

Les bases de Symfony

Table des matières

I. Création d'un projet Symfony	3
A. Création d'une application Symfony avec Composer	3
B. Création d'une application Symfony avec Symfony CLI.....	4
II. Exercice : Quiz	9
III. Structure des dossiers d'un projet Symfony et lancement de l'application	10
A. Structure des dossiers	11
B. Fichiers importants	12
C. Lancement de notre application	13
IV. Exercice : Quiz	14
V. Construire votre première route	14
VI. Exercice : Quiz	15
VII. Affichez votre première page HTML avec Twig	16
VIII. Essentiel	17
IX. Auto-évaluation	18
A. Exercice	18
B. Test.....	18
Solutions des exercices	19

I. Création d'un projet Symfony

Durée : 1 h

Prérequis : connaître PHP et les bases du terminal

Environnement de travail :

- Ordinateur avec Windows, macOS ou Linux
- Éditeur de code (Visual Code est conseillé)
- Un navigateur récent : Firefox, Chrome, Safari, Opera
- PHP > 7.2.5 installé avec les librairies suivantes :
 - Ctype
 - iconv
 - JSON
 - PCRE
 - Session
 - SimpleXML
 - Tokenizer
- Composer

Contexte

Symfony a été développé au sein de SensioLabs, une société de service française. Son auteur principal est Fabien Potencier.

Symfony fait partie d'une série de frameworks qui ont fleuri afin de simplifier la manipulation de PHP, d'accélérer le développement, d'écrire moins de code, de partager des bibliothèques, d'avoir un code plus robuste et plus sécurisé, et de suivre de bonnes pratiques de code. Il s'inscrit dans une lignée de frameworks PHP comme CakePHP, CodeIgniter, Laravel, pour n'en citer que quelques-uns.

Ce cours va vous permettre de vous lancer dans la construction de votre première application Web Symfony en vous donnant les clés initiales pour comprendre les rouages du framework avant de plonger dans une construction plus avancée.

Il y a deux manières de créer un projet Symfony. La première se fait simplement avec Composer, le gestionnaire de dépendances pour PHP. La seconde utilise une ligne de commande dédiée, un petit exécutable Symfony appelé Symfony CLI. Nous allons voir les deux façons et comprendre les avantages de chacune.

A. Création d'une application Symfony avec Composer

L'installation via Composer se fera avec quelques lignes de commande.

Vous allez d'abord télécharger la structure basique de Symfony, appelée **skeleton**, vous rendre dans le dossier du projet et installer la dépendance **webapp** qui vous permettra d'avoir toutes les ressources nécessaires pour développer une application web complète, comme l'envoi d'e-mail, la gestion des formulaires ou de la base de données.

L'intérêt de cette méthode est qu'elle ne nécessite aucune installation particulière en plus de celle de Composer. Elle est rapide, mais elle n'offre aucun outil supplémentaire.

```
1 composer create-project symfony/skeleton:"^5.4" mon_repertoire_de_projet
2 cd mon_repertoire_de_projet
3 composer require webapp
```

Complément

« La dépendance **webapp** est un pack de composants à installer pour disposer de tous les outils nécessaires au développement d'une application web ». Il existe d'autres packs officiels permettant d'étendre Symfony sur plein de sujets (ORM, debug, tests, etc.). On peut retrouver ces packs sur le site de Composer¹.

Exemple Installation de Symfony avec Composer

B. Création d'une application Symfony avec Symfony CLI

La création d'un projet Symfony via Symfony CLI se fait en deux temps. Tout d'abord, il faudra installer Symfony CLI sur votre machine. Ensuite, grâce à ce programme, vous pourrez créer vos applications Web.

Symfony CLI (**Command Line Interface**) est un outil de développement qui vous aide à construire, exécuter et gérer vos applications Symfony directement depuis votre terminal. Il est open source, fonctionne sur macOS, Windows et Linux. Comme vous l'installez globalement à votre machine, **vous ne devez l'installer qu'une seule fois sur votre système**. Cet outil vous simplifiera énormément le développement par la suite, donc **il est fortement recommandé de l'installer**.

Méthode Installer Symfony CLI

L'installation de Symfony CLI va dépendre du système que vous utilisez. Dans tous les cas, cela permettra d'installer Symfony CLI globalement sur votre machine.

Linux ou macOS

Pour ces deux systèmes, une seule ligne de commande suffit :

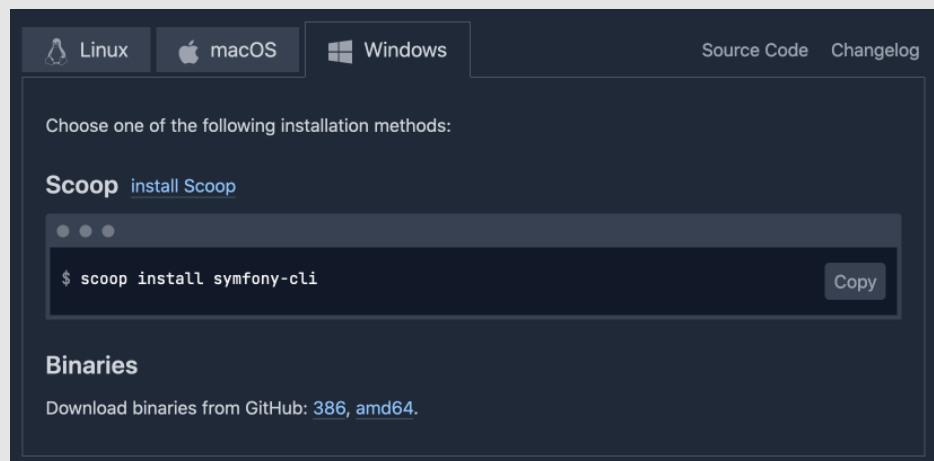
```
1 curl -sS https://get.symfony.com/cli/installer | bash
```

Installation de Symfony CLI sur macOS

Windows

Pour Windows, il faudra un peu plus d'étapes.

1. Tout d'abord, il faut télécharger l'exécutable de Symfony CLI. L'exécutable est téléchargeable directement sur le site de Symfony : Symfony²



¹ <https://getcomposer.org/>

² <https://symfony.com/download>

Source : symfony¹

Dans la section **Binaries**, cliquez sur la version correspondant à votre système : **386** pour les ordinateurs Intel 32 bits et **amd64** pour les ordinateurs avec un processeur AMD 64 bits.

Cela va télécharger l'exécutable.

