

Module : Symfony

TP1 : Initiations Symfony

Objectifs :

- Rappeler les notions de base du framework symfony.
- Préparer l'environnement de travail.
- Démarrer le projet EPIJob.
- Manipulations de base d'un projet symfony

Partie1 : Définitions

Symfony est un ensemble de composants PHP, c'est un framework MVC libre écrit en PHP développé par l'agence web française SensioLabs dont l'apparition date depuis 2005.

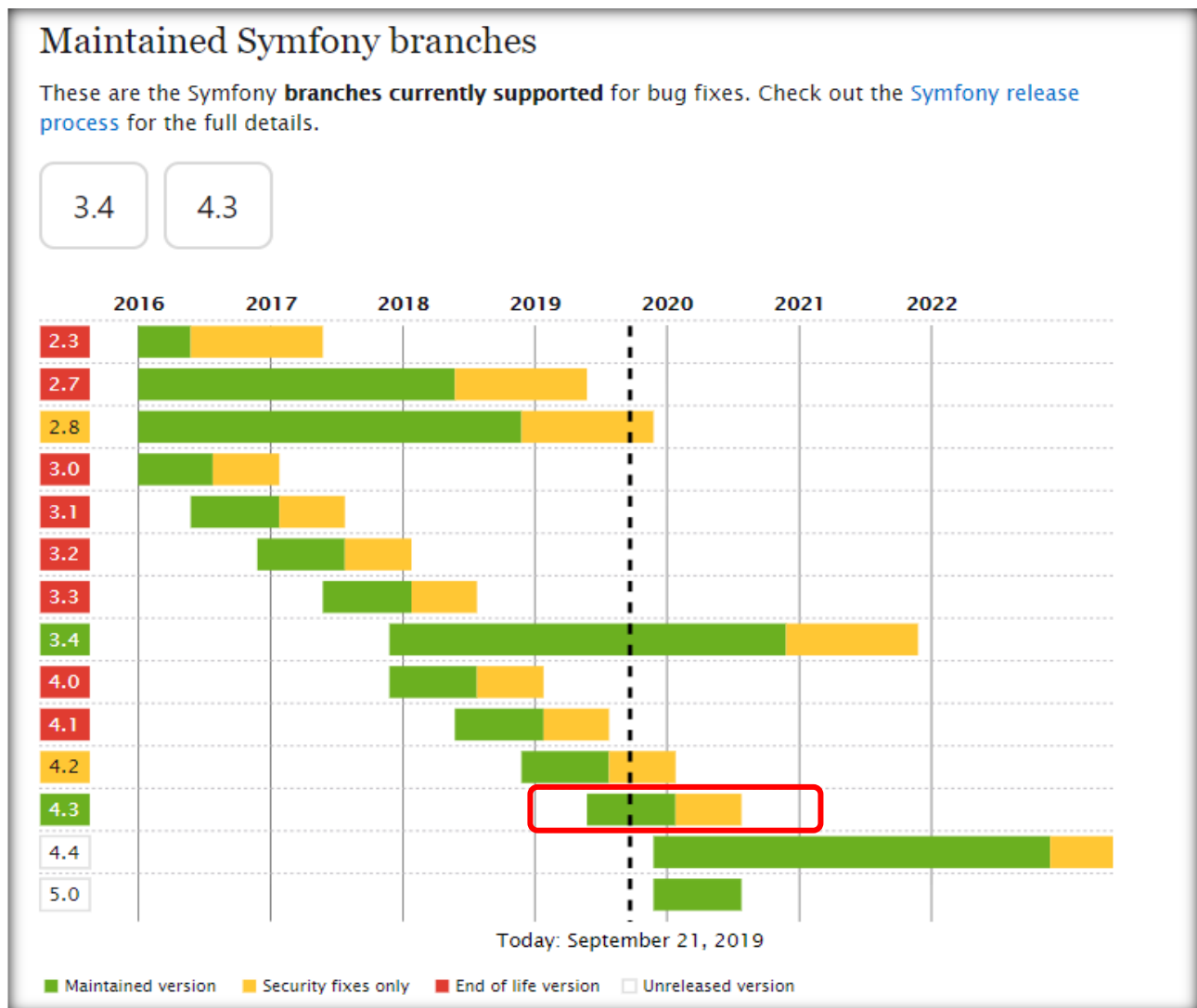


C'est l'un parmi les frameworks web les plus utilisés dans le monde, et qui est doté d'une grande communauté.

Il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d'accélérer le développement d'une application web.

La version 4 du framework Symfony a été mise à disposition le 30 novembre 2017.

Cette nouvelle version représente une révolution dans l'historique du framework et a apporté beaucoup de changements par rapports aux anciennes versions.

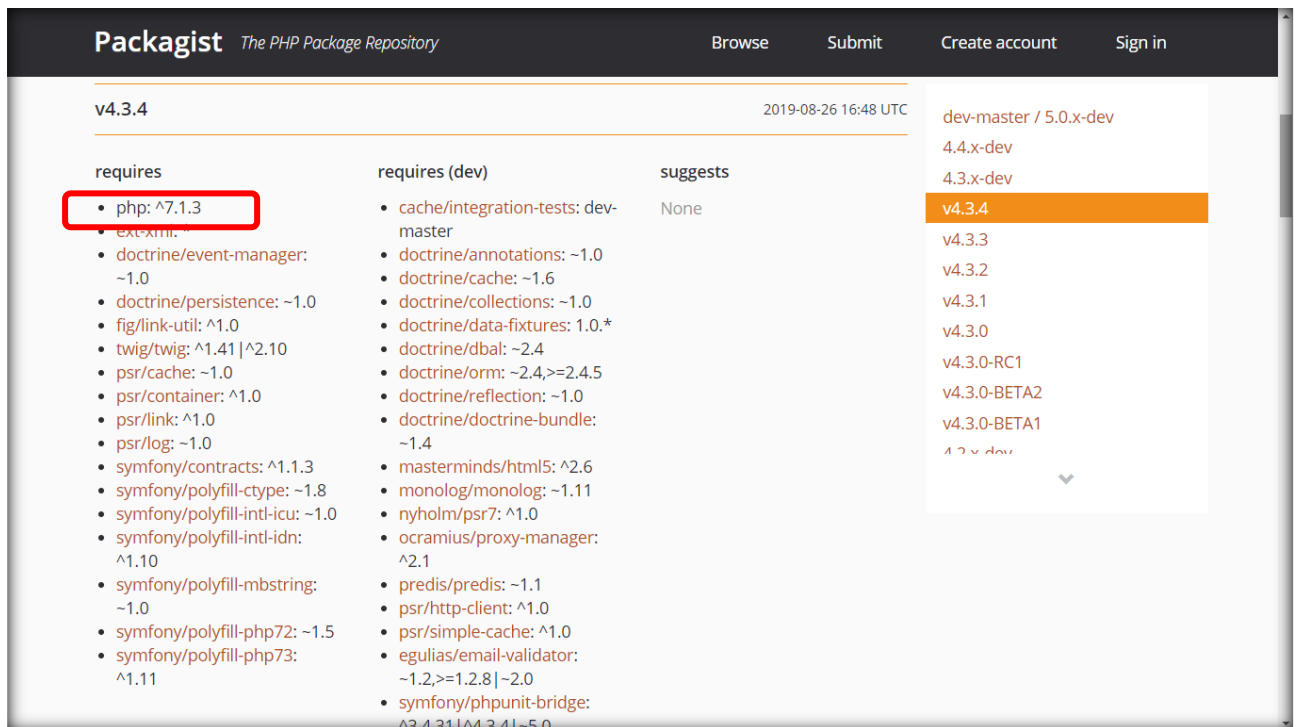


Partie 2 : Installer un projet Symfony4

➤ Etapes de création d'un projet Symfony4

✓ Étape 1 : vérification de l'environnement de mise au point PHP

- Installer Wamp et vérifiez que l'interpréteur PHP en ligne de commande est présent, et que la version de PHP installée est supérieure ou égale à 7.1.3 :
`php -v`

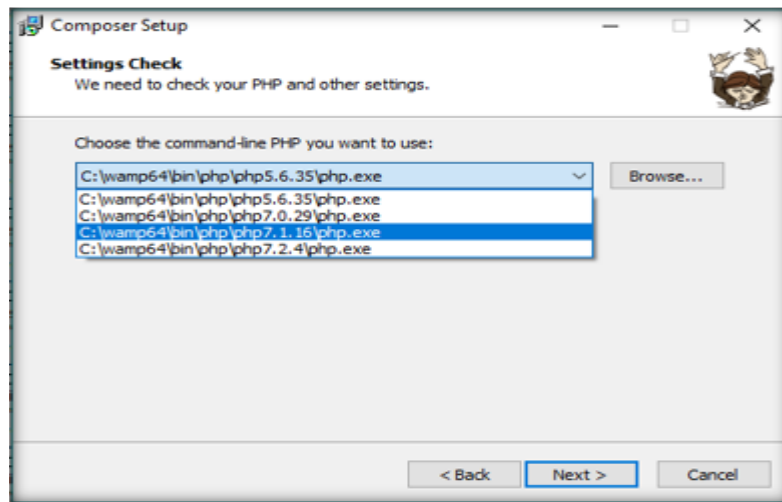


The screenshot shows the Packagist website for the package 'symfony/symfony'. The version 'v4.3.4' is selected. The 'requires' section is highlighted with a red box, showing the requirement 'php: ^7.1.3'. Other requirements include 'doctrine/event-manager: ~1.0', 'doctrine/persistence: ~1.0', 'fig/link-util: ^1.0', 'twig/twig: ^1.41|^2.10', 'psr/cache: ~1.0', 'psr/container: ^1.0', 'psr/link: ^1.0', 'psr/log: ~1.0', 'symfony/contracts: ^1.1.3', 'symfony/polyfill-ctype: ~1.8', 'symfony/polyfill-intl-icu: ~1.0', 'symfony/polyfill-intl-idn: ^1.10', 'symfony/polyfill-mbstring: ~1.0', 'symfony/polyfill-php72: ~1.5', and 'symfony/polyfill-php73: ^1.11'.

