilisateur que vous choisirez a bien les droits de création de base sur le SGBD ...

Au pire utilisez root sans mot de passe (config par défaut !!!)

User	From Host	Login	Account Limits	Administrative Roles	Schema Privileges
benoit	%				
dbuser	localhost				
greta	%				
mysql.session	localhost				
mysql.sys	localhost				
root	localhost				
symfony	%				
test	%				
userGsb	localhost				

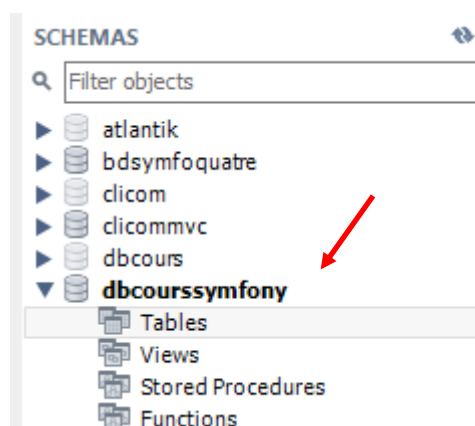
Role	Description
<input type="checkbox"/> DBA	grants the rights to perform all tasks
<input type="checkbox"/> MaintenanceAdmin	grants rights needed to maintain server
<input type="checkbox"/> ProcessAdmin	rights needed to assess, monitor, and kill any user proce...
<input type="checkbox"/> UserAdmin	grants rights to create users logins and reset passwords
<input type="checkbox"/> SecurityAdmin	rights to manage logins and grant and revoke server an...
<input type="checkbox"/> MonitorAdmin	minimum set of rights needed to monitor server
<input checked="" type="checkbox"/> DBManager	grants full rights on all databases
<input checked="" type="checkbox"/> DBDesigner	rights to create and reverse engineer any database sche...
<input type="checkbox"/> ReplicationAdmin	rights needed to setup and manage replication
<input checked="" type="checkbox"/> BackupAdmin	minimal rights needed to backup any database

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



Search Results	Output - Symfony 3 (premierprojet42) X	Usages	Git Repository Browser
	<pre>"C:\wamp64\bin\php\php7.2.4\php.exe" "T:\Wampsites\CoursSymfony\premierprojet42\bin\console" "--ansi" "doctrine:database:create" Created database `dbcourssymfony` for connection named default Done.</pre>		

La base de données a été créée (sans la table ...):



Vous allez maintenant mettre à jour la BD

- ✓ générer les ordres SQL de synchronisation des tables par rapport aux entités.
- ✓

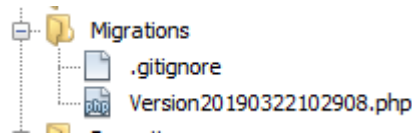
commande `doctrine:migrations:diff`

```
"C:\wamp64\bin\php\php7.2.4\php.exe" "T:\Wampsites\CoursSymfony\premierprojet42\bin\console" "--ansi" "doctrine:migrations:diff"
Generated new migration class to "T:\Wampsites\CoursSymfony\premierprojet42\src\Migrations\Version20190322102908.php"

To run just this migration for testing purposes, you can use migrations:execute --up 20190322102908

To revert the migration you can use migrations:execute --down 20190322102908
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 !!!

```
final class Version20190322102908 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');

        $this->addSql('CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(255) NOT NULL, PRIMARY KEY(id))');
    }

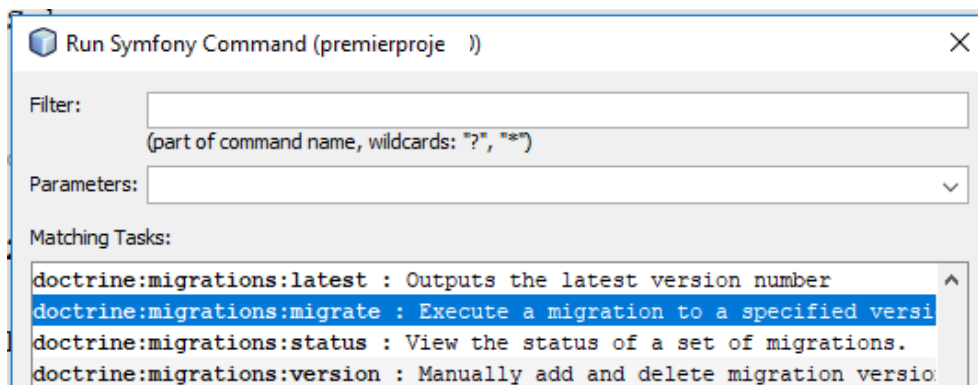
    public function down(Schema $schema) : void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('DROP TABLE user');
    }
}
```

Appliquez les modifications apportées aux entités sur la BD :

- ✓ Exécuter la commande

doctrine:migrations:migrate

Exécutez maintenant la commande doctrine:migrations:migrate :



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

```
"C:\wamp64\bin\php\php7.2.4\php.exe" "T:\Wampsites\CoursSymfony\premierprojet42\bin\console" "--ansi" "doctrine:migrations:migrate"
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) Application Migrations
y
Migrating up to 20190322102908 from 0
++ migrating 20190322102908
--> CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(60) NOT NULL, prenom VARCHAR(60) NOT NULL, mail VARCHAR(60) NOT NULL, password VARCHAR(255) NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET
++ migrated (took 757.8ms, used 14M memory)

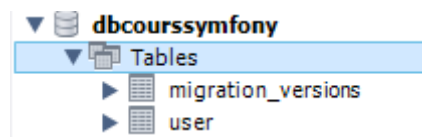
-----

++ finished in 822.5ms
++ used 14M memory
++ 1 migrations executed
++ 1 sql queries
Done.
```

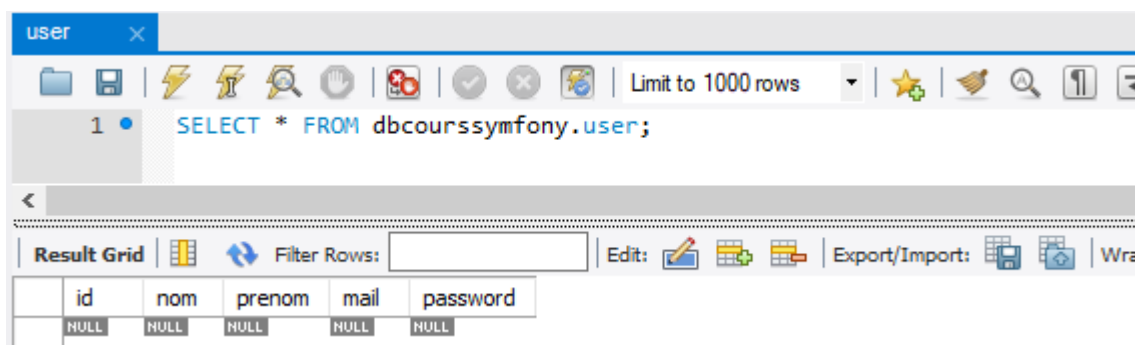
On voit que le fichier de migration a été exécuté et donc l'ordre SQL de création de la table :

