**Installation de Visual Studio Code**

Objet :

Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code et Git intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier et installer des extensions qui ajoutent des fonctionnalités supplémentaires.

Le lien pour le téléchargement :

<https://code.visualstudio.com/>

**Installation de MySQL/ PHP 7.4.12/ composer et Symfony sous Windows 10**

Aide à l’installation :

<https://www.grafikart.fr/tutoriels/windows-php-mysql-901>

Installation de MySQL :

Objet : Moteur de base de données

<https://dev.mysql.com/downloads/installer/>

* Sélectionnez le type custom
* Choisir le serveur approprié et si besoin prendre les applications comme MySQL Wordbench ou MySQL SHELL.
* Choisir la configuration standard
* Rentrez un mot de passe pour la base de données
* Finir l’installation

Une fois l’installation terminée, prendre le chemin d’accès de l’application pour donner l’autorisation à l’invite de commande.

* C:\Program Files\MySQL

Copiez de du chemin puis le coller dans le path.

* Propriété système / informations système / paramètre système avancés / variables d’environnements / sélection de path / copie du chemin d’accès.

Vérification avec le cmd en tapant

Mysql --version

Le numéro de version doit s’afficher.

Installation de PHP :

Objet : Langage de programmation des pages HTML

Allez sur le site ci-dessous :

<https://windows.php.net/download#php-7.4>

Une fois la version choisie, il faut la télécharger. Créez un dossier PHP dans programme file puis déplacez le contenu téléchargé dedans. Même chose que pour MySQL, il faut ajouter la commande au path en copiant le chemin d’accès.

* C:\Program Files\PHP

Voici l’accès au chemin du path :

* Propriété système / informations système / paramètre système avancés / variables d’environnements / sélection de path / copie du chemin d’accès.

Vérification avec le cmd en tapant

php --version

Le numéro de version doit s’afficher.

Installation de Composer :

Objet : Gestionnaire de dépendances libre écrit un PHP. Il permet de déclarer et d’installer des bibliothèques.

Allez sur le site ci-dessous :

<https://getcomposer.org/>

Cliquez sur l’exécutable puis suivre les instructions jusqu’au message d’erreur. A partir de ce moment, il faut retourner dans le dossier PHP et faire une copie du fichier php.ini-developpement au même endroit et le renommer php.ini.

Avec ce fichier, il faut modifier les droits d’utilisateurs.

* Clic droit
* Propriété
* Sécurité
* Administrateurs
* Cochez contrôle total
* Validez tous les choix

Ensuite, rentrez dans le fichier et modifiez les extensions suivantes en décommentant les points virgules devant. Voici la listes : environ vers les lignes 900.

* extension=fileinfo
* extension=gd2
* extension=gettext
* extension=curl
* extension=mbstring
* extension=mysqli
* extension=openssl
* extension=pdo\_mysql
* extension=pdo\_sqlite

Un peu plus au dessus, il faut décommenter :

* extension\_dir = "ext"

Reprenez l’installation de Composer et cliquez sur suivant.

Vérification avec le cmd en tapant

composer --version

Installation de Symfony :

Objet :

Symfony est un ensemble de composants PHP, un framework d'application Web, une philosophie et une communauté - tous travaillant ensemble en harmonie.

Symfony est un framework basé sur le modèle MVC (Model View Controller).

Pour faire simple, le modèle MVC va nous aider à séparer les requêtes de la base de données (Model) de la logique relative au traitement des demandes (Controller) et au rendu de la présentation (View).

Allez sur le site ci-dessous :

<https://symfony.com/doc/current/setup.html#creating-symfony-applications>

Créez un dossier où vous voulez pour pouvoir stocker le package Symfony. L’installation est nécessaire pour chaque projet. Ensuite, se rendre avec le cmd au dossier que vous avez créé.

* D:\sites\lab\demo

L’installation va se faire via Composer. Lancez le cmd puis copier la commande

**ATTENTION :** la commande préconisée ci-dessous ne fonctionne pas correctement.

composer create-project symfony/website-skeleton my\_project\_name

Il faut prendre celle-ci :

composer create-project symfony/skeleton my\_project\_name

Une fois l’installation terminée, pour que Symfony soit interprété par le cmd, il faudra télécharger le Symfony binaire depuis l’URL suivante.

<https://symfony.com/download>

Il va se télécharger dans le répertoire :

* C:\Program Files\Symfony

Maintenant, vous pouvez lancer le serveur depuis l’invite de commande. Attention, il faut bien se placer dans l’arborescence des dossiers pour ouvrir le fichier que l’on souhaite.