- Installer Composer et vérifiez que la commande composer est présente sur votre installation de PHP :
`composer -v`

Composer est le gestionnaire de dépendances utilisé par les applications PHP modernes. Il permet de déclarer les bibliothèques dont dépend le projet et les gérer (installation/MAJ).

- Accéder à <https://getcomposer.org/download/>
- Télécharger et exécuter Composer-Setup.exe qui permet d'installer la dernière version du composeur à chaque exécution.
- Il faut faire attention à la version du php utilisée lors de l'installation !!!



✓ Étape 2 : Lancement de la création du projet Symfony

Deux façons de le faire :

- Via l'installateur Symfony :

Il faut télécharger le fichier Symfony.phar dans le répertoire www depuis le lien suivant : <https://symfony.com/download>

```
symfony new --full EPIJOB
```

- Via Composer

Pour créer une nouvelle application web Symfony 4, il faut lancer la commande suivante :

```
composer create-project symfony/website-skeleton:"^4.4" EPIJOB
```

Composer va faire le nécessaire pour créer un dossier du projet et télécharger les dépendances associées.

```
Invite de commandes - composer create-project symfony/website-skeleton EPIJOB

C:\wamp64\www>composer create-project symfony/website-skeleton EPIJOB
Installing symfony/website-skeleton (v4.3.99)
- Installing symfony/website-skeleton (v4.3.99): Loading from cache
Created project in EPIJOB
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing symfony/flex (v1.4.6): Loading from cache
Symfony operations: 1 recipe (caa52e76ef5ac3da3de784d38ce0321d)
- Configuring symfony/flex (>=1.0): From github.com/symfony/recipes:master
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.3.*"
Package operations: 103 installs, 0 updates, 0 removals
- Installing ocrapius/package-versions (1.4.0): Loading from cache
- Installing doctrine/lexer (1.0.2): Loading from cache
- Installing doctrine/annotations (v1.7.0): Loading from cache
- Installing doctrine/reflection (v1.0.0): Loading from cache
- Installing doctrine/event-manager (v1.0.0): Loading from cache
- Installing doctrine/collections (v1.6.2): Loading from cache
- Installing doctrine/cache (v1.8.0): Loading from cache
- Installing doctrine/persistence (1.1.1): Loading from cache
- Installing symfony/polyfill-php73 (v1.12.0): Loading from cache
- Installing symfony/polyfill-mbstring (v1.12.0): Loading from cache
- Installing symfony/polyfill-php72 (v1.12.0): Loading from cache
- Installing symfony/polyfill-intl-idn (v1.12.0): Loading from cache
- Installing symfony/mime (v4.3.4): Loading from cache
- Installing symfony/http-foundation (v4.3.4): Loading from cache
- Installing symfony/event-dispatcher-contracts (v1.1.5): Loading from cache
```

Composer va alors créer un dossier pour notre projet en se basant sur le modèle de projet Symfony et télécharger les dépendances associées.

```
Invite de commandes

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. Go to the project directory
  2. Create your code repository with the git init command
  3. Download the Symfony CLI at https://symfony.com/download to install a development web server

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

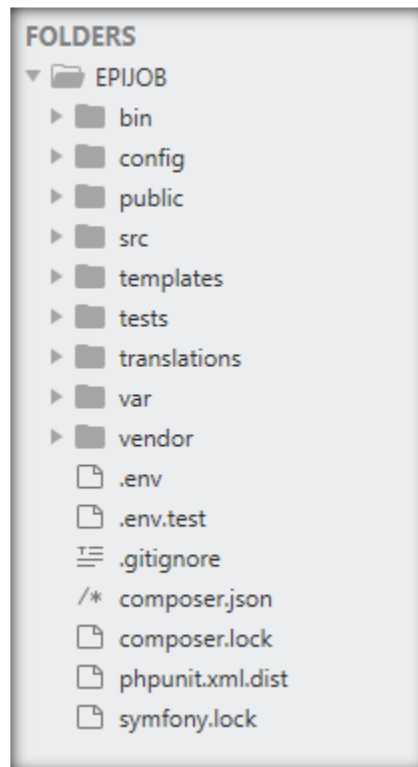
Database Configuration

* Modify your DATABASE_URL config in .env

* Configure the driver (mysql) and
  server_version (5.7) in config/packages/doctrine.yaml

How to test?

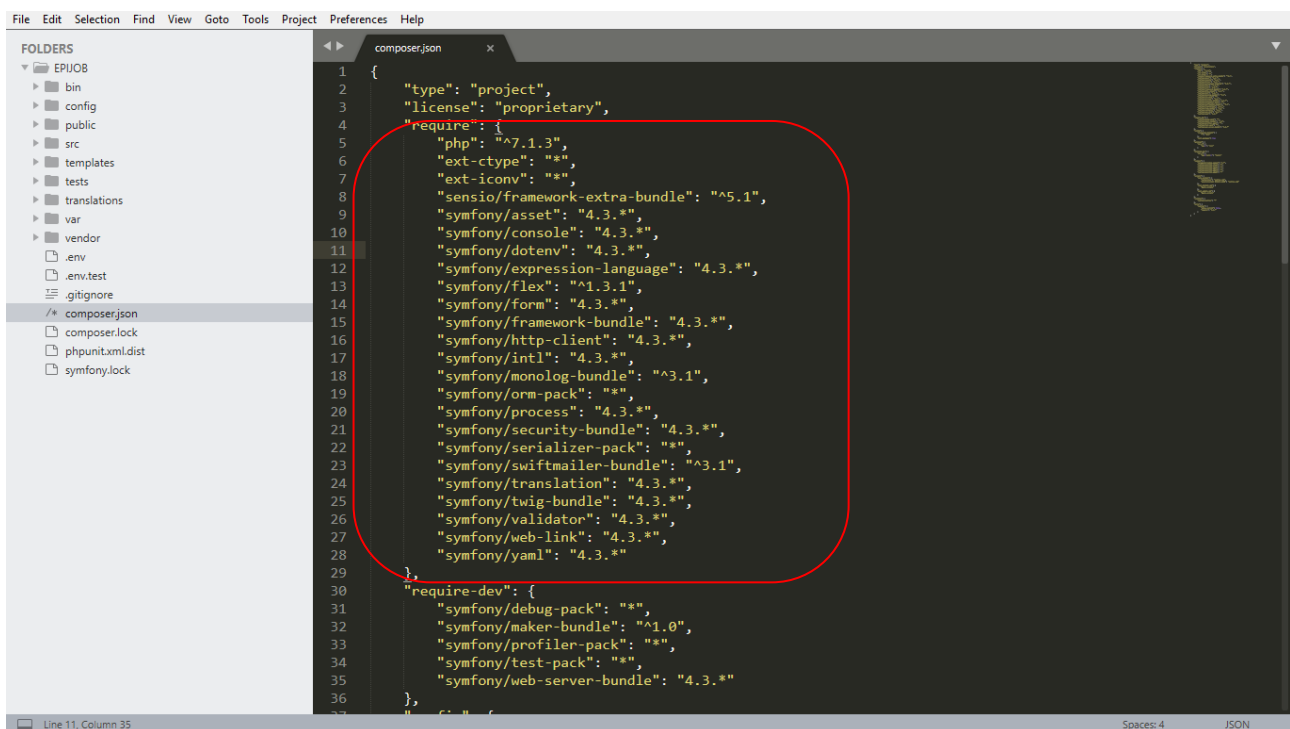
* Write test cases in the tests/ folder
* Run php bin/phpunit
```



Analysons la structure de notre projet vide :

- **public** : Le répertoire contenant les fichiers desservis par les serveurs http(on place tous les fichiers accessibles au public ici)
- **config** : dossier les fichiers de configuration
- **bin** : contient l'utilitaire console.
- **src** : Contiendra les fichiers sources PHP contenant toute la logique de notre projet.
- **vendor** : Le dossier utilisé par Composer et contenant les dépendances (les bibliothèques PHP du *framework* Symfony qui seront utilisées et qui décrites dans le fichier composer.json) nécessaires au fonctionnement de notre projet.
- **Var** : contient log et cache
- **env** : le fichier contenant la configuration de l'environnement d'exécution de notre code.