```
++ migrating 20190322102908
--> CREATE TABLE user (id INT !
```

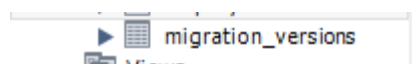
Ouvrez votre client mysql : MysqlWorkbench ou phpMyAdmin et constatez que la table a été créée dans la bonne BD :



Faites un select * from dbcourssymfony.user et voir que la table ... est vide !!!



Notez aussi la création de la table des migrations :



Faites un SELECT * FROM dbcourssymfony.migration_versions pour vérifier le contenu :

Vous voyez le contenu :

	version	executed_at
	20190322102908	2019-03-22 10:36:16
	NULL	NULL

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

On y reviendra.

3. Création du formulaire d'enregistrement d'un utilisateur de l'application

Vous allez maintenant revenir à symfony et créer le formulaire de création d'un utilisateur de l'application.

Vous allez créer un nouveau contrôleur : RegistrationController avec la commande make:controller

```
"C:\wamp64\bin\php\php7.2.4\php.exe" "T:\Wampsites\CoursSymfony\premierprojet42\bin\console" "--ansi" "make:controller"

Choose a name for your controller class (e.g. BravePizzaController):
> RegistrationController

created: src/Controller/RegistrationController.php
created: templates/registration/index.html.twig

Success!

Next: Open your new controller class and add some pages!
Done.
|
```

Ce contrôleur va implémenter les méthodes nécessaires pour enregistrer un utilisateur.

Il va donc falloir créer un formulaire d'enregistrement.

Mais cette fois vous n'allez pas implémenter la méthode *createFormBuilder()* de création de formulaire dans ce contrôleur. Eh oui !

Il existe une deuxième manière de créer un formulaire.

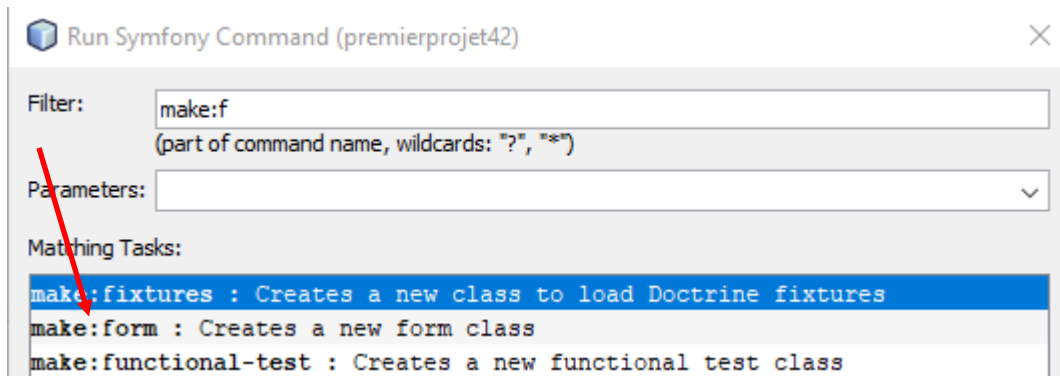
Je vous présente les deux, pour vous montrer qu'il existe plusieurs façons de créer de faire appel à des formulaires dans des contrôleurs.

Vous pourrez ensuite choisir l'une ou l'autre méthode. Si vous avez beaucoup de code dans votre contrôleur, il est préférable de gérer votre formulaire dans un fichier à part.

L'idée est de créer un formulaire indépendant du contrôleur, et donc qui pourra être appelé depuis n'importe quel endroit de l'application.

Le formulaire sera géré par une classe dédiée dans le dossier src/Form

Vous allez donc créer le formulaire UserType (Type est obligatoire comme suffixe) avec la commande symfony *make:form*



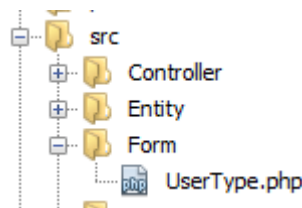
... il vous le dit : Create a new form class !!!

Répondez aux 2 questions posées :

- ✓ Le nom du formulaire (pour nous UserType)
- ✓ Le nom de l'entity rattachée (pour nous User)

La création s'est bien passée.

Un nouveau dossier (Form) et un nouveau fichier formulaire ont été créés.



Ouvrez ce fichier :

```
namespace App\Form;

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

class UserType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('nom')
            ->add('prenom')
            ->add('mail')
            ->add('password')
        ;
    }

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

Remarquez la méthode buildForm ... le code est déjà écrit !!!



Pour utiliser le formulaire, vous n'avez quasiment plus rien à faire !!!

Commencez par rajouter l'espace de noms suivant dans le contrôleur RegistrationController :

```
use Symfony\Component\Routing\Annotation\Route;
use App\Form\UserType;
```

Puis la méthode testForm dans le même contrôleur qui sera appelée avec l'URL :

<http://premierprojet42.sym/testform>

```
/**
 * @Route("/testform", name="testform")
 */
public function testForm(Request $request, RegistryInterface $doctrine) {
    $user = new User();
    $form = $this->createForm(UserType::class, $user);
    return $this->render('registration/testuser.html.twig', array(
        'form' => $form->createView()));
}
```

Vous n'oublierez pas les bons alias sur les espaces de noms :

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Bridge\Doctrine\RegistryInterface;
use App\Form\UserType;
use App\Entity\User;
```

Enfin, créez le formulaire testuser.html.twig dans le dossier

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

{% block title %}Employe{% endblock %}
{% block body %}
    {{ form_start(form) }}
    {{ form_row(form.nom) }}
    {{ form_row(form.prenom) }}
    {{ form_row(form.mail) }}
    {{ form_row(form.password) }}
    {{ form_end(form) }}
{% endblock %}
```

... il ne vous reste plus qu'à tester : <http://premierprojet42.sym/testform>

et voici le résultat :

Pas beau mais ... efficace !

Vous allez améliorer tout ça et surtout, vous allez créer un utilisateur.

Vous commencerez par modifier la classe UserType :

Attention aux Use...

```
namespace App\Form;

use App\Entity\User;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
use Symfony\Component\Form\Extension\Core\Type>PasswordType;
```

```
class UserType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('nom', TextType::class)
            ->add('prenom', TextType::class)
            ->add('mail', EmailType::class)
            ->add('password', RepeatedType::class, array(
                'type' => PasswordType::class,
                'first_options' => array('label' => 'Mot de passe'),
                'second_options' => array('label' => 'Répétez le mot de passe')
            ));
    }
}
```



Remarquez le mot de passe qui doit être confirmé !!!