2. Dans le dossier « *C:\Program Files* », créez un dossier nommé « *Symfony* ».

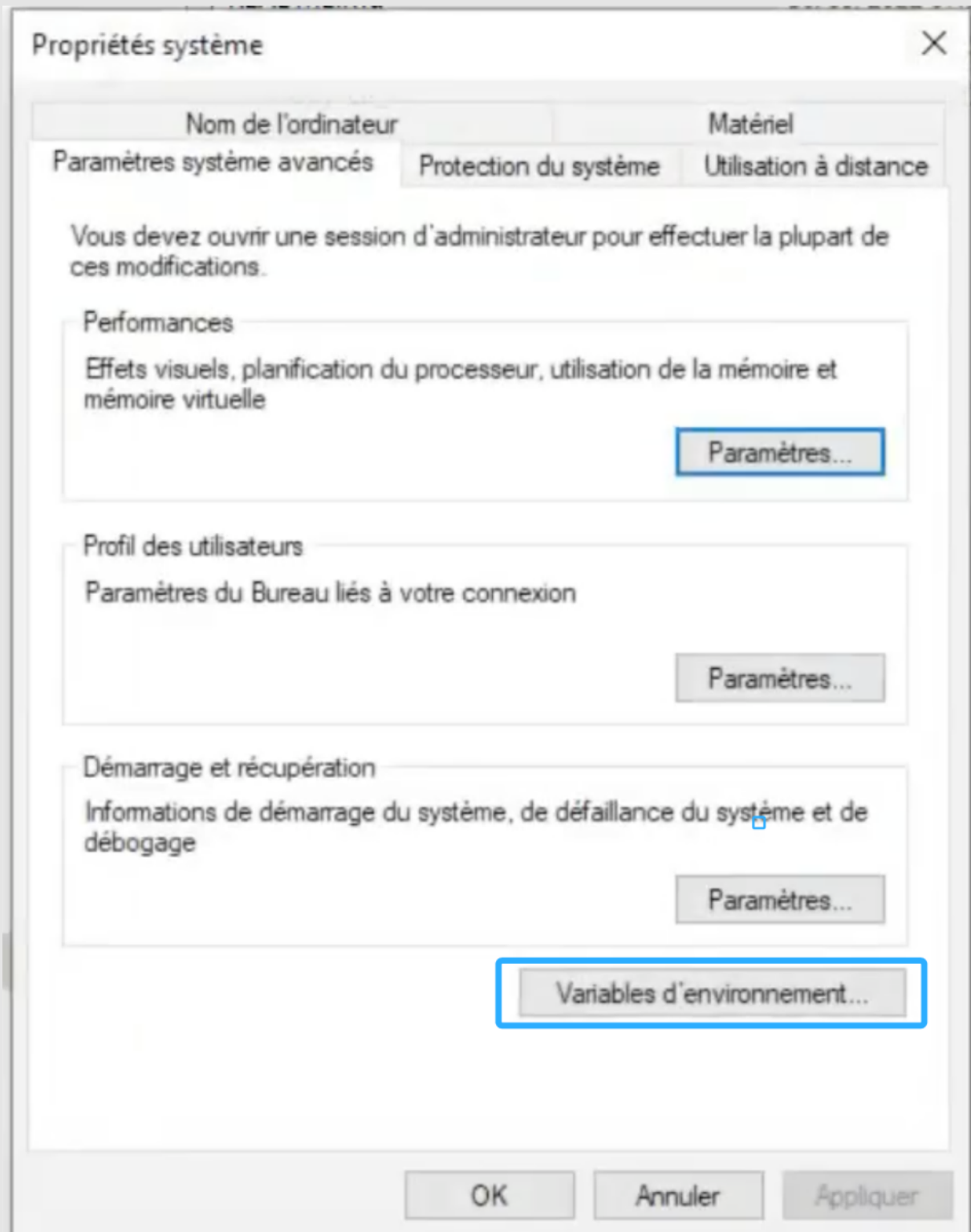
3. Déplacez l'archive téléchargée à l'étape 1 dans ce dossier.

4. Décompressez l'archive dans ce dossier. Vous devriez avoir 3 fichiers : « *LICENSE* », « *README.md* » et « *symfony.exe* ».

5. Il faudra ensuite configurer les variables d'environnement, et notamment la variable *PATH*, afin de rendre la commande disponible n'importe où en ligne de commande. Cette variable définit les chemins de votre système dans lesquels la ligne de commande pourra chercher globalement des commandes.

Pour configurer cette variable, dans la barre de recherche de Windows, tapez « *env* ». Le premier résultat devrait être « *Modifier les variables d'environnement système* ». Cliquez dessus, la fenêtre des **Propriétés système** va s'afficher.