- **env.dist** : est un fichier versionné qui sert de modèle pour que les développeurs qui vont être amenés à travailler sur le projet puissent connaître les informations à renseigner.
- **composer.json** : descriptif du projet Composer de l'application Symfony. Il référence notamment les bibliothèques PHP utilisées.



```
1 {
2     "type": "project",
3     "license": "proprietary",
4     "require": {
5         "php": "^7.1.3",
6         "ext-ctype": "*",
7         "ext-iconv": "*",
8         "sensio/framework-extra-bundle": "^5.1",
9         "symfony/asset": "4.3.*",
10        "symfony/console": "4.3.*",
11        "symfony/dotenv": "4.3.*",
12        "symfony/expression-language": "4.3.*",
13        "symfony/flex": "^1.3.1",
14        "symfony/form": "4.3.*",
15        "symfony/framework-bundle": "4.3.*",
16        "symfony/http-client": "4.3.*",
17        "symfony/intl": "4.3.*",
18        "symfony/monolog-bundle": "^3.1",
19        "symfony/orm-pack": "*",
20        "symfony/process": "4.3.*",
21        "symfony/security-bundle": "4.3.*",
22        "symfony/serializer-pack": "*",
23        "symfony/swiftmailer-bundle": "^3.1",
24        "symfony/translation": "4.3.*",
25        "symfony/twig-bundle": "4.3.*",
26        "symfony/validator": "4.3.*",
27        "symfony/web-link": "4.3.*",
28        "symfony/yaml": "4.3.*"
29    },
30    "require-dev": {
31        "symfony/debug-pack": "*",
32        "symfony/maker-bundle": "^1.0",
33        "symfony/profiler-pack": "*",
34        "symfony/test-pack": "*",
35        "symfony/web-server-bundle": "4.3.*"
36    },
37 }
```

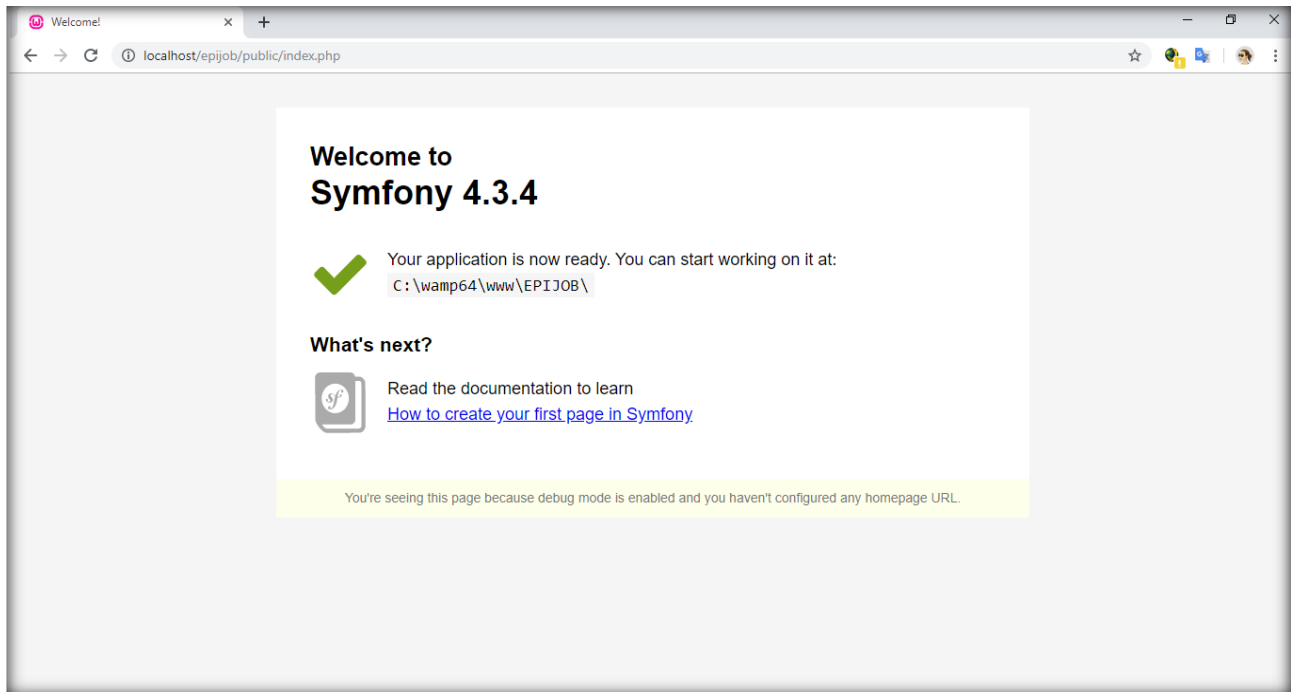
Pour vérifier la version de Symfony utilisée dans le projet

bin/console -V

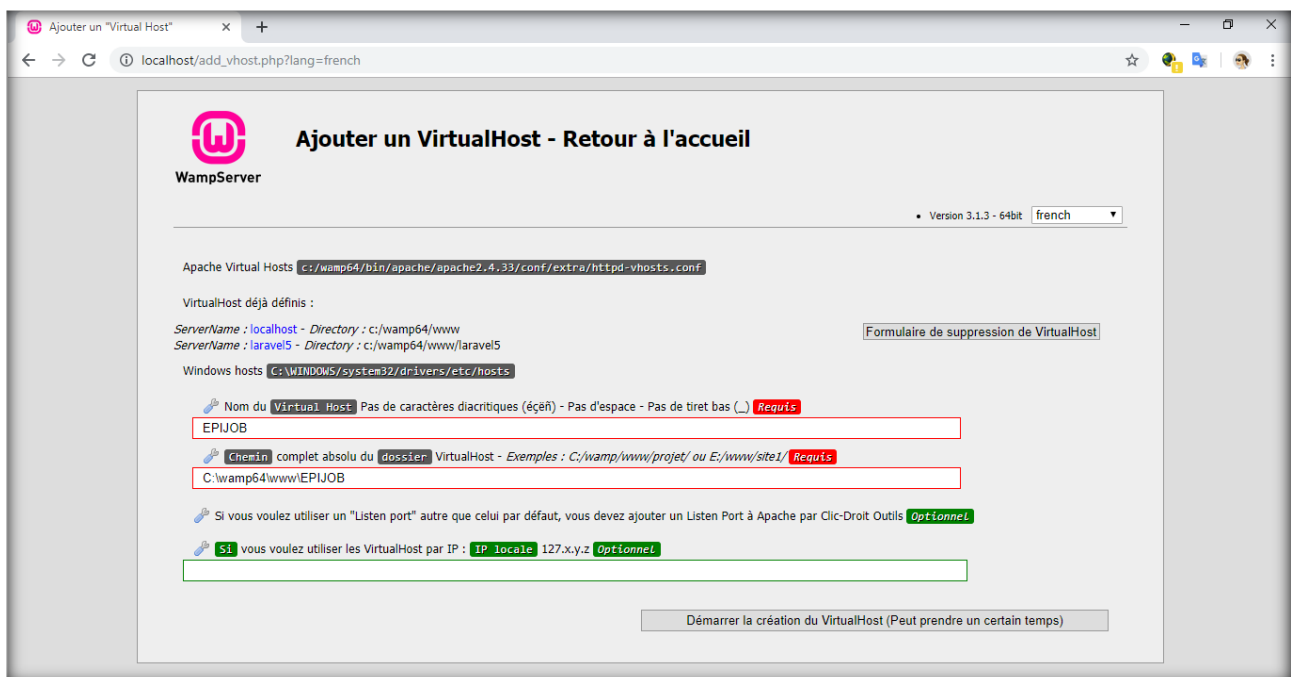
```
C:\wamp64\www\EPIJOB>php bin/console -V
Symfony 4.3.4 (env: dev, debug: true)
```

Accéder à notre site via l'url

<http://localhost/epijob/public/index.php>



Pour un accès plus simple, il faut rajouter un virtualHost pour le nouveau projet :



Partie3 : Code Bundless

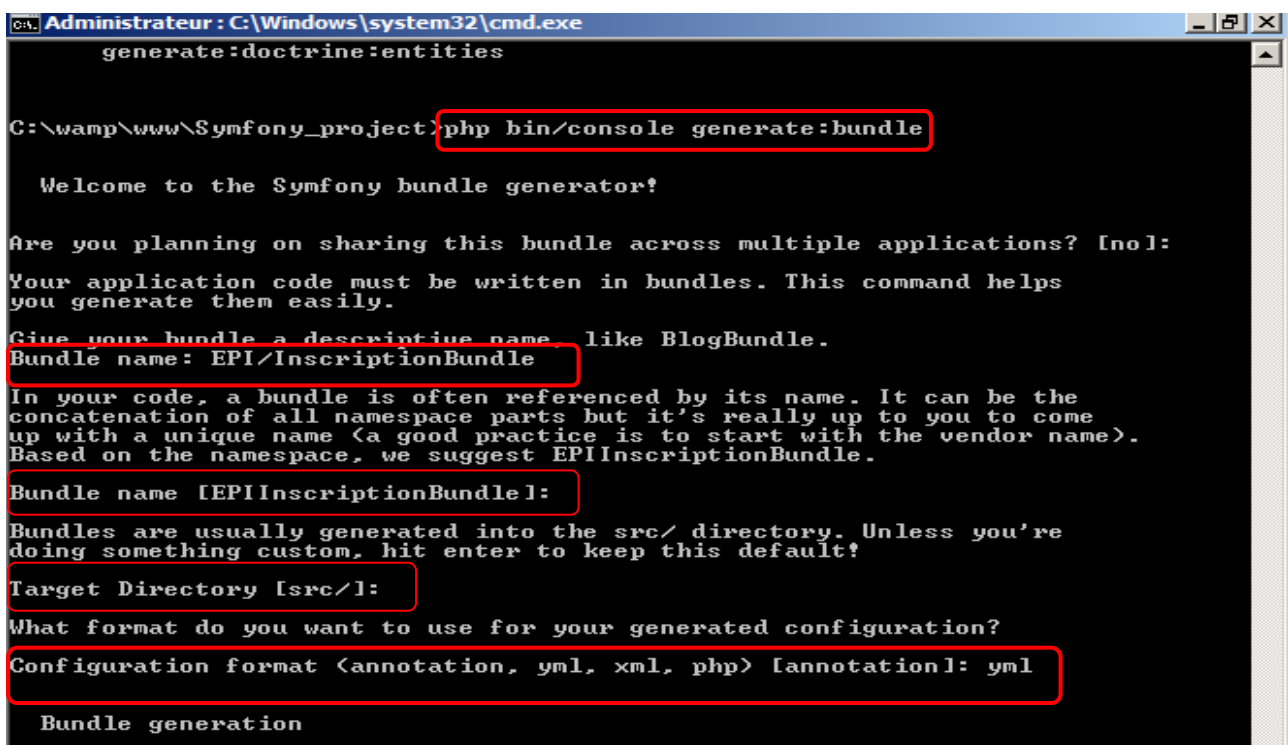
La première étape à suivre lors du démarrage d'un projet Symfony 3 était de créer un bundle en utilisant la commande suivante :

```
php bin/console generate:bundle
```