Vous modifierez ensuite la méthode register de la classe RegistrationController :

Vous commencerez par placer les bonnes clauses Use :

```
namespace App\Controller;

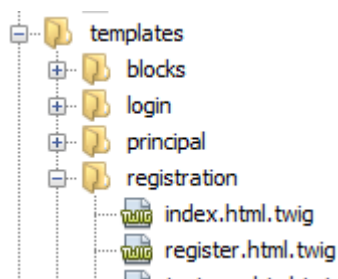
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Bridge\Doctrine\RegistryInterface;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;
use App\Form\UserType;
use App\Entity\User;

class RegistrationController extends AbstractController {
```

```
/**
 * @Route("/register", name="user_registration")
 */
public function register(Request $request) {
    // 1) construction du formulaire
    $user = new User();
    $form = $this->createForm(UserType::class, $user);

    // 2) Test si c'est un retour de formulaire
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // 3) si c'est un retour de formulaire on
        // stocke l'utilisateur en BD si le formulaire est valide
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($user);
        $entityManager->flush(); // Partie Doctrine
        // On rappelle le formulaire
        return $this->render(
            'registration/register.html.twig',
            array(
                'form' => $form->createView()
            )
        );
    }
    return $this->render(
        'registration/register.html.twig',
        array('form' => $form->createView())
    );
}
```

Puis vous créez le template register.html.twig dans le dossier templates/registration :



Et le code :

```
{% extends 'blocks/navbar.html.twig' %}

{% block title %}Register{% endblock %}

{% block body %}
    {{ parent() }}
    <div class="container">
        <section>
            <h1 class="display-4">Register</h1>
            <article>
                {{ form_start(form) }}
                <div class="form-group">
                    {{ form_row(form.nom, { 'attr': { 'class': 'form-control' } }) }}
                </div>
                <div class="form-group">
                    {{ form_row(form.prenom, { 'attr': { 'class': 'form-control' } }) }}
                </div>
                <div class="form-group">
                    {{ form_row(form.mail, { 'attr': { 'class': 'form-control' } }) }}
                </div>
                <div class="form-group">
                    {{ form_row(form.password.first, { 'attr': { 'class': 'form-control' } }) }}
                </div>
                <div class="form-group">
                    {{ form_row(form.password.second, { 'attr': { 'class': 'form-control' } }) }}
                </div>
                <button class="btn btn-primary" type="submit">Register</button>
                {{ form_end(form) }}
            </article>
        </section>
    </div>
{% endblock %}
```



Vous remarquerez la zone de répétition du mot de passe

Il ne vous reste plus qu'à tester : <http://premierprojet42.sym/register>

Renseignez les champs

Nom

Prenom

Mail

Mot de passe

Répétez le mot de passe

Et allez voir en base de données :

user

SELECT * FROM dbcourssymfony.user;

Result Grid

	id	nom	prenom	mail	password
1	1	Roche	Benoît	benoit.roche@gmail.com	roche
2	NULL	NULL	NULL	NULL	NULL



cool ... Votre utilisateur est enregistré !



Pas cool ... vous n'avez mis aucune sécurité !

Le mot de passe est stocké en clair dans la base de données

De plus, le formulaire de login et le formulaire d'enregistrement ne comportent aucune sécurité.

Il faut pour cela installer certains package et configurer le fichier config/packages/security.yaml. Le but de ce TP est de se familiariser avec l'environnement de Symfony en manipulant les entités. Vous verrez la sécurité un peu plus loin dans ce TD...

Quelques liens :

https://symfony.com/doc/current/security/entity_provider.html
https://symfony.com/doc/current/doctrine/registration_form.html
http://symfony.com/doc/current/security/form_login_setup.html
<https://symfony.com/doc/current/security.html>

Partie 10 : PAGE D’AFFICHAGE DES INFORMATIONS D’UN UTILISATEUR

Maintenant que vous avez créé les formulaires de login et d’enregistrement, il faudrait une page où afficher les informations de l’utilisateur qui vient de s’identifier.

Vous ferez une page très simple, l’idée est de vous faire récupérer une information de la base de données en utilisant la puissance de doctrine.

Dans le contrôleur LoginController, créez la méthode privée fetchUser() :

```
private function fetchUser(string $mail, string $password) {
    $repository = $this->getDoctrine()->getManager()->getRepository(User::class);
    $result = $repository->findOneBy(
        array(
            'mail' => $mail,
            'password' => $password
        )
    );
    return $result;
}
```

Cette méthode va permettre de récupérer l’enregistrement en base de données dont l’adresse mail et le mot de passe correspondent aux paramètres passés grâce à la méthode findOneBy de la classe ServiceEntityRepository dont la classe UserRepository hérite.

```
$result = $repository->findOneBy(
    array(
        'mail' => $mail,
        'password' => $password
    )
)
```

Cette méthode va donc renvoyer une instance de la classe User.
Ensuite, modifiez la méthode login() du même contrôleur :



**Vous serez amenés à rajouter des alias sur des classes (des use....)
Mais là, je vous laisse faire ... vous regarderez bien les messages !!!**

```
public function login(Request $request) {
    $formLogin = $this->createFormLogin();
    $formLogin->handleRequest($request);
    if ($formLogin->isSubmitted() && $formLogin->isValid()) {
        $mail = $formLogin['mailAddress']->getData();
        $password = $formLogin['password']->getData();
        $user = $this->fetchUser($mail, $password);
        return $this->render('login/user.html.twig',
            array(
                'user' => $user,
                'title' => "info login"
            )
        );
    }
    return $this->render('login/login.html.twig',
        array(
            'formLogin' => $formLogin->createView(),
            'title' => 'Login'
        )
    );
}
```

Si le formulaire renvoyé est valide, on affiche le formulaire user.html.twig

Cette ligne va nous permettre de récupérer l'adresse mail saisie dans le formulaire (après validation).

```
$mail = $formLogin['mailAddress']->getData();
```

Ensuite on passe l'objet au formulaire appelé (avec le titre de la page) :

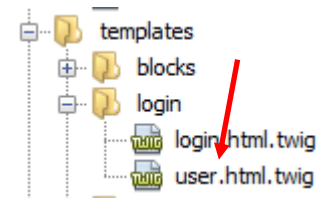
```
return $this->render('login/user.html.twig',
    array(
        'user' => $user,
        'title' => "info login"
    )
);
```

Il ne vous reste plus qu'à créer le formulaire d'affichage templates/user.html.twig