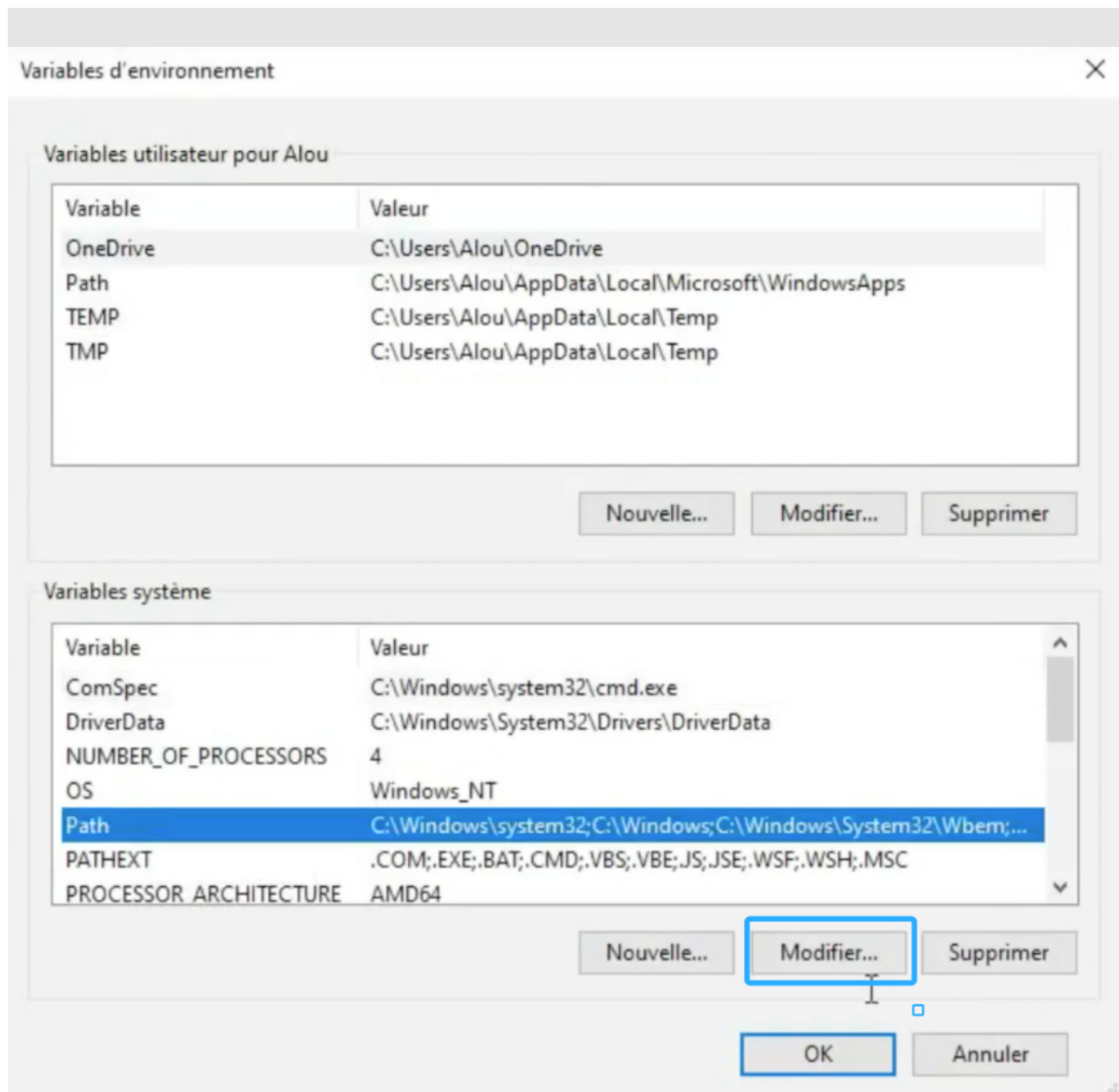
¹ <https://symfony.com/>



Source : YouTube¹

6. Cliquez alors sur le bouton « **Variables d'environnement** ». Cela ouvrira la fenêtre des variables d'environnement.

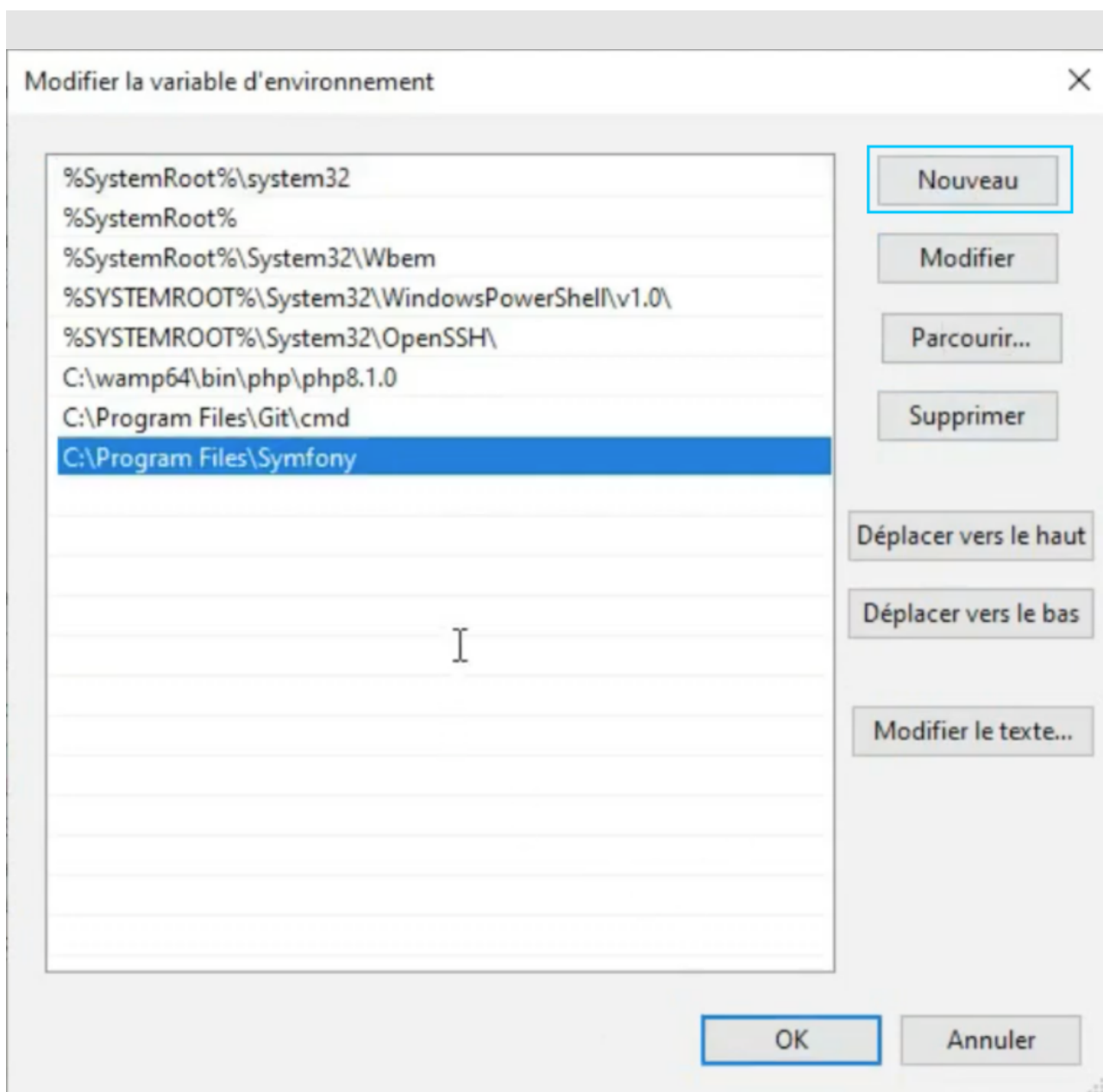
¹ https://www.youtube.com/supported_browsers?next_url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DX9asQwfBYhA



Source : YouTube¹

7. Dans la section « **Variables système** », choisissez « **Path** » et cliquez sur le bouton « **Modifier** ». Cela va ouvrir une troisième fenêtre pour modifier cette variable.