Un bundle peut être considéré comme un "plugin".

Ils permettent de regrouper du code qui est regroupé par fonctionnalité et redistribuable dans n'importe quel projet Symfony.

Dans Symfony 2 et 3, les applications étaient elles-mêmes des bundles.



```
Administrateur : C:\Windows\system32\cmd.exe
generate:doctrine:entities

C:\wamp\www\Symfony_project>php bin/console generate:bundle

Welcome to the Symfony bundle generator!

Are you planning on sharing this bundle across multiple applications? [no]:
Your application code must be written in bundles. This command helps
you generate them easily.

Give your bundle a descriptive name, like BlogBundle.
Bundle name: EPI/InscriptionBundle

In your code, a bundle is often referenced by its name. It can be the
concatenation of all namespace parts but it's really up to you to come
up with a unique name (a good practice is to start with the vendor name).
Based on the namespace, we suggest EPIInscriptionBundle.
Bundle name [EPIInscriptionBundle]:

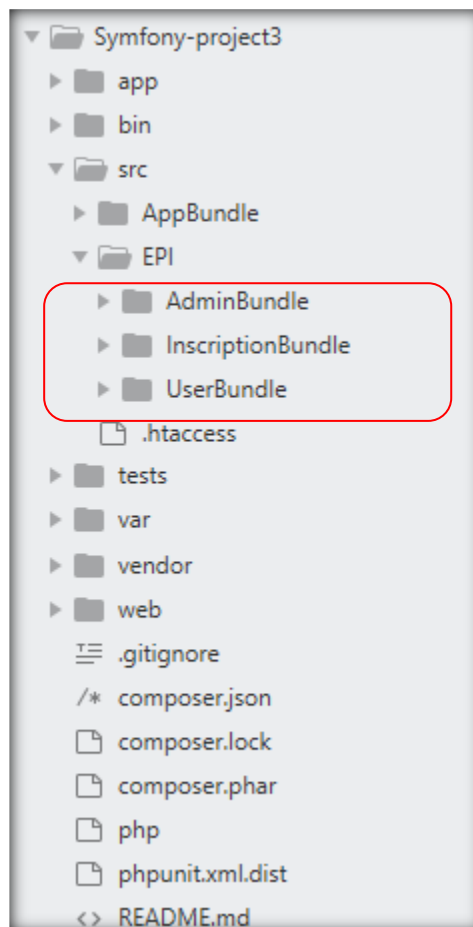
Bundles are usually generated into the src/ directory. Unless you're
doing something custom, hit enter to keep this default!
Target Directory [src/]:

What format do you want to use for your generated configuration?
Configuration format <annotation, yaml, xml, php> [annotation]: yaml

Bundle generation
```

Coder un bundle dans son application imposait d'écrire du code très spécifique pour l'intégration avec Symfony lui-même.

Ce code n'avait que peu de valeur ajoutée, mais il était indispensable, induisant un dette technique inutile.



A partir de sa 4^{ème} version, Symfony recommande et génère des applications orientées bundle-less.

Bundle-less ne signifie pas que l'application ne consomme plus de bundles.

La nuance est que désormais les bundles ne sont plus que pour du code tiers.

Le but avec Symfony 4 c'est de garder une structure assez légère.

Vous êtes libres d'organiser votre code dans le dossier 'src'.

Partie4 : Symfony Flex

Avant Symfony 4, l'installation d'un bundle tiers était fastidieuse, on devait passer par la liste des étapes suivantes :

- `composer require bundle`
- Ajouter une ligne dans `app/AppKernel.php`
- Créer la configuration dans `app/config/config.yml`
- Importer le routing dans `app/config/routing.yml`

Symfony Flex est la nouvelle façon d'installer et de gérer les applications Symfony.

En effet, c'est une surcouche à Composer et qui permet entre autres de configurer automatiquement les dépendances que vous installez dans votre application.

Symfony Flex peut marcher sur un projet Symfony 3.3 mais devient obligatoire sur un projet Symfony 4.

En d'autres termes, Symfony Flex est un plugin Composer qui modifie le comportement des commandes `require`, `update` et `remove`.

Lorsque Symfony Flex est installé dans l'application et que vous exécutez `composer require`, l'application envoie une requête au serveur Symfony Flex avant d'essayer d'installer le paquet avec Composer :

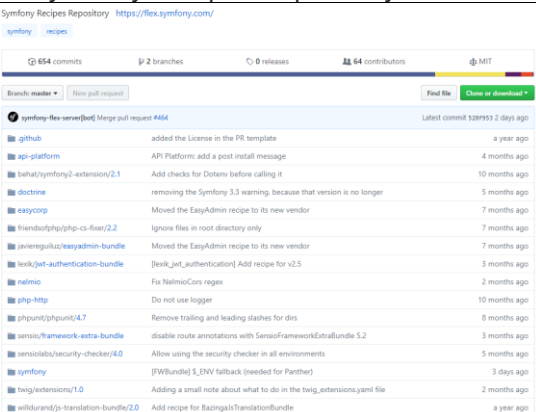
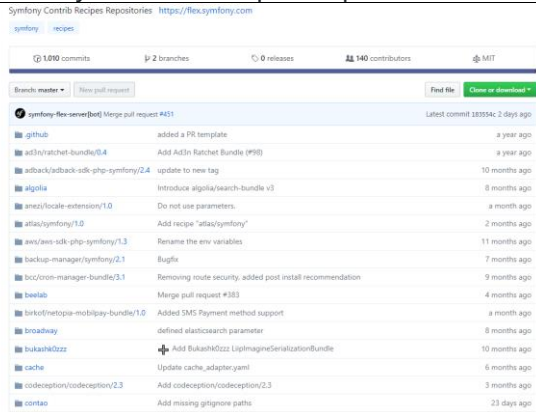
- S'il n'y a aucune information sur ce paquet, le serveur Flex ne renvoie rien et l'installation du paquet suit la procédure habituelle basée sur Composer.
- S'il existe des informations spéciales sur ce package, Flex le renvoie dans un fichier appelé « recette » et l'application l'utilise pour décider du package à installer et des tâches automatisées à exécuter après l'installation.

Partie5 : Recettes

Symfony 4 introduit la notion des recettes (recettes), une suite de ligne de commandes à exécuter lors de l'installation des bundles.

Deux dépôts regroupent les recettes pour installer des paquets tiers :

1. **Main recipe repository** : C'est une liste organisée de recettes pour des paquets de haute qualité et maintenus. Symfony ne regarde que dans ce référentiel par défaut.
<https://github.com/symfony/recipes>
2. **Contrib recipe repository** : C'est une liste de recettes créées par la communauté. Elles sont toutes garanties pour fonctionner, mais leurs paquets associés pourraient ne pas être maintenus. Symfony vous demandera la permission avant d'installer l'une de ces recettes <https://github.com/symfony/recipes-contrib>

Symfony Recipes Repository	Symfony Contrib Recipes Repositories
 <p>The screenshot shows the GitHub repository for Symfony Recipes. It has 654 commits, 2 branches, 0 releases, and 64 contributors. The MIT license is listed. The repository contains a list of recipes, each with a description and the time since the latest commit. Examples include 'github' (added the License in the PR template), 'api-platform' (API Platform: add a post install message), and 'doctrine' (removed the Symfony 3.3 warning).</p>	 <p>The screenshot shows the GitHub repository for Symfony Contrib Recipes. It has 1,010 commits, 2 branches, 0 releases, and 140 contributors. The MIT license is listed. The repository contains a list of recipes, each with a description and the time since the latest commit. Examples include 'symfony-flex-server' (Merge pull request #451), 'github' (added a PR template), and 'api-platform' (Add Adin Ratchet Bundle (PSR)).</p>

En effet cela décrit comment un package doit être configuré.