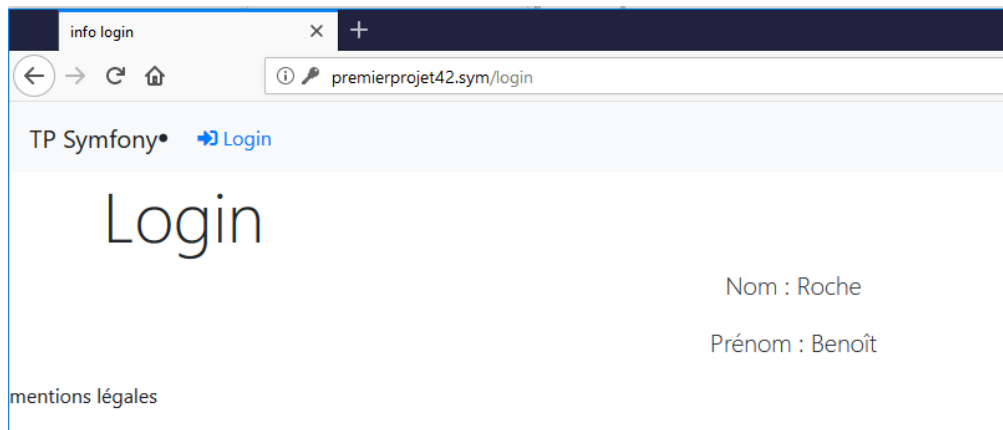
```
{% extends 'blocks/navbar.html.twig' %}

{% block title %}{{ title }}{% endblock %}

{% block body %}
    {{ parent() }}
    <div class="container">
        <section>
            <h1 class="display-4">Login</h1>
            <utilisateur class="text-center">
                <p class="lead">Nom : {{ user.nom }}</p>
                <p class="lead">Prénom : {{ user.prenom }}</p>
            </utilisateur>
        </section>
    </div>
{% endblock %}
```



Il ne vous reste plus qu'à tester : <http://premierprojet42.sym/login>



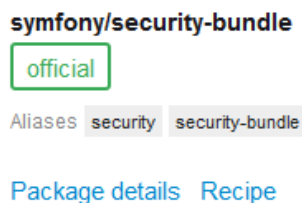
Ça a marché ... Vous avez pu récupérer un enregistrement de la base de données sur plusieurs critères et à l'afficher dans un formulaire !!!

Partie 11 : FORMULAIRE D'ENREGISTREMENT SECURISE

L'objectif dans cette partie est de sécuriser l'application tant au niveau des formulaires que du chargement des entités depuis la base de données.

Symfony vous fournit les outils pour assurer la sécurité des applications grâce au package `symfony/security-bundle`.

Vous installerez donc ce recipe :



Je vous invite donc à l'installer en ligne de commande et à le rajouter dans votre fichier `loadrecipes.bat` :

```

T:\Wampsites\CoursSymfony\premierprojet42>composer require symfony/security-bundle
Restricting packages listed in "symfony/symfony" to "4.2.*"
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.2.*"
Package operations: 5 installs, 0 updates, 0 removals
  - Installing symfony/security-core (v4.2.4): Loading from cache
  - Installing symfony/security-http (v4.2.4): Loading from cache
  - Installing symfony/security-guard (v4.2.4): Loading from cache
  - Installing symfony/security-csrf (v4.2.4): Loading from cache
  - Installing symfony/security-bundle (v4.2.4): Loading from cache
Writing lock file
Generating autoload files
ocramius/package-versions: Generating version class...
ocramius/package-versions: ...done generating version class
Symfony operations: 1 recipe (6677dc76f7d57c3bb957f68dc6c715ee)
  - Configuring symfony/security-bundle (>=3.3): From github.com/symfony/recipes:master
Executing script cache:clear [OK]
Executing script assets:install public [OK]
Executing script security-checker security:check [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\premierprojet42>
    
```

Votre package est maintenant installé. Vous pouvez vérifier dans le fichier composer.json

```
"symfony/profiler-pack": "^1.0",
"symfony/security-bundle": "4.2.*",
"symfony/twig-bundle": "4.2.*"
```

Ce package vous permettra entre autre d'implémenter l'interface `UserInterface` sur les Entities que vous créerez.



Une interface définit le comportement d'une classe. Tous les éléments d'une interface doivent être implémentés dans la classe qui y fait appel.

Ici cette interface possède les méthodes suivantes :

- ✓ `getRoles()` : retourne les privilèges de l'utilisateur (`ROLE_USER`, `ROLE_ADMIN`, etc...)
- ✓ `getPassword()` : retourne le mot de passe utilisé pour authentifier l'utilisateur
- ✓ `getSalt()` : retourne le sel utilisé pour chiffrer le mot de passe
- ✓ `getUsername()` : retourne l'identifiant de l'utilisateur
- ✓ `eraseCredential()` : supprime les données sensibles de l'utilisateur

Vous pourrez trouver le détail de ces méthodes à l'url :

<https://api.symfony.com/4.2/Symfony/Component/Security/Core/User/UserInterface.html>



Le bundle précédemment installé fournit également une classe `User` prédéfinie, mais celle-ci contient des propriétés qui ne nous seront pas utiles dans notre cas, et il y a des propriétés dont nous avons besoin et qui ne sont pas présentes. Il est donc préférable de créer notre propre entité et de configurer ensuite le tout pour avoir une solution plus sur-mesure.

Lien vers la documentation de `User` :

<https://api.symfony.com/4.2/Symfony/Component/Security/Core/User/User.html>

Commencez par modifier votre entité `User` et faites la implémenter l'interface `UserInterface` :

```
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 */
class User implements UserInterface
{
```

Mais aussi dans la même classe :

```
/**
 * @Assert\NotBlank()
 * @Assert\Length(max=4096)
 */
private $plainPassword;

public function getPlainPassword(): ?string
{
    return $this->plainPassword;
}

public function setPlainPassword(string $plainPassword): void
{
    $this->plainPassword = $plainPassword;
}

public function getUsername(): string
{
    return $this->mail;
}

public function getSalt()
{
    return null;
}

public function getRoles(): array
{
    return array('ROLE_USER');
}

public function eraseCredentials()
{
}
```

La propriété \$plainPassword n'est pas à persister en base de données. Il s'agit d'une propriété qui accueillera le mot de passe en clair de l'utilisateur pour pouvoir ensuite le chiffrer et le stocker dans \$password.

nous allons utiliser l'adresse mail en tant que "username".

Et enfin , vous allez créer les méthodes serialize et unserialize.



À quoi servent les méthodes serialize et unserialize? ¶

À la fin de chaque requête, l'objet User doit être sérialisé en variable de session. Cette méthode va aider PHP à effectuer correctement la sérialization. Toutefois, vous n'avez pas besoin de tout sérialiser. Vous n'avez besoin que de quelques champs (ceux indiqués ci-dessous plus quelques suppléments si vous décidez d'implémenter) À

chaque demande, l'id est utilisé pour rechercher un nouvel objet utilisateur dans la base de données.