¹ <https://www.youtube.com/watch?v=X9asQwfBYhA>

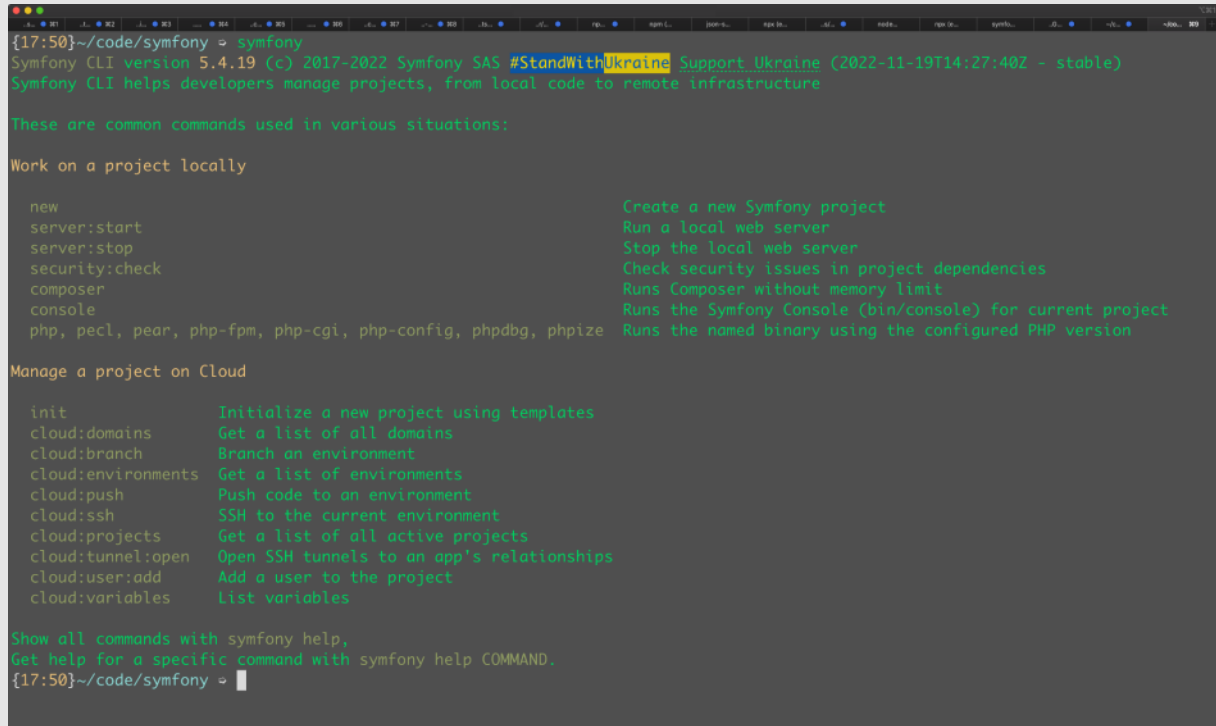


Source : YouTube¹

8. Cliquez sur le bouton « **Nouveau** » et saisissez le chemin du dossier contenant l'exécutable de Symfony CLI (symfony.exe) décompressé à l'étape 4. Dans notre cas ce sera : « *C:\Program Files\Symfony* ».

¹ <https://www.youtube.com/watch?v=X9asQwfBYhA>

9. Cliquez sur les boutons « **OK** » des trois fenêtres afin de toutes les fermer. En ouvrant maintenant un terminal et en tapant la commande `symfony` vous devriez avoir un écran similaire à celui-ci :



```
{17:50}~/code/symfony ~ symfony
Symfony CLI version 5.4.19 (c) 2017-2022 Symfony SAS #StandWithUkraine Support Ukraine (2022-11-19T14:27:40Z - stable)
Symfony CLI helps developers manage projects, from local code to remote infrastructure

These are common commands used in various situations:

Work on a project locally

new                                Create a new Symfony project
server:start                       Run a local web server
server:stop                       Stop the local web server
security:check                    Check security issues in project dependencies
composer                          Runs Composer without memory limit
console                           Runs the Symfony Console (bin/console) for current project
php, pecl, pear, php-fpm, php-cgi, php-config, phpdbg, phpsize  Runs the named binary using the configured PHP version

Manage a project on Cloud

init                               Initialize a new project using templates
cloud:domains                     Get a list of all domains
cloud:branch                      Branch an environment
cloud:environments                Get a list of environments
cloud:push                       Push code to an environment
cloud:ssh                        SSH to the current environment
cloud:projects                   Get a list of all active projects
cloud:tunnel:open                Open SSH tunnels to an app's relationships
cloud:user:add                   Add a user to the project
cloud:variables                  List variables

Show all commands with symfony help,
Get help for a specific command with symfony help COMMAND.
{17:50}~/code/symfony ~
```

Installation de Symfony avec Symfony CLI

Maintenant que Symfony CLI est installé, il est temps de créer votre application. Pour cela, il suffit de se rendre dans un terminal et de taper la commande suivante :

```
1 symfony new mon_projet --version=5.4 --webapp
```

Cela va créer un dossier nommé « *mon_projet* », puis télécharger toute la structure de l'application dans ce dossier. Bien entendu, vous pouvez choisir le nom du dossier comme vous le souhaitez.

Si vous ne souhaitez pas utiliser la dernière version de Symfony, il est important de préciser la version que vous souhaitez utiliser grâce au flag `--version`. Dans notre cas, nous souhaitons utiliser la dernière LTS, la version 5.4, donc nous précisons `--version=5.4`.

Exemple Installation de Symfony avec Symfony CLI

Conseil

Bien que facultative, l'installation de Symfony est fortement recommandée, car son utilisation vous aidera grandement dans le développement de votre application.

Exercice : Quiz

[solution n°1 p.21]

Question 1

Symfony est le seul framework PHP sur le marché.

- ☐ Vrai
- ☐ Faux

Question 2

Composer est un outil en ligne de commande qui permet de gérer les dépendances de notre projet.

- ☐ Vrai
- ☐ Faux

Question 3

Composer permet d'initialiser une application Symfony.

- ☐ Vrai
- ☐ Faux

Question 4

Symfony CLI ne fonctionne que sur Windows.

- ☐ Vrai
- ☐ Faux

Question 5

À la création d'une application Symfony, on ne peut pas choisir sa version.

- ☐ Vrai
- ☐ Faux

Question 6

Symfony CLI doit être installée à chaque nouveau projet.

- ☐ Vrai
- ☐ Faux

Question 7

Symfony CLI est un gadget qui n'est pas très utile.

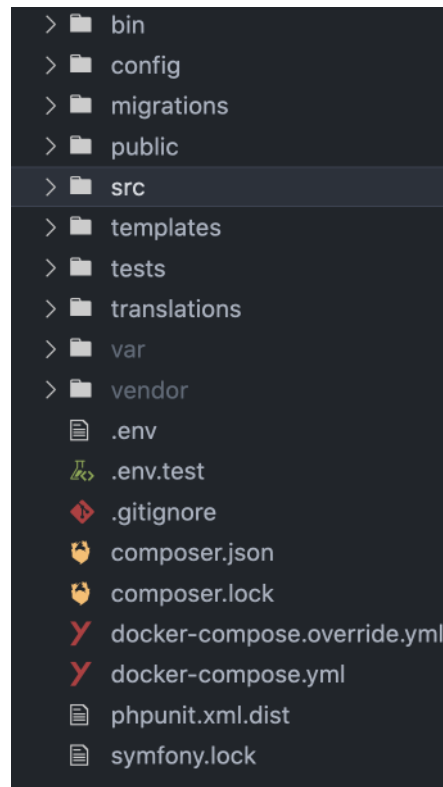
- ☐ Vrai
- ☐ Faux

III. Structure des dossiers d'un projet Symfony et lancement de l'application

Maintenant que votre application est créée, nous allons explorer en détail les entrailles des dossiers afin de comprendre à quoi ils servent. Nous lancerons enfin notre application avec le serveur intégré de Symfony CLI.

A. Structure des dossiers

Ouvrez votre application dans votre éditeur de code favori. Vous devriez trouver la structure de dossiers suivante :



- **bin**

Contient certains exécutables comme la console ou phpunit pour lancer les tests. Vous n'intervenez que très peu sur ce dossier.

- **config**

Contient tous les fichiers de configuration de l'application. Cela peut être la configuration de Symfony lui-même ou bien celle des packages installés par Composer. Vous serez amené à beaucoup travailler dans ce dossier pour donner des indications sur le fonctionnement de votre application.