Les recettes Symfony permettent l'automatisation de la configuration des packages Composer via le plugin Symfony **FlexComposer**.

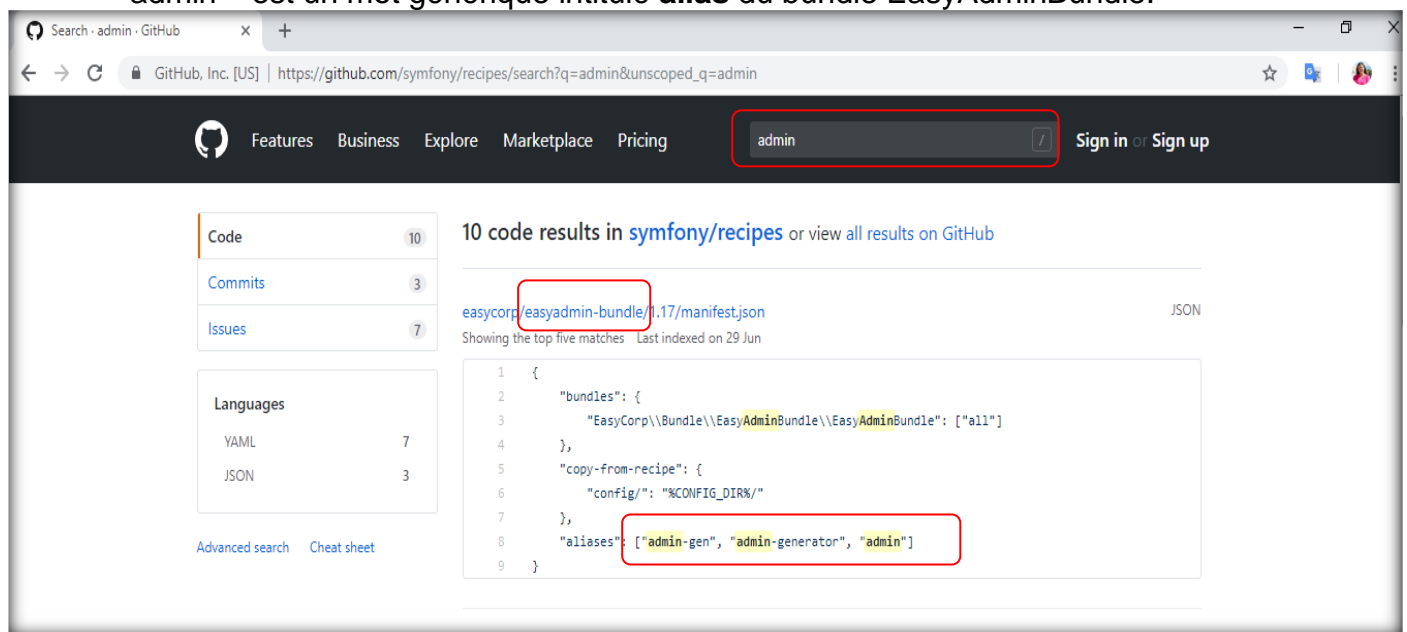
L'installation d'un bundle devient plus simple.

Exemple : Installation du **EasyAdminBundle** :

```
composer require admin
```

Outre l'installation du faisceau du générateur d'administration, il installe également toutes ses dépendances transitives et les configure automatiquement tous : TwigBundle, SecurityBundle, DoctrineBundle et FrameworkExtraBundle(annotations).

« admin » est un mot générique intitulé **alias** du bundle EasyAdminBundle.



Exemples d'alias:

- template : pour Twig.
- mailer : pour Swiftmailer.
- profiler : pour le profileur.

Partie6 : MakerBundle de Symfony

Symfony **MakerBundle** a été développé dans le but d'aider les développeurs à créer des commandes vides, des contrôleurs, des classes de formulaire ...

Ce bundle est une alternative à SensioGeneratorBundle pour les applications Symfony modernes.

Une simple commande magique permet de l'installer :

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

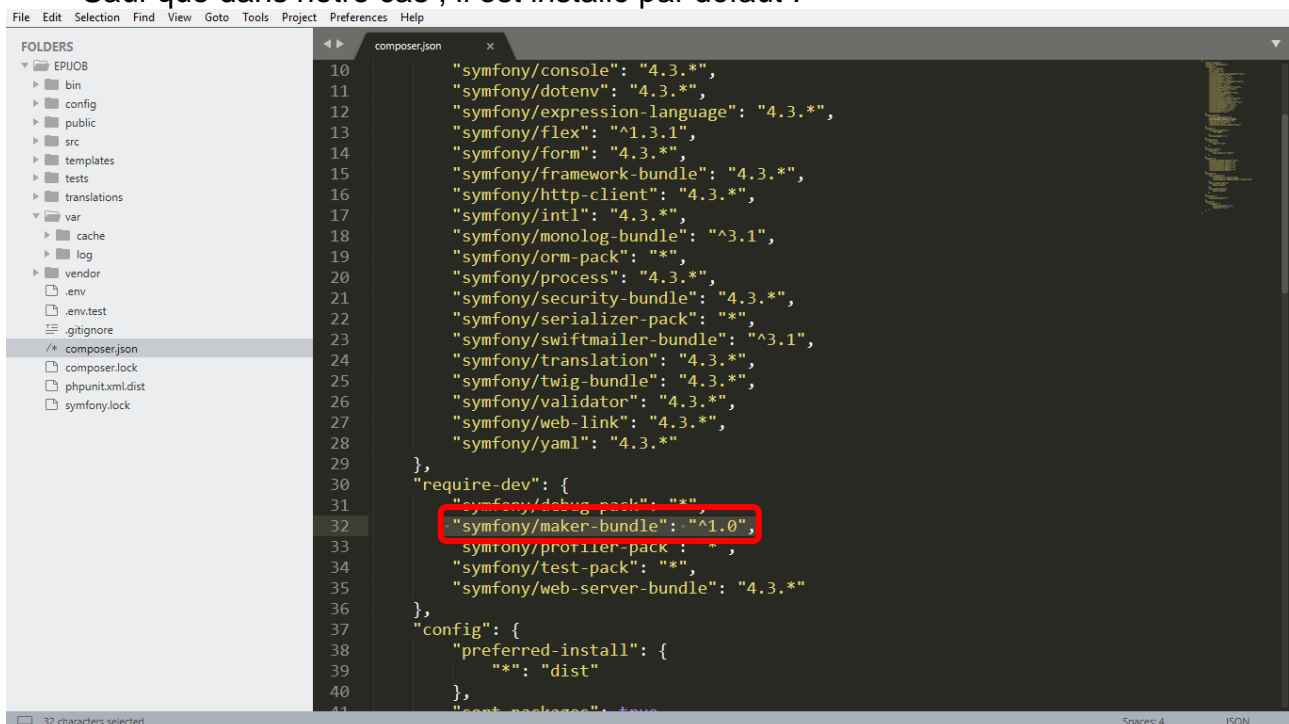
Équivalente à

```
composer require maker
```

```
C:\wamp64\www\Symfony-project4>composer require maker
Using version ^1.7 for symfony/maker-bundle
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.1.*"
Nothing to install or update
Writing lock file
Generating autoload files
ocramius/package-versions: Generating version class...
ocramius/package-versions: ...done generating version class
Executing script cache:clear [OK]
Executing script assets:install public [OK]
Executing script requirements-checker [OK]

C:\wamp64\www\Symfony-project4>
```

Sauf que dans notre cas , il est installé par défaut :



Lister les commandes du symfony MakerBundle :

```
php bin/console list make
```

```
C:\wamp64\www\Symfony-project4>php bin/console list make
Symfony 4.1.4 (kernel: src, env: dev, debug: true)

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
  -e, --env=ENV          The Environment name. [default: "dev"]
      --no-debug         Switches off debug mode.
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands for the "make" namespace:
  make:auth              Creates an empty Guard authenticator
  make:command            Creates a new console command class
  make:controller         Creates a new controller class
  make:crud               Creates CRUD for Doctrine entity class
  make:entity             Creates or updates a Doctrine entity class, and optionally an API Platform resource
  make:fixtures            Creates a new class to load Doctrine fixtures
  make:form               Creates a new form class
  make:functional-test    Creates a new functional test class
  make:migration           Creates a new migration based on database changes
  make:serializer:encoder Creates a new serializer encoder class
  make:subscriber         Creates a new event subscriber class
  make:twig-extension     Creates a new Twig extension class
  make:unit-test          Creates a new unit test class
  make:user               Creates a new security user class
  make:validator           Creates a new validator and constraint class
  make:voter              Creates a new security voter class

C:\wamp64\www\Symfony-project4>
```

Partie7 : Préparer l'environnement de travail

➤ Installer Sublime text

✓ Installer Package contrôle

▪ méthode simple :

- Dans la console du sublime (View > Show Console) copier un bout de code depuis ce site : <https://packagecontrol.io/installation>

▪ méthode manuelle :

- Cliquer sur Preferences > Browse Packages...
- Accéder au dossier parent Sublime Text 3 puis rentrer dans le dossier Installed Packages/
- Télécharger le [Package Control.sublime-package](https://packagecontrol.io/installation) depuis le site <https://packagecontrol.io/installation>
- copier le fichier dans le dossier Installed Packages/
- Redémarrer Sublime Text