A partir de :

* D:\Symfony\lab\monProjet

Symfony server:start

Si le serveur fonctionne correctement, on peut le quitter en appuyant sur :

Ctrl + c

Installation de Symfony Flex :

Symfony Flex est un moyen d’installer et de gérer des applications Symfony. Il automatise les tâches les plus courantes des applications Symfony. Il a été lancer avec la version 4 de Symfony.

Voici la commande pour installer Symfony Flex :

composer require symfony/flex

Pour avoir un aperçu d’un site web configuré sous Symfony, la documentation en fin de page permet de télécharger un modèle complet.

Maintenant la documentation recommande d’utiliser Symfony Flex pour structurer correctement les répertoires. Voici la structure recommandée pour les répertoires :

your-project/

── assets/

── bin/

└── console

── config/

├── bundles. php

├── packages/

├── routes.yaml

└── services.yaml

── public/

└── index. php

── src/

├── ...

└── Kernel.php

── templates/

── tests/

── translations/

── var/

── vendor/

Voici le détail des fichiers que vous trouverez dans votre projet :

* bin: c’est ici que se trouve les exécutables de Symfony, comme le fichier console. Les fichiers dans ce dossier seront donc exécutables en ligne de commande.
* config: comme son nom l’indique, ce dossier va contenir la configuration de notre application Symfony. C’est ici que nous allons configurer les routes, les services, …
* public: Ceci est le dossier d’entré de notre application, mais aussi le dossier public, nous allons y mettre tout les fichiers accessibles au public. Il contient notamment le contrôleur frontal de Symfony.
* src: C’est ici que la magie s’opère, c’est ici que vous écrirez vos fichiers PHP. Les contrôleurs, entités, migrations, … sont dans ce dossier.
* templates: Les vues de notre application sont ici, toutes les pages qui vont être affichées à l'écran vont être ici.
* tests: C’est ici que vous allez mettre les différents tests pour tester vos fonctionnalités.
* var: Dans ce dossier Symfony va mettre les caches et logs
* vendor: Ce dossier contient toutes les dépendances créées avec composer.
* .env: Ce fichier définit les variables d’environnement de notre application, il définit l’environnement dans lequel nous sommes, développement ou production, les informations de connexion à la BDD, ...

Pour voir une liste globale des commandes, il suffit de taper dans le cmd :

php bin/console

Voici un peu de vocabulaire pour utiliser correctement Symfony :

Doctrine

Doctrine est ce qu’on appelle un ORM (Object Relational Mapper), pour faire simple, il va nous permettre de créer, modifier, supprimer ou récupérer des objets dans la base de données en utilisant des classes.

Contrôleur

L’un des composants de l’acronyme MVC, le contrôleur est une fonction PHP que nous allons créer et qui va permettre de lire une requête et renvoyer une réponse, c’est tout son fonctionnement, recevoir une requête et retourner une réponse.

Route

La route représente le lien entre une URL et un contrôleur. Quand le serveur reçoit une requête HTTP, symfony regarde dans le fichier config/routes.yaml pour trouver la route, et cette route va définir le contrôleur a appelé dans ce cas.

**Création d’un espace de travail sous VSC.**

Objet :

Un espace de travail est une collection virtuelle de dossiers ouverts simultanément dans VSCode et définis dans un .code-workspacefichier. L'ouverture de ce fichier ouvrira automatiquement la collection de dossiers. C'est ce qu'on appelle un espace de travail "multi-racine".

Procédure :

Dans la barre de menu \* → *Fichier* → *Ouvrir le dossier ...* . Accédez à l'emplacement où vous souhaitez placer votre dossier et cliquez avec le bouton droit pour ouvrir un nouveau dossier. Nommez-le comme vous voulez, puis cliquez sur *"Sélectionner un dossier"* . Il apparaîtra dans \* l'explorateur de code Visual Studio.

Maintenant , dans le menu *Fichier* → *Enregistrer sous ... Espace de travail* . Nommez l'espace de travail et enregistrez-le où vous voulez conserver tous vos espaces de travail (pas nécessairement où se trouvent vos dossiers de projet). J'ai mis tous les miens dans un dossier appelé "espace de travail Visual Studio Code".

Il sera enregistré en tant que .code-workspacefichier ( ) et n'est qu'un index de tous les fichiers et dossiers qu'il contient (ou pointe vers) où qu'ils se trouvent sur votre disque dur.