- **migrations**

Ce dossier contient l'historique, étape par étape, de la construction de la base de données. À chaque nouvelle modification de la structure de la base de données (ajout/suppression d'une table, ajout/suppression d'un champ, changement de type d'un champ, par exemple) un fichier reflétant l'opération réalisée sera généré, permettant ainsi de suivre la construction de la base de données et pour pouvoir la recréer automatiquement.

Les outils de Symfony CLI liés à la base de données se servent de ce dossier pour travailler.

- **public**

Ce dossier contient le premier fichier qui sera lu lors du lancement de l'application. Il contient également tout ce que le visiteur du site sera autorisé à voir : fichiers HTML, CSS, js, les images, etc. Le visiteur du site n'a pas accès à ce qui se trouve en dehors de ce dossier. Il est donc fondamental de ne rien mettre dans ce dossier qui serait confidentiel.

- **src**

C'est le dossier principal de votre application. Tout le code de votre application se trouvera dans ce dossier. Vous y passerez donc la plupart de votre temps.

- **templates**

Ce dossier contient toutes les vues de l'application, c'est-à-dire vos fichiers HTML, mais avec quelques particularités. Les fichiers templates utilisent le moteur de rendu TWIG pour travailler avec Symfony.

- **tests**

Contient tous les fichiers permettant d'effectuer des tests unitaires, d'intégration ou d'application. La structure de ce dossier doit obligatoirement reprendre la structure du dossier « /src » de votre application. PHPUnit cherchera par défaut les fichiers de test dans ce dossier.

- **translations**

Si votre application gère le multilingue, ce dossier contiendra les fichiers de traductions. Chaque fichier pouvant correspondre à un domaine particulier et à une langue donnée.

- **var**

Accessible en écriture par le serveur, ce dossier contient les fichiers temporaires, les fichiers de travail, le cache, etc. Vous n'avez généralement pas besoin d'aller dans ce dossier.

- **vendor**

Contient tous les dossiers système nécessaires au fonctionnement interne de Symfony. Ce dossier contient également les dossiers des packages installés par Composer.

B. Fichiers importants

À la racine du projet se trouvent quelques fichiers importants dont nous allons voir l'utilité.

Le fichier « .env »

Ce fichier permet de stocker toutes les informations sensibles dont votre application a besoin et que vous ne souhaitez pas exposer à tout le monde. Il peut contenir des mots de passe, des clés d'API, des informations de connexion à une base de données, etc. Ce fichier ne sera pas versionné et ainsi pas exposé dans votre dépôt Git.

Les fichiers « composer.json » et « composer.lock »

Ces deux fichiers contiennent la liste des dépendances de votre projet. La différence entre les deux est subtile, mais importante.

- Le fichier **composer.json** contient la liste des dépendances de premier niveau (celles dont votre projet a effectivement besoin) avec comme précision une plage de versions acceptables (ni trop ancienne, ni trop récente par exemple). **C'est une sorte de liste de souhaits.**
- Le fichier **composer.lock**, lui, contient la liste de l'ensemble de toutes les dépendances réellement installées avec la version précise installées ainsi que toutes les sous-dépendances, c'est-à-dire les dépendances des dépendances. **C'est un registre de toutes les dépendances installées dans votre dossier /vendor.**

Le fichier « symfony.lock »

Symfony possède un système de recettes appelé *Flex*. Une recette est un ensemble de dépendances déjà configurées pour fonctionner ensemble.

Symfony Flex est un outil qui rend l'ajout de nouvelles fonctionnalités très simple. Grâce à une seule ligne de commande, vous pouvez utiliser une des recettes proposées par Symfony pour ajouter une fonctionnalité.

Le fichier *symfony.lock* est le fichier qui tient le registre des dépendances installées via des recettes, ce qui est plus simple à suivre que de le deviner à partir du fichier *composer.lock*.

C. Lancement de notre application

Voilà, Symfony est installé. Vous avez compris de quoi il était fait, maintenant vous voulez sans doute savoir à quoi il ressemble. Pour cela, vous allez lancer un serveur PHP afin de pouvoir interpréter le code PHP et afficher vos pages dans le navigateur.

Complément Serveur interne à PHP

Si vous avez installé Symfony avec Composer et sans l'aide de Symfony CLI, vous pouvez utiliser le serveur interne de PHP de cette manière :

```
1 php -S 127.0.0.1:8000 -t public/
```

127.0.0.1 correspond à l'adresse IP locale de votre machine.

8000 correspond au port sur lequel vous voulez lancer votre application. Vous pouvez choisir celui que vous souhaitez, mais attention cependant, certains ports sont réservés pour des usages précis. Le port par défaut est généralement 8000.

-t public/ précise à Symfony que le point d'entrée de votre application se trouve dans le dossier « *public/* » (implicitement, c'est le fichier « *index.php* » qui sera sollicité). Il est indispensable de préciser ce paramètre pour lancer une application Symfony via le serveur interne de PHP.