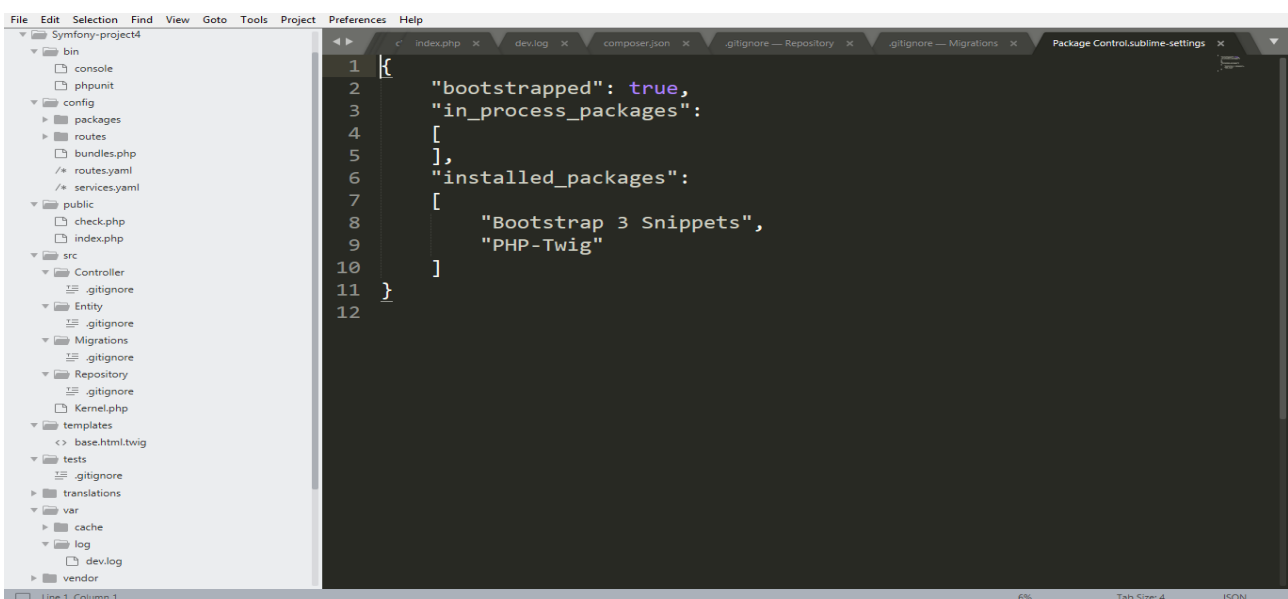
✓ Installer Bootstrap3Snippet

- Accéder à Préférence→Package Control
- Chercher Package Control : Install package
- Cliquer sur Bootstrap3Snippet pour l'installer
- Redémarrer sublime3

✓ Installer phpTwig

- Suivre les mêmes étapes.

✓ En accédant à Préférence→Package Settings → Package Control→ Setting–User, vous aurez le listing de vos packages installés.



```
1 {
2   "bootstrapped": true,
3   "in_process_packages":
4   [
5   ],
6   "installed_packages":
7   [
8     "Bootstrap 3 Snippets",
9     "PHP-Twig"
10  ]
11 }
12
```


Partie 8 : Description de notre projet EPIJob

On se propose, durant ce module (tout au long du semestre), de créer un site web symfony4 qui permet aux candidats à la recherche *d'emploi*, et les entreprises à la recherche de collaborateurs de retrouver les meilleures offres qui répondent à leurs besoins.

Un projet Symfony repose sur le pattern MVC (Model View Controller). Ce type d'architecture permet d'organiser le code en le séparant en trois couches :

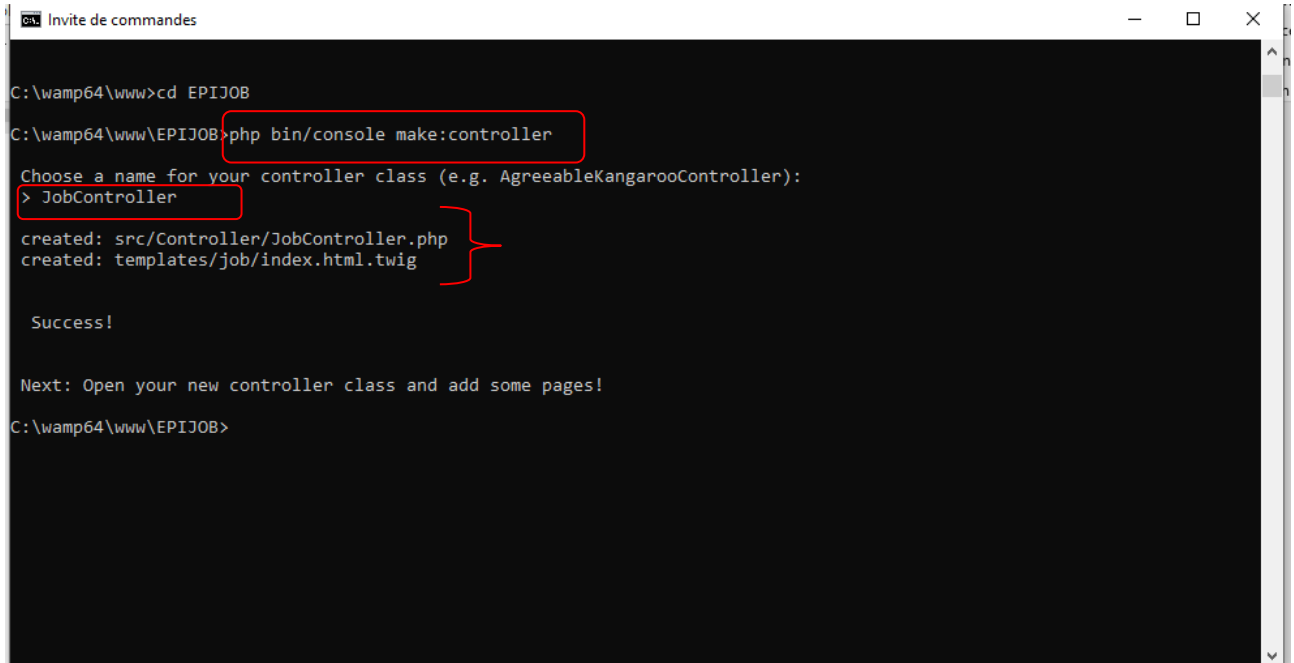
1. La couche **Modèle** contenant le traitement logique de vos données (on y retrouve les traitements métier, les accès à la base de données, ...).
2. La **Vue** est la modélisation de l'IHM (Interface Homme Machine). Elle représente ce qui est rendu à l'utilisateur (sous la forme d'une page Web, de données JSON/XML, ...).
3. Le **Contrôleur** correspond au code faisant le lien entre le modèle et la vue. Il récupère les données utilisateurs pour y appliquer les traitements et donner les résultats à la vue.

Dans notre projet Web, cela va se matérialiser par des classes qui vont contenir des fonctions qui seront appelées en fonction d'une URL.

Ces dernières renverront un objet de type `Symfony\Component\HttpFoundation\Response` qui est l'abstraction d'une réponse HTTP dans Symfony.

- Lancer la création de notre contrôleur principale

```
php bin/console make:controller
```