Quelques extensions qui vont être utiles dans Visual Studio Code :

**Live server :** Permet d’ouvrir le navigateur en un clic via le localhost.

**Symfony for VSCode :** Cette extension vise à aider au développement de projets Symfony2 +, en fournissant une autocomplétion et une visualisation basiques du conteneur Symfony.

**Xdebug :** outil de débug pour le langage PHP via VSCode.

Procédure : Dans le cmd, il faut rentrer la commande :

Php -i

Ensuite il faut copier les informations que l’on a fait apparaitre pour les copier sur le site internet de Xdebug.

<https://xdebug.org/wizard>

Le résultat va nous proposer de télécharger l’extension adéquate.

Une fois l’extension téléchargée, il suffit de la mettre à la racine du dossier PHP.

* C:\Program Files\PHP

Ensuite, nous devons modifier quelques lignes dans le fichier php.ini.

Voici les lignes que nous devons rajouter à la fin des extensions aux alentours des lignes 930.

zend\_extension = C:\Program Files\PHP\ext\php\_xdebug-3.0.0-7.4-vc15-nts-x86\_64.dll

[XDebug]

xdebug.remote\_enable = 1 \* pour activer le débogueur

xdebug.remote\_autostart = 1 \* pour la mise en route automatique

Afin d'utiliser Xdebug dans Visual Studio Code, nous allons installer l'extension PHP Debug.

Une fois l'extension installée, nous allons nous rendre dans la partie "Debug" de VS Code Debug

Dans cette partie, la 1ère fois que nous y accédons dans un projet, VS Code nous propose de créer un fichier "launch.json", ce que nous allons faire.

Il va nous être proposé de choisir avec quel langage nous souhaiterions travailler. Il faudra prendre PHP.

Il suffit maintenant de lancer l’extension.

**Création d’un simple site sous Symfony 5**

Objet :

Partir de quelques fichiers puis les ajouter au fur et à mesure du besoin. Voici le lien qui permet de créer tout ce qui suit :

<https://www.youtube.com/watch?v=yCPiX7_fy30>

Nous allons télécharger une version plus légère de Symfony et ajouter au fur et à mesure les extensions que l’on va avoir besoin.

symfony new hello-city

Commande simplifiée pour se connecter au serveur en fond (cmd toujours utilisable) :

symfony serve -d

Commande pour lancer le serveur via le navigateur directement :

symfony open:local

Commande pour lancer le maker-bundle. C’est pour pouvoir créer des fichiers par la suite (--dev pour l’utiliser par la suite directement en développement).

composer req maker --dev

Nous pouvons à présent créer un contrôleur (ici, pages sera le nom du contrôleur) :

symfony console make:controller pages

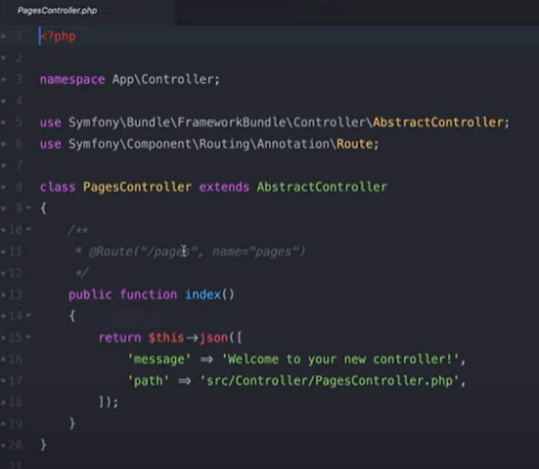
Petite erreur, on nous demande de créer avant doctrine/annotations. Voici la commande :

Composer require doctrine/annotations

Nous pouvons reprendre la commande pour créer le contrôleur.

Une fois le fichier créé, il faudra le modifier pour ensuite lui donner une route. Le voici avant

Avant Après modification

Nous créons en mêm temps le chemin pour aller chercher le fichier twig dans le dossier template. Mais il n’existe pas encore. Il faut taper dans l’invite de commande :

Composer req twig-bundle

Le fichier base.html.twig a été créé dans le dossier templates. Il faut créer un nouveau fichier .twig dans un dossier nommé « pages » pour pouvoir afficher une nouvelle page. Il s’appelera [home.html.twig](http://home.html.twig). Il faudra le relier à la base.html.twig comme montré ci-dessus.



Pour ajouter une seconde page, il faut ajouter une route dans le contrôleur et la fonction about comme indiqué ci-dessous.