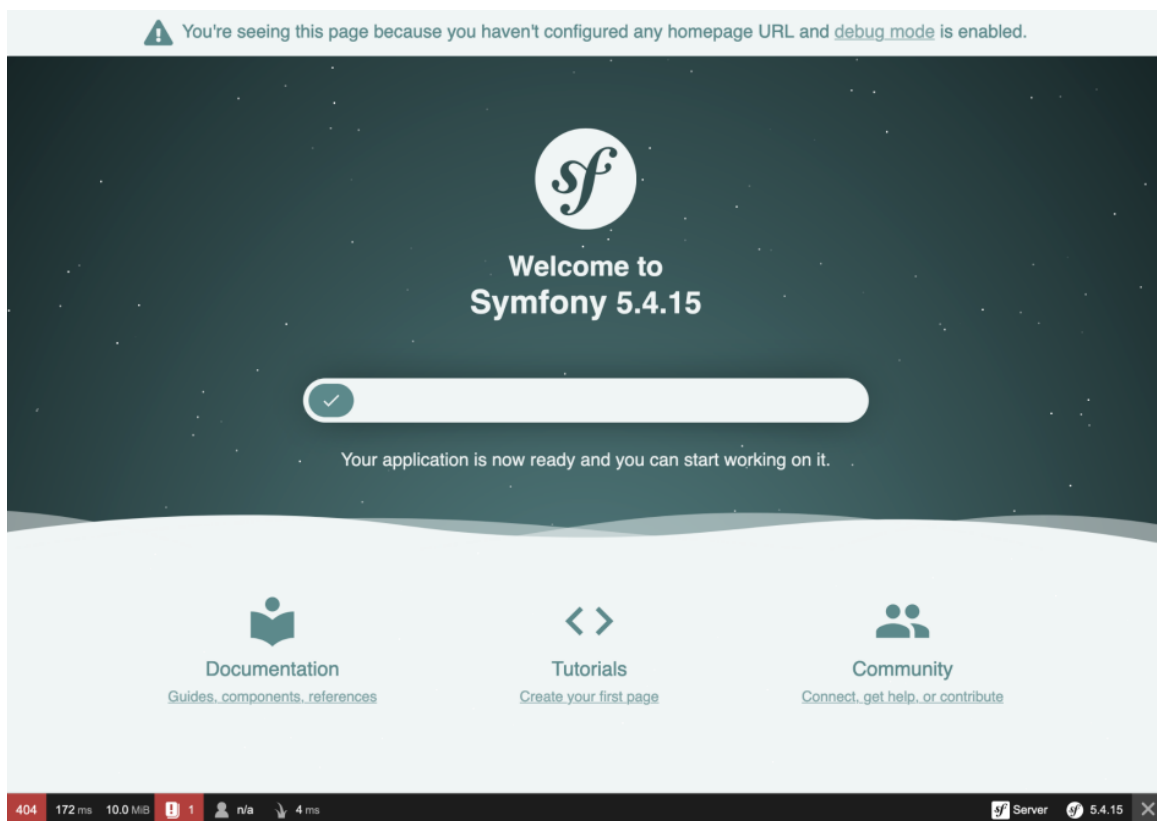
Serveur de Symfony CLI

Grâce à Symfony CLI, vous avez, un serveur intégré ainsi que d'autres outils. Pour lancer le serveur, il suffit d'une ligne de commande :

```
1 symfony server:start -d
```

L'option -d (pour *daemon*) lance le serveur en arrière-plan.

Dans tous les cas, en vous rendant sur l'adresse <http://127.0.0.1:8000> vous devriez obtenir cette page, félicitations !



Exercice : Quiz

[solution n°2 p.22]

Question 1

C'est dans le dossier « */vendor* » que le code propre à votre application sera écrit.

- ☐ Vrai
- ☐ Faux

Question 2

Que contient le dossier « */migrations* » ?

- ☐ Des documentaires sur les oiseaux
- ☐ Un ensemble de fichiers qui retrace l'historique de la construction de la base de données
- ☐ Les fichiers de tests

Question 3

Tout ce qui se trouve en dehors du dossier « */public* » sera inaccessible au visiteur du site.

- ☐ Vrai
- ☐ Faux

Question 4

Symfony CLI possède un serveur interne.

- ☐ Vrai
- ☐ Faux

Question 5

L'adresse 127.0.0.1 correspond à l'adresse IP locale de ma machine.

- ☐ Vrai
- ☐ Faux

V. Construire votre première route

Définition

Une route est une adresse internet propre à notre site. Cette route est un point d'accès qui peut retourner une page ou des données, par exemple. Pour naviguer dans notre site, nous avons besoin de définir un ensemble de routes.

Lorsque vous arrivez sur la page initiale de Symfony, un message d'alerte tout en haut de la page précise ceci : « *You're seeing this page because you haven't configured any homepage URL and debug mode is enabled.* »

Cela signifie qu'aucune route racine n'a été définie. Cette route racine correspond à l'URL /. Vous allez donc en créer une. Pour cela, vous devrez créer un Controller, car ce sont eux qui connaissent les routes auxquelles votre application peut répondre.

Définition

Symfony est un framework HTTP dont le but principal est de transformer une requête en réponse. Pour cela, plusieurs architectures sont possibles, mais on utilisera le plus répandu d'entre eux : le modèle MVC, pour **Model**, **View**, **Controller**, qui convient parfaitement pour réaliser des applications web classiques.

C'est une organisation du code qui permet de correctement séparer les responsabilités des portions de codes de notre application. Les models servent à interagir avec notre base de données, les views contiennent le rendu visuel de nos pages et les controllers servent d'intermédiaire et pilotent les interactions entre toutes nos portions de code. Il est en quelque sorte le chef d'orchestre de notre application.

Méthode

Ici, le controller est chargé de connaître les routes disponibles sur votre application et d'exécuter du code lorsque vous vous connectez à cette route.

Pour créer votre première route, il vous faut créer votre premier Controller. Pour cela :

1. Rendez-vous dans le dossier « `/src/Controller` » et créez un fichier « `HomeController.php` ».
2. Collez le code suivant dans ce fichier :

```
1 <?php
2 namespace App\Controller;
3
4 use Symfony\Component\Routing\Annotation\Route;
5 use Symfony\Component\HttpFoundation\Response;
6
7 class HomeController
8 {
9     #[Route('/')]
10    public function number() : Response
11    {
12        return new Response(rand(0,100));
13    }
14 }
```

3. Désormais, si vous rafraîchissez la page initiale, vous devriez voir un nombre compris en 0 et 100 s'afficher. Félicitations, vous venez de créer votre première route !

Exemple**Exercice : Quiz**

[solution n°3 p.23]

Question 1

Les routes dans Symfony reflètent la structure des dossiers de l'application.

- ☐ Vrai
- ☐ Faux

Question 2

Quelle est la bonne syntaxe pour lancer le serveur interne à Symfony ?

- ☐ `composer serveur:launch`
- ☐ `symfony start`
- ☐ `symfony server:start -d`

Question 3

Le code de notre application sera morcelé en plusieurs fichiers qui auront chacun leur responsabilité.

- ☐ Vrai
- ☐ Faux

Question 4

Il est obligatoire d'avoir plusieurs routes sur notre application.

- ☐ Vrai
- ☐ Faux

Question 5

Un controller ne peut retourner qu'une page HTML.

- ☐ Vrai
- ☐ Faux

VII. Affichez votre première page HTML avec Twig

Si vous voulez retourner du HTML depuis votre controller, vous voudrez probablement rendre un modèle de page. On appelle ce modèle de page un **template**. Heureusement, Symfony est livré avec un moteur de template appelé **Twig** : un langage de templating minimal, puissant et assez facile à prendre en main.

Définition

Un système de template permet de générer des pages HTML classiques tout en injectant des variables provenant de Symfony. Nous générons alors des Vues. Ce sont les **Views** de la terminologie MVC. Ce mécanisme permet une approche très puissante de modèles de pages.

Méthode

Pour que votre controller puisse générer des vues en Twig, il faut que ce controller étende de la classe abstraite `AbstractController`. Cette classe contient une collection de méthodes qui donneront à nos controller plus de fonctionnalités notamment sur la gestion des requêtes et des réponses.