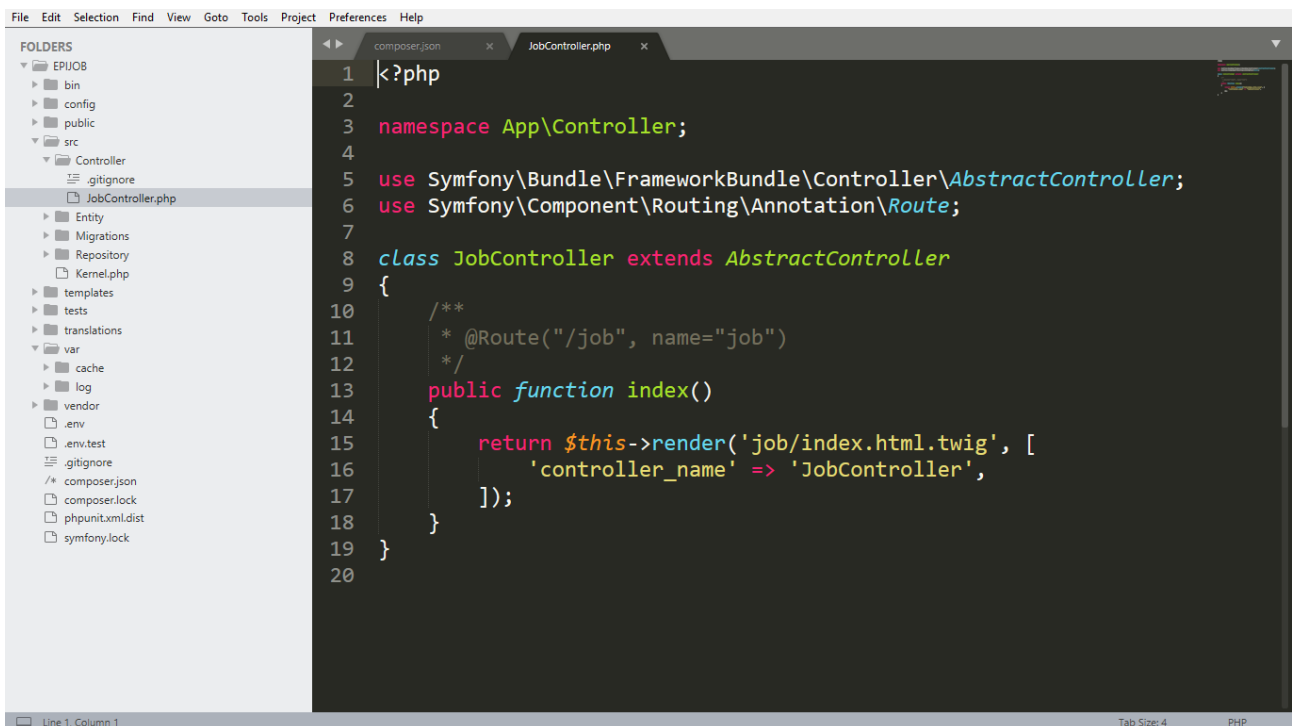
```
C:\wamp64\www>cd EPIJOB
C:\wamp64\www\EPIJOB>php bin/console make:controller

Choose a name for your controller class (e.g. AgreeableKangarooController):
> JobController

created: src/Controller/JobController.php
created: templates/job/index.html.twig

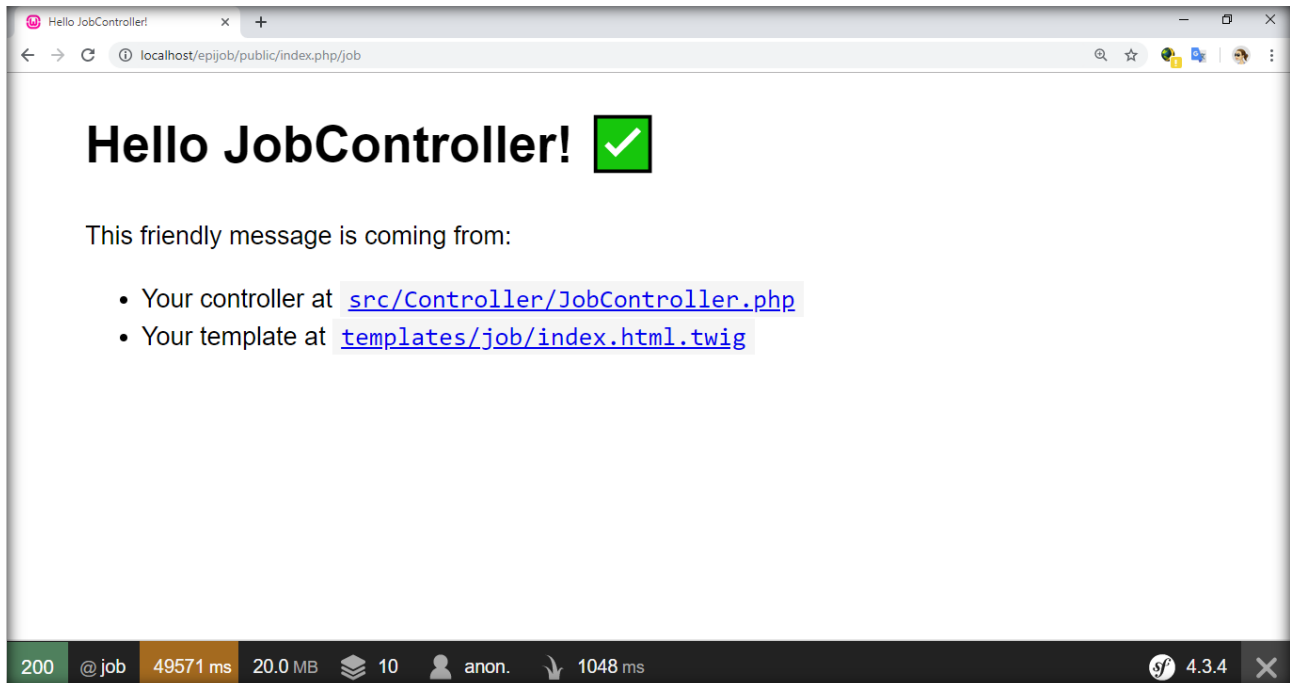
Success!

Next: Open your new controller class and add some pages!
C:\wamp64\www\EPIJOB>
```



```
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  EPIJOB
    bin
    config
    public
    src
      Controller
        JobController.php
      Entity
      Migrations
      Repository
      Kernel.php
    templates
    tests
    translations
    var
      cache
      log
    vendor
    .env
    .env.test
    .gitignore
    /* composer.json
    composer.lock
    phpunit.xml.dist
    symfony.lock

1 |<?php
2
3 | namespace App\Controller;
4
5 | use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
6 | use Symfony\Component\Routing\Annotation\Route;
7
8 | class JobController extends AbstractController
9 | {
10 |     /**
11 |      * @Route("/job", name="job")
12 |      */
13 |     public function index()
14 |     {
15 |         return $this->render('job/index.html.twig', [
16 |             'controller_name' => 'JobController',
17 |         ]);
18 |     }
19 | }
20
```



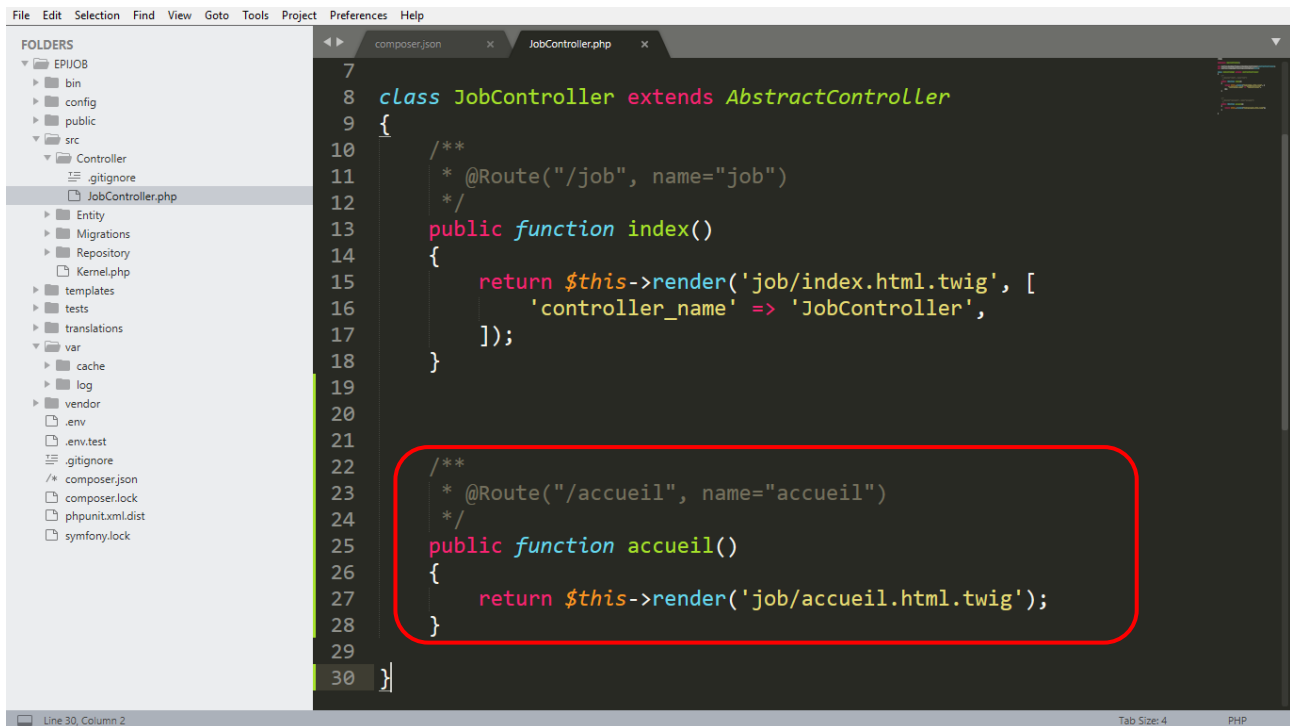
➤ **NB :** La mission du contrôleur est de RETOURNER UNE REPONSE !!

Symfony s'est inspiré des concepts du protocole HTTP.

Dans Symfony, il existe les classes `Request` et `Response`.

Le rôle du contrôleur est de retourner au noyau un objet `Response`, qui contient la réponse HTTP à envoyer à l'internaute (page HTML ou redirection).