Pour en savoir plus :

Voir https://symfony.com/doc/4.0/security/entity_provider.html#security-serialize-equatable

```
/**
 * @see \Serializable::serialize()
 */
public function serialize()
{
    return serialize(array(
        $this->id,
        $this->mail,
        $this->password
    ));
}

/**
 * @see \Serializable::unserialize()
 *
 * @param $serialized
 */
public function unserialize($serialized)
{
    list (
        $this->id,
        $this->mail,
        $this->password
    ) = unserialize($serialized, array('allowed_classes' => false));
}
```

Il vous faut maintenant passer à la phase de configuration de symfony. Vous définirez notamment l'algorithme de chiffrement pour le mot de passe et le provider pour le pare-feu de symfony.

Vous aurez donc à modifier le fichier config/packages/security.yml :

```
security:
    # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
    encoders:
        App\Entity\User:
            algorithm: bcrypt
    providers:
        #in_memory: { memory: ~ }
        td_symfony:
            entity:
                class: App\Entity\User
                property: mail

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
            pattern: ^/
            http_basic: ~
            provider: td_symfony
```

Voyons ce fichier en détails :

```
encoders:
    App\Entity\User:
        algorithm: bcrypt

providers:
    #in_memory: { memory: ~ }
    td_symfony:
        entity:
            class: App\Entity\User
            property: mail

main:
    anonymous: true
    pattern: ^/
    http_basic: ~
    provider: td_symfony
```

On définit l'algorithme de chiffrement pour le mot de passe et par quelle entité il va être utilisé.

Le provider (ou fournisseur), que j'ai appelé `td_symfony`, va dire au pare-feu de Symfony qui est le fournisseur d'entités. Le pare-feu va donc aller chercher les utilisateurs dans la base de données. Il cherchera non pas par l'id mais par l'adresse mail qui jouera le rôle d'identifiant.

Il s'agit du pare-feu principal de l'application. `anonymous` signifie qu'on peut y accéder sans s'identifier

- ✓ `pattern` est un masque qui définit sur quelle URL agit le pare-feu (ici il s'agit de la racine donc le pare-feu agit sur tout le site)
- ✓ `provider` : le fournisseur (provider) du pare-feu

Maintenant vous allez apporter une petite modification à votre formulaire d'enregistrement dans le fichier `/src/Form/UserType.php`.

Remplacez, dans la méthode `buildForm` :

```
->add('password', RepeatedType::class, array(
```

Par :

```
->add('plainPassword', RepeatedType::class, array(
```

Puis modifier le contrôleur `RegistrationController` en ajoutant le « use » qui va bien :

```
use App\Entity\User;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class RegistrationController extends AbstractController {
```

Puis vous allez rajouter à la méthode `register` le paramètre `$passwordEncoder` de la classe `UserPasswordEncoderInterface`. Cette classe fournit toutes les méthodes pour pouvoir encoder les mots de passe.

```
* @Route("/register", name="user_registration")
*/
public function register(Request $request, UserPasswordEncoderInterface $passwordEncoder) {
```

Ensuite, toujours dans la même classe, vous allez encoder le mot de passe saisi et récupéré par le contrôleur :

```
// (3) encoder le mot de passe password
$password = $passwordEncoder->encodePassword($user, $user->getPlainPassword());
$user->setPassword($password);
// 4) si c'est un retour de formulaire on
// stocke l'utilisateur en BD si le formulaire est valide
$entityManager = $this->getDoctrine()->getManager();
$entityManager->persist($user);
```

Il faut aussi modifier votre vue /templates/registration/register.html.twig afin de remplacer *password* par *plainPassword* :

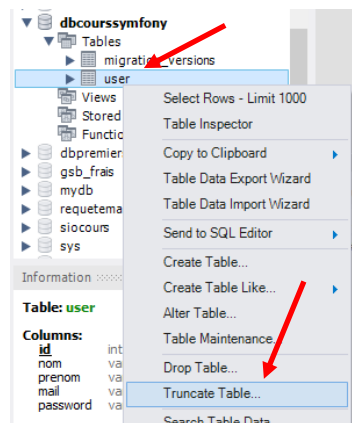
```
</div>
<div class="form-group">
    {{ form_row(form.plainPassword.first, {'attr': {'class': 'form-control'}} )}}
</div>
<div class="form-group">
    {{ form_row(form.plainPassword.second, {'attr': {'class': 'form-control'}} )}}
</div>
```

Maintenant vous allez réinitialiser la table *user* de votre base de données.

- ✓ Soit en exécutant la commande

```
truncate table dbcourssymfony.user;
```

- ✓ Soit en faisant un clic droit sur celle-ci puis *Truncate table...* :



Lorsque vous souhaitez réinitialiser le contenu d'une table, il est préférable d'utiliser l'instruction `TRUNCATE TABLE [nom_table]` plutôt que `DELETE FROM [nom_table]`. En effet, la première réinitialise la table en la remettant dans son état d'origine et en réinitialisant les id à 0 tandis que la deuxième ne fait que supprimer les données.

Admettons que nous ayons une table contenant dix enregistrements, les id vont de 1 à 10. Lorsque que vous allez faire persister une nouvelle entité :

- ✓ avec l'instruction `DELETE FROM [nom_table]`, l'id de cette nouvelle entité sera 11
- ✓ avec l'instruction `TRUNCATE TABLE [nom_table]`, l'id sera 1.

Vérifiez que votre table est vide :

1 • `SELECT * FROM dbcourssymfony.user;`

Result Grid | Filter Rows: | Edit: |

id	nom	prenom	mail	password
NULL	NULL	NULL	NULL	NULL

Et retournez dans votre navigateur à l'adresse pour enregistrer un nouvel utilisateur :
<http://premierprojet42.sym/register>

Register

Nom

Prenom

Mail

Mot de passe

Répétez le mot de passe

[Register](#)

[mentions légales](#)

Et vérifiez dans la bd :

1 • `SELECT * FROM dbcourssymfony.user;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

id	nom	prenom	mail	password
1	Roche	Benoît	benoit.roche@gmail.com	\$2v\$13\$zM5CNMvih9u9mvO7PFaYUeLX9nDbFw2ki6aI0OJJ10wVf2GTiUo62



Le chiffrement du mot de passe a bien fonctionné.

Partie 12 : FORMULAIRE DE LOGIN ET CHARGEMENT D'UN UTILISATEUR

Dans cette partie, vous allez modifier votre formulaire de login. L'idée cette fois est de conserver le login et le password de l'utilisateur tout au long de sa navigation.

Vous allez donc faire en sorte que vos identifiants de connexion soient enregistrés dans une variable de session.