Cette classe `AbstractController` contient aussi la méthode `render` qui permet de générer un rendu HTML depuis les templates Twig. Donc dans votre controller initial, il va falloir apporter quelques modifications :

1. Ajoutez la mention de l'extension de la classe grâce à `extends AbstractController`.
2. Importer la classe `AbstractController` qui se trouve dans le namespace `Symfony\Bundle\FrameworkBundle\Controller`.
3. Utilisez la vue Twig par défaut se trouvant dans le dossier « `/templates` ».
4. Utilisez la méthode `render` en précisant la vue à rendre et les paramètres que l'on souhaite passer à la vue.

```
1 <?php
2 namespace App\Controller;
3
4 use Symfony\Component\Routing\Annotation\Route;
5 use Symfony\Component\HttpFoundation\Response;
6 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
7
8 class HomeController extends AbstractController
```



```

9 {
10  #[Route('/')]
11  public function number() : Response
12  {
13      $number = rand(0, 100);
14      return $this->render('base.html.twig', [
15          'number' => $number,
16      ]);
17  }
18 }

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>{% block title %}Welcome!{% endblock %}</title>
6     <link rel="icon" href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22
viewBox=%220 128 128%22><text y=%221.2em%22 font-size=%2296%22>●</text></svg>">
7     {# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #}
8     {% block stylesheets %}
9         {{ encore_entry_link_tags('app') }}
10    {% endblock %}
11
12    {% block javascripts %}
13        {{ encore_entry_script_tags('app') }}
14    {% endblock %}
15  </head>
16  <body>
17      {% block body %}
18          <h1>Votre numéro porte-bonheur est le {{ number }}</h1>
19      {% endblock %}
20  </body>
21 </html>

```

Ici, vous utilisez le template de base fourni avec Symfony. Ce fichier se trouve dans le dossier « `/src/templates` ». Par défaut, la méthode `render` cherchera les templates dans ce dossier. Ainsi, si un template se trouve à la racine de ce dossier, il n'est pas nécessaire de donner toute l'arborescence.

Pour préciser les paramètres que vous souhaitez passer à votre vue, la méthode `render` attend un tableau associatif, contenant en clé l'identifiant du paramètre que vous souhaitez utiliser dans votre vue et en valeur la variable provenant de votre contrôleur.

Exemple Affichez votre première page HTML avec Twig

VIII. Essentiel

Pour créer une application **Symfony**, il suffit de lancer une simple commande dans un terminal. Il est fortement conseillé d'installer **Symfony CLI** pour créer une nouvelle application, lancer un serveur et interagir facilement avec votre application.

Composer sera également un outil de choix pour vous permettre d'ajouter des dépendances à votre projet.

La plus grande partie de **votre application** trouvera sa place dans **deux répertoires** : « `/config` » et « `/src` ». Le répertoire « `/templates` » peut aussi être important si votre application affiche des pages HTML.

Pour lancer le serveur de Symfony il suffit de taper la commande `symfony server:start -d`.

Les routes de notre application seront rattachées à des contrôleurs.

Pour rendre des pages HTML, nous utiliserons le moteur de template Twig qui génère des pages HTML avec des paramètres passés depuis la fonction `render`.

IX. Auto-évaluation

A. Exercice

Objectif de l'exercice :

Ce cas pratique va vous permettre de réunir toutes les connaissances nécessaires pour mettre en œuvre une application Symfony affichant une page HTML.

Contexte :

On vous demande de créer une application Symfony qui affiche un nombre aléatoire à chaque rafraîchissement. Ce nombre est votre nombre porte-bonheur. Ainsi, la page devra afficher « *Votre nombre porte-bonheur est le* » suivi du nombre en question. Développez les étapes à réaliser ci-dessous.

Question 1

[solution n°4 p.24]

Installez Symfony CLI.

Question 2

[solution n°5 p.24]

Créez un projet Symfony web appelé *porte-bonheur*.

Question 3

[solution n°6 p.24]

Lancez un serveur pour exécuter votre code.

Question 4

[solution n°7 p.24]

Créez un controller nommé *LuckyController*

Question 5

[solution n°8 p.24]

Créez une méthode *makeMeHappy* avec une route associée nommée« */porte-bonheur* » qui génère un nombre porte-bonheur aléatoire entre 0 et 100.

Question 6

[solution n°9 p.25]

Rendez la vue de base de Symfony en lui passant le nombre aléatoire généré.

Question 7

[solution n°10 p.25]

Utilisez le template de base pour afficher le message « *Votre nombre porte-bonheur est le* » suivi du nombre en question.

B. Test

Exercice 1 : Quiz

[solution n°11 p.26]

Question 1

Composer est utile pour :

- ☐ Écrire le prochain tube de l'été
- ☐ Ajouter des dépendances à votre projet
- ☐ Compiler le code PHP

Question 2

Parmi ces frameworks, lequel n'est pas un framework PHP ?

- ☐ Express
- ☐ Laravel
- ☐ CodeIgniter

Question 3

Le moteur de template permettant de générer les vues de Symfony s'appelle **Flig**.

- ☐ Vrai
- ☐ Faux

Question 4

Symfony doit obligatoirement tourner sur le port 8000.

- ☐ Vrai
- ☐ Faux

Question 5

La méthode `render` est directement disponible dans les controllers.

- ☐ Vrai
- ☐ Faux


Solutions des exercices

Exercice p. 9 Solution n°1**Question 1**

Symfony est le seul framework PHP sur le marché.

☐ Vrai

☒ Faux


 Il existe plusieurs frameworks PHP sur le marché, comme Laravel qui est le concurrent principal de Symfony. Mais il existe aussi CakePHP ou CodeIgniter par exemple.

Question 2

Composer est un outil en ligne de commande qui permet de gérer les dépendances de notre projet.

☒ Vrai

☐ Faux


 En effet, Composer permet d'ajouter des packages PHP pour apporter des fonctionnalités supplémentaires à notre application.

Question 3

Composer permet d'initialiser une application Symfony.

☒ Vrai

☐ Faux


 Oui, on peut créer une application Symfony complète avec Composer en quelques lignes de commande. Cependant, ce n'est pas la seule manière et l'installation d'une application Symfony avec Symfony CLI apportera de nombreux avantages.

Question 4

Symfony CLI ne fonctionne que sur Windows.

☐ Vrai

☒ Faux


 L'outil Symfony CLI est disponible sur les 3 systèmes principaux : Linux, macOS et Windows.

Question 5

À la création d'une application Symfony, on ne peut pas choisir sa version.


☐ Vrai

☒ Faux

 Que ce soit avec Composer ou avec Symfony CLI, il est toujours possible de choisir la version de Symfony que l'on souhaite utiliser au moment de sa création.


Question 6

Symfony CLI doit être installée à chaque nouveau projet.

- ☐ Vrai
- ☒ Faux
-  Symfony CLI est à installer une seule fois globalement sur votre machine. Ensuite, chaque nouveau projet ne demandera qu'une seule ligne de commande.

Question 7


Symfony CLI est un gadget qui n'est pas très utile.

- ☐ Vrai
- ☒ Faux
-  Symfony CLI est une boîte à outils qui vous servira dans de nombreuses situations. Elle contient notamment un serveur PHP intégré qui évitera l'installation de Nginx ou d'Apache. Il est fortement recommandé de l'installer pour simplifier le développement de votre application.

Exercice p. 14 Solution n°2


Question 1

C'est dans le dossier « */vendor* » que le code propre à votre application sera écrit.

- ☐ Vrai
- ☒ Faux
-  Le dossier « */vendor* » contient le code des composants de Symfony et d'autres paquets installés via Composer. Ce dossier ne doit jamais être édité ou versionné, car celui-ci sera souvent modifié par Composer. C'est dans le dossier « */src* » que le code métier sera écrit.