Partie 9: Toolbar symfony

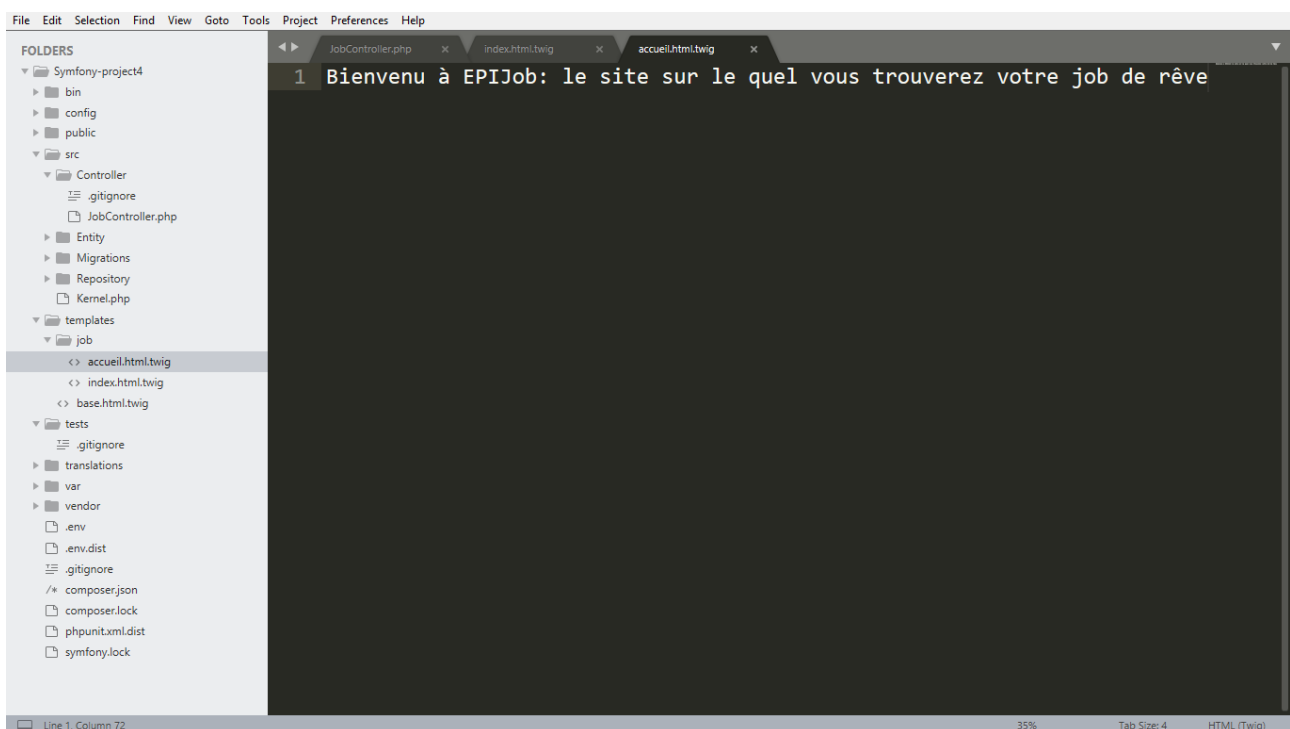


```

7
8 class JobController extends AbstractController
9 {
10     /**
11      * @Route("/job", name="job")
12      */
13     public function index()
14     {
15         return $this->render('job/index.html.twig', [
16             'controller_name' => 'JobController',
17         ]);
18     }
19
20
21
22     /**
23      * @Route("/accueil", name="accueil")
24      */
25     public function accueil()
26     {
27         return $this->render('job/accueil.html.twig');
28     }
29
30 }

```

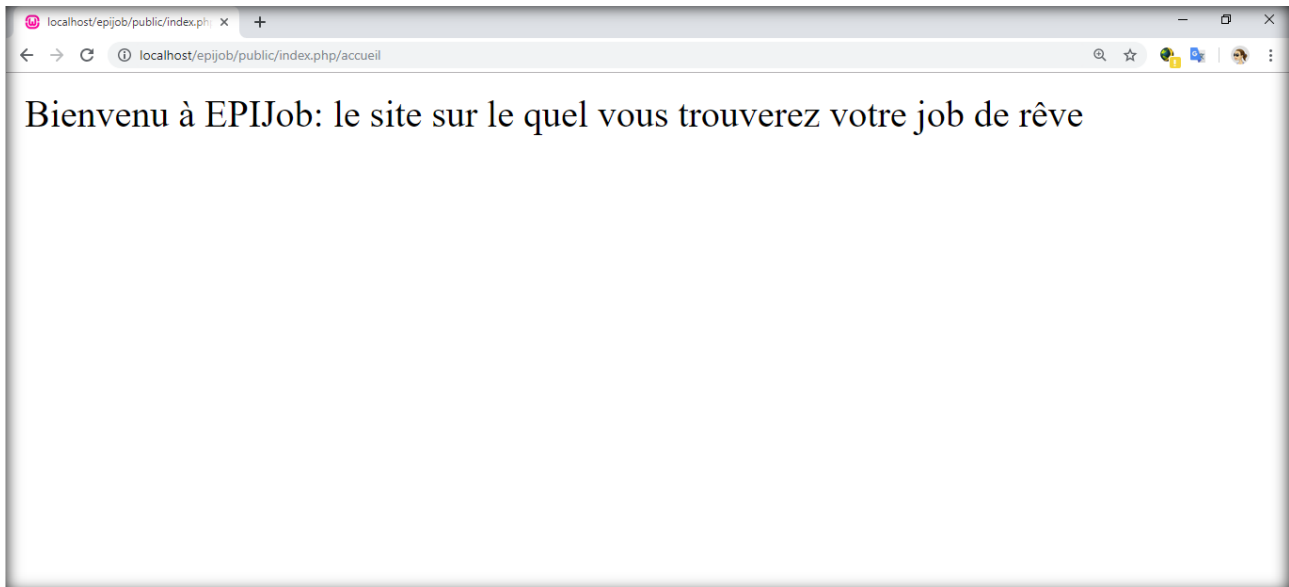
Créer la template correspondante :



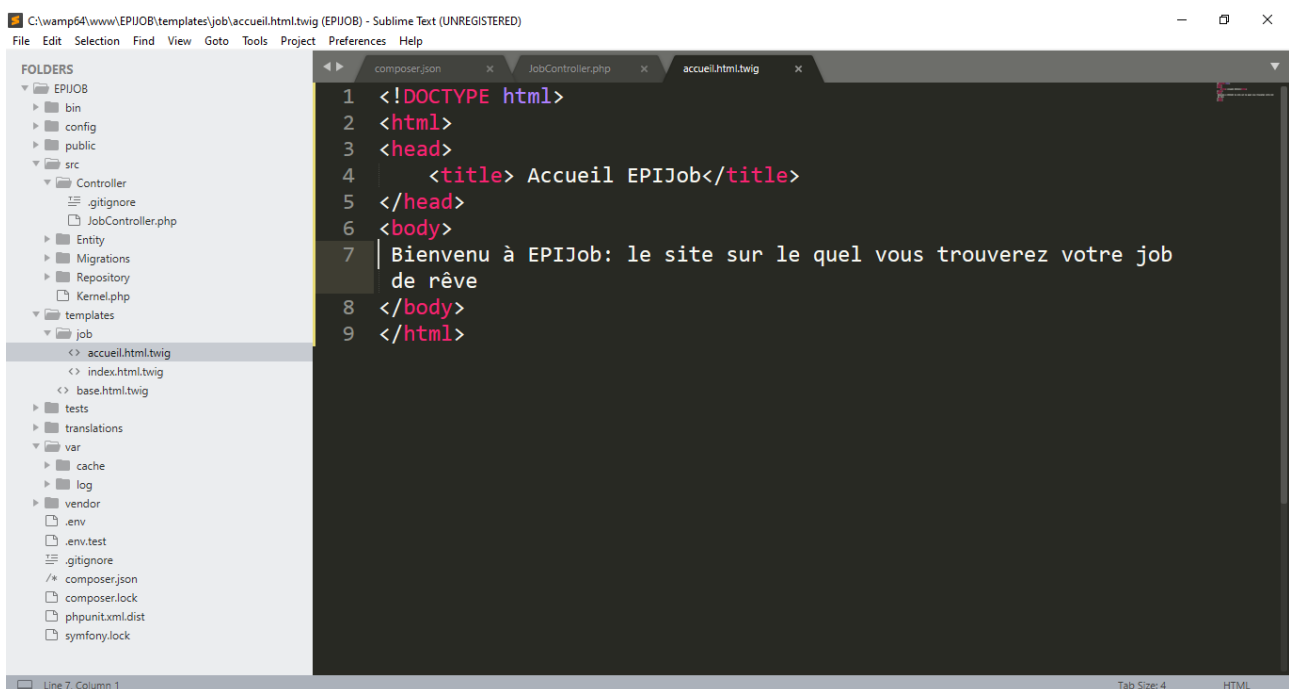
```

1 Bienvenu à EPIJob: le site sur le quel vous trouverez votre job de rêve

```



La **toolbar** (barre de débogage en français) est un petit bout de code HTML que rajoute Symfony à chaque page contenant la balise `</body>`. Or sur cette page, vous pouvez afficher la source depuis votre navigateur, il n'y a aucune balise HTML, donc Symfony n'ajoute pas la **toolbar**.



A screenshot of a web browser window. The address bar shows "localhost/epijob/public/index.php/accueil". The page content says "Bienvenu à EPIJob: le site sur le quel vous trouverez votre job de rêve". A developer console is open on the left, showing details for the "accueil" route. A red rectangle highlights the bottom status bar of the browser, which includes the status code "200", the URL "@ accu...", response time "703 ms", size "2.0 MB", number of requests "4", user "anon.", and server response time "28 ms". The Symfony logo and version "4.3.4" are also visible in the status bar.

Accueil EPIJob

localhost/epijob/public/index.php/accueil

Bienvenu à EPIJob: le site sur le quel vous trouverez votre job de rêve

HTTP status 200 OK
Controller [JobController :: accueil](#)
Route name accueil
Has session yes

200 @ accu... 703 ms 2.0 MB 4 anon. 28 ms sf 4.3.4