La première chose à modifier est une nouvelle fois fichier `security.yml` en ajoutant les lignes suivantes :

```
main:
    anonymous: true
    pattern: ^/
    http_basic: ~
    provider: td_symfony
    form_login:
        login_path: login
        check_path: login
        default_target_path: user
```

Explications :

- ✓ form_login : c'est la méthode d'authentification utilisée par le pare-feu
- ✓ login_path : correspond à la route du formulaire de connexion, il s'agit de la route login que nous avons définie pour ce formulaire.
- ✓ check_path : correspond à la route de validation du formulaire de connexion, c'est sur cette route que seront vérifiés les identifiants saisis par l'utilisateur.
- ✓ Default_target_path : contient la route ciblée après qu'on se soit connecté.

Vous aurez aussi à modifier la méthode login de la classe LoginController :

```
/**
 * @Route("/login", name="login")
 */
public function login(AuthenticationUtils $authenticationUtils) {
    $error = $authenticationUtils->getLastAuthenticationError();
    $lastUsername = $authenticationUtils->getLastUsername();
    return $this->render('login/login.html.twig', array(
        'last_username' => $lastUsername,
        'error' => $error,
        'title' => "login"
    ));
}
```

- ✓ \$error va contenir l'erreur de login s'il y en a une (erreur dans les éléments saisis)
- ✓ \$lastUsername va contenir le nom d'utilisateur renseigné dans le formulaire de login

Et les use qui vont bien

```
use App\Entity\User;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;
use Symfony\Component\Security\Core\Security;
```

Les autres méthodes de la classe ne nous seront plus utiles.

Vous corrigerez maintenant la vue associée : login.html.twig

```
{% extends 'blocks/navbar.html.twig' %}

{% block title %}{{ title }}{% endblock %}

{% block body %}
    {{ parent() }}
    <div class="container">
        <section>
            <h1 class="display-4">Login</h1>
            {% if error %}
                <div class="alert alert-warning text-center">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
            {% endif %}
            <article>
                <form action="{{ path('login') }}" method="post">
                    <div class="form-group">
                        <label for="username">Email :</label>
                        <input type="text" class="form-control" id="username" aria-describedby="emailHelp" name="_username" value="{{ last_username }}">
                        <small id="emailHelp" class="form-text text-muted">Votre adresse mail ne sera pas diffusée à nos partenaires</small>
                    </div>
                    <div class="form-group">
                        <label for="password">Password :</label>
                        <input type="password" class="form-control" id="password" name="_password">
                    </div>
                    <button type="submit" class="btn btn-primary">Login</button>
                </form>
            </article>
        </section>
    </div>
{% endblock %}
```

Enfin, vous allez compléter la barre de navigation (demandez au professeur qu'il vous donne le code !!)

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

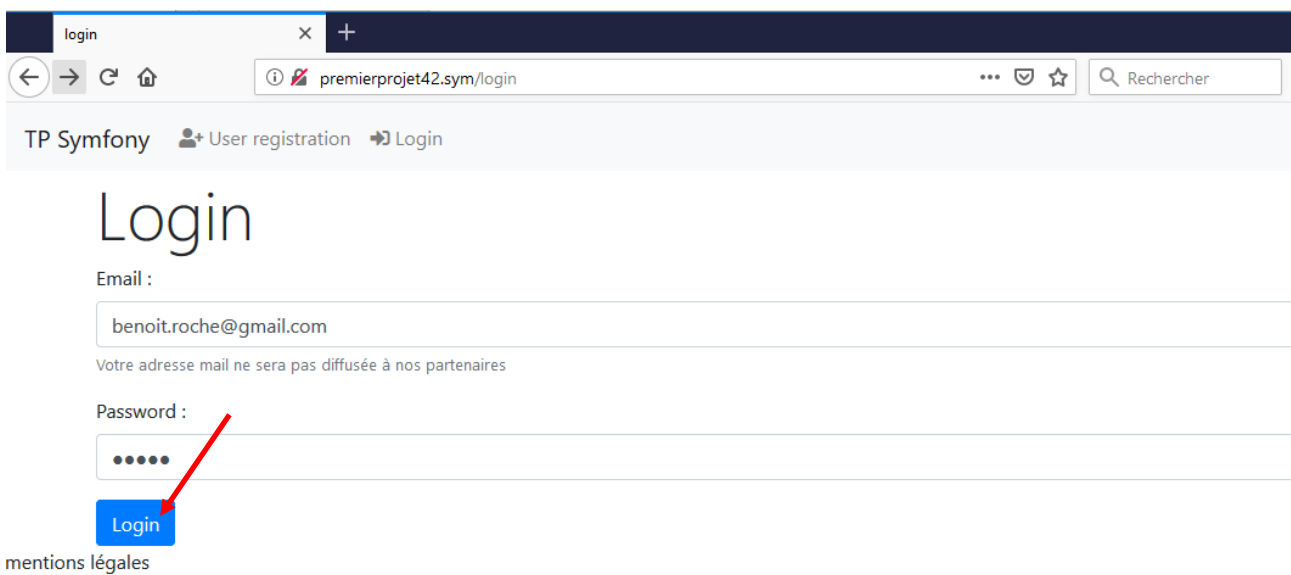
{% block body %}
    <header>
        <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <a class="navbar-brand" href="#">TF Symfony</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav mr-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{{ path('user_registration') }}"><i class="fas fa-user-plus"></i> User registration</a>
                    </li>
                    {% if is_granted('ROLE_USER') %}
                        <li class="nav-item dropdown">
                            <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                                <i class="fas fa-user"></i> {{ app.user.prenom }} {{ app.user.nom }}
                            </a>
                            <div class="dropdown-menu" aria-labelledby="navbarDropdown">
                                <a class="dropdown-item" href="{{ path('user') }}">Account</a>
                                <div class="dropdown-divider"></div>
                                <a class="dropdown-item" href="{{ path('logout') }}"><i class="fas fa-sign-out-alt"></i> Disconnect</a>
                            </div>
                        </li>
                    {% else %}
                        <li class="nav-item">
                            <a class="nav-link" href="{{ path('login') }}"><i class="fas fa-sign-in-alt"></i> Login</a>
                        </li>
                    {% endif %}
                </ul>
            </div>
        </nav>
    </header>
{% endblock %}
```

Ici nous vérifions que l'utilisateur possède bien les droits **ROLE_USER** (rôle qu'on lui a passé dans la méthode **getRoles()** de l'interface **UserInterface**). S'il possède bien les droits, on affiche le contenu souhaité.

Ensuite, retour dans le contrôleur **LoginController** pour écrire la méthode **showUser**. C'est la méthode qui sera appelée après l'authentification de l'utilisateur. (voir la précédente modification du **security.yml**).