Question 2

Que contient le dossier « */migrations* » ?

- ☐ Des documentaires sur les oiseaux
- ☒ Un ensemble de fichiers qui retrace l'historique de la construction de la base de données
- ☐ Les fichiers de tests
-  En effet, le dossier « */migrations* » contient un ensemble de fichiers qui sont générés à chaque nouvelle modification de la structure de la base de données, permettant ainsi de suivre la construction de la base de données.


Question 3

Tout ce qui se trouve en dehors du dossier « */public* » sera inaccessible au visiteur du site.

- ☒ Vrai
- ☐ Faux
-  En effet, le dossier « */public* » est la seule partie du site qui sera accessible par le visiteur. On pourra y mettre les fichiers CSS, js, et les images, par exemple.


Question 4

Symfony CLI possède un serveur interne.

- ☒ Vrai
- ☐ Faux
-  Oui, parmi d'autres outils, Symfony CLI possède un serveur PHP intégré qui se lance avec une seule ligne de commande.

Question 5


L'adresse 127.0.0.1 correspond à l'adresse IP locale de ma machine.

- ☒ Vrai
- ☐ Faux
-  Chaque machine sur un réseau informatique doit être identifiée par une adresse unique. Pour atteindre votre machine locale sur un navigateur, c'est cette adresse qui sera utilisée. Vous pourrez rencontrer le terme *localhost* également pour désigner votre machine.

Exercice p. 15 Solution n°3


Question 1

Les routes dans Symfony reflètent la structure des dossiers de l'application.

- ☐ Vrai
- ☒ Faux
-  Les routes sont les points d'accès de notre application qui seront associés aux URLS de notre site.


Question 2

Quelle est la bonne syntaxe pour lancer le serveur interne à Symfony ?

- ☐ composer serveur:launch
- ☐ symfony start
- ☒ symfony server:start -d
-  Le nom de la commande de Symfony CLI pour lancer un serveur de développement est *server:start*. L'option *-d* (signifie daemon) permet de spécifier que le serveur tournera en tâche de fond sans monopoliser le terminal.

Question 3

Le code de notre application sera morcelé en plusieurs fichiers qui auront chacun leur responsabilité.

- ☒ Vrai
- ☐ Faux
-  En effet, c'est le principe du modèle MVC (Model / View / Controller) de séparer la logique de notre application dans plusieurs fichiers qui seront responsables de différentes actions. Les modèles (models) pour l'interaction avec notre base de données, les vues (views) pour notre affichage et les contrôleurs (controllers) pour gérer nos routes et faire office de chef d'orchestre.

Question 4

Il est obligatoire d'avoir plusieurs routes sur notre application.

☐ Vrai

☒ Faux

Q Non, il n'est pas obligatoire d'avoir plusieurs routes. Vous pouvez très bien avoir une application très simple avec une seule route et une seule page. En revanche, il est obligatoire d'avoir au moins une route pour pouvoir se connecter à la racine de notre domaine.

Question 5

Un controller ne peut retourner qu'une page HTML.

☐ Vrai

☒ Faux

Q Non, lorsque l'on se connecte à une route, le retour de la méthode de controller associé peut être au format HTML ou bien d'autres données comme du XML ou encore du JSON, si vous construisez une API par exemple.

p. 18 Solution n°4

Pour l'installation de Symfony CLI, se référer au chapitre correspondant dans le cours en fonction de votre système.

p. 18 Solution n°5

```
1 symfony new porte-bonheur --version=5.4 --webapp
```

p. 18 Solution n°6

```
1 symfony server:start
```

p. 18 Solution n°7

Se rendre dans le dossier /src/Controllers et créer un fichier nommé LuckyController.php

```
1 <?php
2     namespace App\Controller;
3
4     class LuckyController
5     {
6     }
7
```

p. 18 Solution n°8

```
1 <?php
2     namespace App\Controller;
3
4     use Symfony\Component\Routing\Annotation\Route;
5     use Symfony\Component\HttpFoundation\Response;
6
7     class LuckyController
8     {
```



```

9         #[Route('/porte-bonheur')]
10        public function makeMeHappy() : Response
11        {
12            return new Response(rand(0,100));
13        }
14    }

```

p. 18 Solution n°9

```

1 <?php
2     namespace App\Controller;
3
4     use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5     use Symfony\Component\Routing\Annotation\Route;
6     use Symfony\Component\HttpFoundation\Response;
7
8     class LuckyController extends AbstractController
9     {
10         #[Route('/porte-bonheur')]
11         public function makeMeHappy() : Response
12         {
13             $number = rand(0, 100);
14
15             return $this->render('base.html.twig', [
16                 'number' => $number,
17             ]);
18         }
19     }

```

p. 18 Solution n°10

Se rendre ensuite dans le dossier `/template` et modifier le fichier `base.html.twig`

```

1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta charset="UTF-8">
5         <title>{% block title %}Welcome!{% endblock %}</title>
6         <link rel="icon" href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22
viewBox=%220 128 128%22><text y=%221.2em%22 font-size=%2296%22>●</text></svg>"
7         {# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #}
8         {% block stylesheets %}
9             {{ encore_entry_link_tags('app') }}
10        {% endblock %}
11
12        {% block javascripts %}
13            {{ encore_entry_script_tags('app') }}
14        {% endblock %}
15    </head>
16
17    <body>
18
19        {% block body %}
20            Votre numéro porte-bonheur est le {{ number }}
21        {% endblock %}
22    </body>

```


23 </html>

Exercice p. 18 Solution n°11

Question 1

Composer est utile pour :


- ☐ Écrire le prochain tube de l'été
- ☒ Ajouter des dépendances à votre projet
- ☐ Compiler le code PHP

 *Composer* est un outil permettant d'ajouter des packages PHP à votre projet avec une simple ligne de commande. La liste des dépendances est alors tenue dans le fichier *composer.json*.

Question 2

Parmi ces frameworks, lequel n'est pas un framework PHP ?


- ☒ Express
- ☐ Laravel
- ☐ CodeIgniter

 En effet *Express* est un serveur pour Nodejs.

Question 3

Le moteur de template permettant de générer les vues de Symfony s'appelle **Flig**.


- ☐ Vrai
- ☒ Faux

 Non, le moteur de template utilisé par Symfony s'appelle **Twig**. Il permet d'injecter dans du code HTML des données issues de Symfony afin de préparer les pages pour l'affichage.

Question 4

Symfony doit obligatoirement tourner sur le port 8000.

- ☐ Vrai
- ☒ Faux

 C'est le port par défaut en lançant la commande `symfony server:start`, mais vous pouvez le configurer selon vos besoins ou les ports disponibles. En effet, il n'est pas rare que plusieurs applications tournent en même temps sur votre machine locale et il peut y avoir des conflits de ports.

Question 5

La méthode `render` est directement disponible dans les controllers.

- ☐ Vrai
- ☒ Faux

Q Non, il faudra faire étendre les controllers qui doivent rendre des vues avec la méthode `render` avec la classe `AbstractController`.