```
/**
 * @Route("/user", name="user")
 */
public function showUser(Security $security) {
    $user = $security->getUser();
    return $this->render('login/user.html.twig', array(
        'user' => $user, 'title' => "login"
    ));
}
```

Maintenant il ne nous reste plus qu'à essayer de nous connecter !



login

premierprojet42.sym/login

TP Symfony

User registration Login

Login

Email :

benoit.roche@gmail.com

Votre adresse mail ne sera pas diffusée à nos partenaires

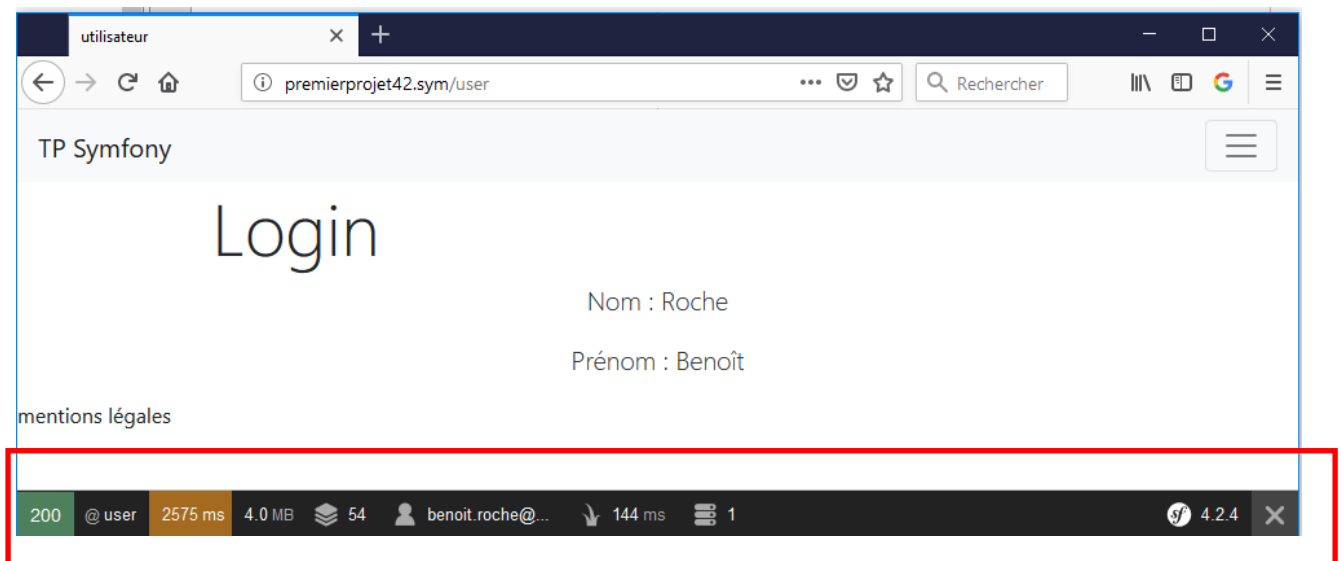
Password :

.....

Login

mentions légales

Et voilà le résultat :



utilisateur

premierprojet42.sym/user

TP Symfony

Login

Nom : Roche

Prénom : Benoît

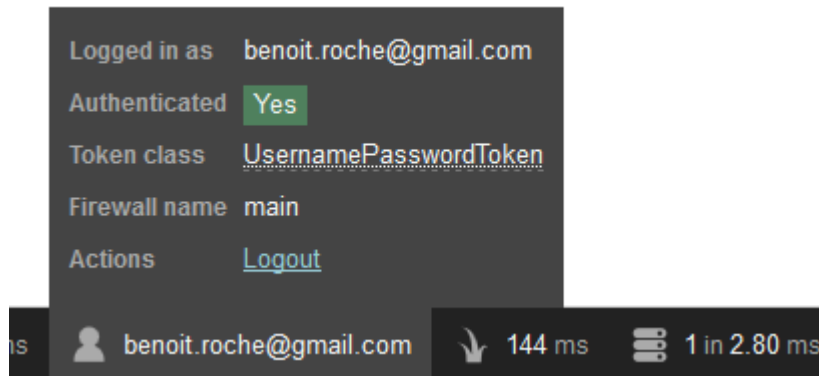
mentions légales

200 @ user 2575 ms 4.0 MB 54 benoit.roche@... 144 ms 1

sf 4.2.4

Ça fonctionne !

Jusqu'ici, nous ne nous étions pas intéressé à la barre de débogage de Symfony. Je vous invite à la regarder d'un peu plus près !



Ceci nous confirme bien que nous sommes bien connectés en tant qu'utilisateur. Mettez un point d'arrêt dans la méthode showUser de la classe LoginController :

```

    ));
}

/**
 * @Route("/user", name="show_user")
 */
public function showUser()
{
    $user = $security->getUser();
}

```

```

App\Entity\User object {
  id => (int) 1
  nom => (string) Roche
  prenom => (string) Benoît
  mail => (string) benoit.roche@gmail.com
  password => (string) $2y$13$8SqWloBLDWZzOd3CMxdC2u9gkrRxpxe9KAnQzw2l/ptHyc5XLdPTG
  plainPassword => null
}

```

Vous voyez que le plainPassword est vide et que le password est bien chiffré.

4. Déconnexion

Après la connexion, vous allez programmer la déconnexion. Ce sera plus facile.... Vous aurez à modifier le fichier security.yml :

```

check_path: login
default target path: user

logout:
  path: logout
  target: login

```

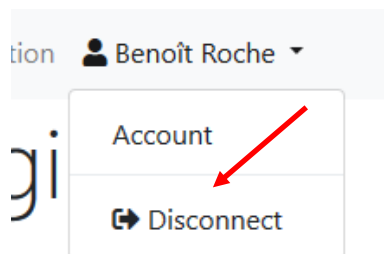
Vous indiquez ici la routes pour le formulaire de déconnexion puis la route après la déconnexion... Il faut donc créer cette route et la méthode action qui va avec. Vous rajouterez donc la méthode action logout dans la classe LoginController :

```
/**
 * @Route("/logout", name="logout")
 */
public function logout() {
    return $this->render(array(
        'login/index.html.twig',
        'title' => "deconnexion")
    );
}
```

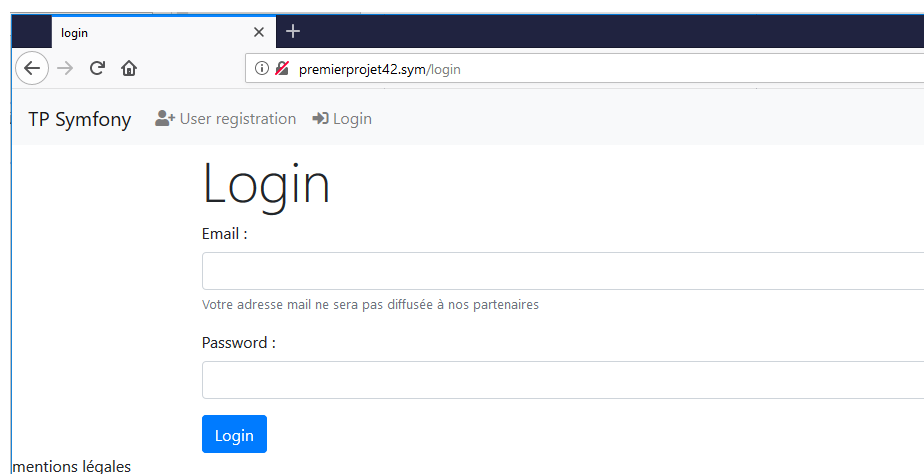
Et vérifiez que la ligne suivante est dans votre barre de navigation :

```
<div class="dropdown-divider"></div>
<a class="dropdown-item" href="{{ path('logout') }}"><i class="fas fa-sign-out-alt"></i> Disconnect</a>
```

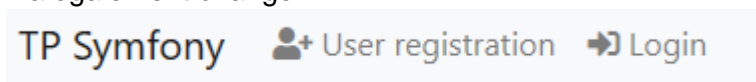
Une fois connecté, il ne vous reste plus qu'à tester la déconnexion :



Vous reviendrez sur ce formulaire :



Et vous constaterez dans la barre de débogage que la déconnexion a bien eu lieu. La barre de navigation a également changé :



Partie 13 : EXERCICES BONUS

1. Exercice 1

Vous avez vu que nous n'avons rien pour vérifier qu'un utilisateur a bien été enregistré mis à part jeter un œil dans la base de données. Essayez de mettre en place un moyen d'afficher un message lorsqu'un utilisateur a réussi à s'enregistrer.

2. Exercice 2

Comme il s'agit d'un système de login par adresse mail, essayez de mettre en place un moyen permettant de vérifier que l'adresse mail saisie dans le formulaire d'enregistrement n'est pas déjà utilisée pour un compte existant. Afficher un message si l'adresse mail est déjà utilisée.

Partie 14 : SOLUTIONS

1. Exercice 1:

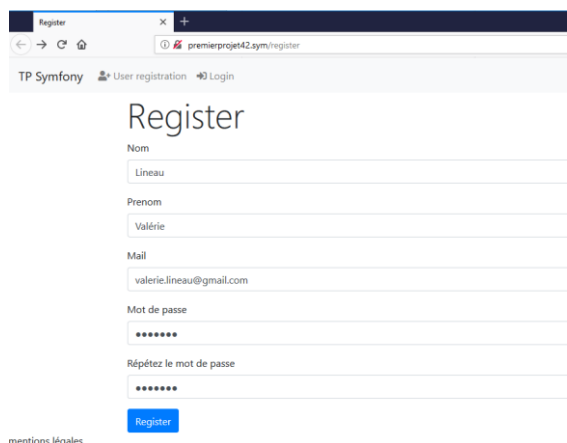
Vous aurez à modifier la méthode Register de la classe RegistrationController :

```
return $this->render(
    'registration/register.html.twig',
    array(
        'form' => $form->createView(),
        'text_alert'=>'Vous êtes authentifié',
        'class_alert'=>'alert-success'
    )
)
```

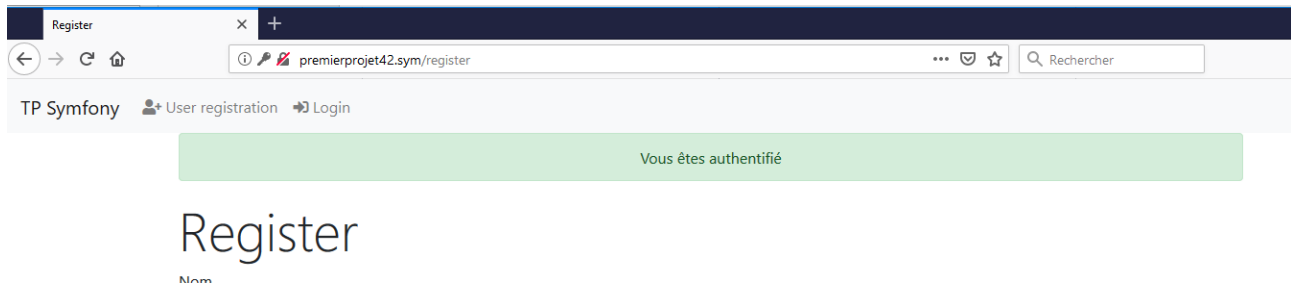
Puis la vue register.html.twig :

```
{{ parent() }}
<div class="container">
    {% if text_alert is defined and class_alert is defined %}
        <div class="alert {{ class_alert }}" text-center">{{ text_alert }}</div>
    {% endif %}
    <section>
```

Testez :



Et vous obtiendrez ceci :



2. Exercice 2

L'idée ici est d'aller vérifier, avant d'enregistrer dans la base de données que l'adresse mail, qui sert d'identifiant de connexion et don qui doit être unique, n'existe pas déjà en vase.

Vous créerez donc, dans la classe RegistrationController la méthode privée findOneBymail qui retournera soit rien (null) ou alors un enregistrement. Dans ce cas vous devrez générer un message d'erreur dans le formulaire d'enregistrement.

La méthode findOneByMail :

```
private function findOneByMail(string $mail) {
    $repository = $this->getDoctrine()->getManager()->getRepository(User::class);
    $result = $repository->findOneBy(array(
        'mail' => $mail
    ));
    return $result;
}
```

Le test dans le contrôleur :

```
$form->handleRequest($request);
if ($form->isSubmitted() && $form->isValid()) {
    if($this->findOneByMail($user->getEmail())=== null){
        // le mail n'existe pas dans la BD, on peut donc enregistrer la demande
    }
}
```

Et le else correspondant :

```
//
}
else {
    // le mail existe déjà, on ne peut donc enregistrer la demande
    return $this->render(
        'registration/register.html.twig',
        array(
            'form' => $form->createView(),
            'text_alert' => 'Cette adresse mail existe déjà, inscription impossible',
            'class_alert' => 'alert-danger'
        )
    );
}
```

Il ne vous reste plus qu'à tester :

Register

TP Symfony User registration Login

Register

Nom
Lineau

Prenom
Benoît

Mail
valerie.lineau@gmail.com

Mot de passe
.....

Répétez le mot de passe
.....

Register

mentions légales

Le résultat :

Register

TP Symfony User registration Login

Cette adresse mail existe déjà, inscription impossible

Register

Nom
Lineau

3. Autre exercice

Essayez de créer un utilisateur et saisissez 2 mots de passe différents. Que se passe-t-il ? Vous pourrez améliorer le rendu en explorant le chapitre validation de formulaire.

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

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