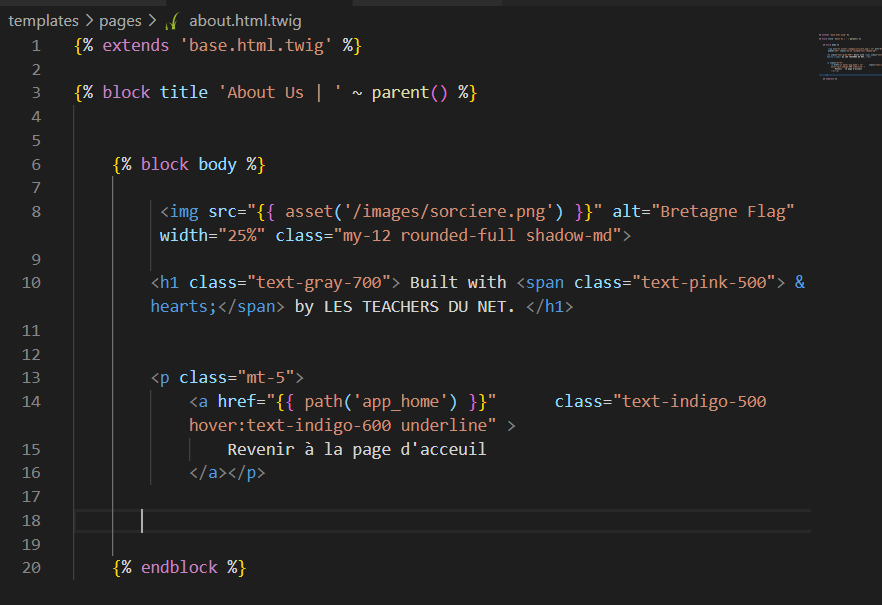


Maintenant, tout comme l’autre route, il lui faut un fichier .twig pour exister. Création du fichier twig en enregistrant sous un nouveau fichier nommé about.html.twig .

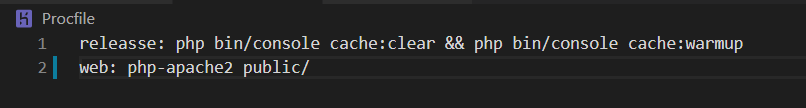
La suite sera surtout de la mise en page avec l’aide du framework tailwindcss.com.

<https://tailwindcss.com/>

Le résultat de la page [home.html.twig](http://home.html.twig) est un peu plus au dessus. Le résultat du fichier about.html.twig va se trouver ci-dessous.



Pour mettre en production, il faut créer un fichier à la racine du projet et le nommé « Procfile » sans aucune extension. Insérez le code suivant à l’intérieur :



La première ligne veut dire que le cache va être vidé automatiquement lors de la requête. La seconde ligne de code signifie qu’il va se connecter au serveur web en php sous apache2 depuis le dossier public.

La dernière étape sera d’installer apache pour obtenir le fichier htaccess afin que le site affiche uniquement ce qui se trouve en public. Voici la ligne de commande pour installer le package apache2 :

Composer req apache-pack

Le fichier htaccess se génère automatiquement dans la dossier public.

Petit soucis au niveau du déploiement, le site fonctionne correctement sous un serveur d’applications cloud mais ne fonctionne pas vers un hébergeur classique. Il faut rajouter 2 fichiers htaccess.

Il faut 3 fichiers ".htaccess" pour rediriger vers les sites créés sous Symfony et placés sous "www/symfony".

Premier fichier ".htaccess" : sous le répertoire d’entrée de l’URL, indiqué par l’hébergeur ("www" chez Online.net) :

<IfModule mod\_rewrite.c>

# redirection "www-symfony/public/"

RewriteEngine On

RewriteCond %{REQUEST\_URI} !^/www-symfony/public/

RewriteRule ^(.\*)$ /www-symfony/public/$1 [L]

</IfModule>

Second fichier ".htaccess" : sous le répertoire de dépôt du site crée, pour le site Symfony, "www/www-symfony/" :

<IfModule mod\_rewrite.c>

RewriteCond %{REQUEST\_URI} !^/public/

RewriteRule ^(.\*)$ /public/$1 [L]

</IfModule>

Troisième fichier ".htaccess" : sous l’arborescence du framework (et généralement généré automatiquement par celui-ci), c’est-à-dire pour Symfony, "www/www-symfony/public" :

# Use the front controller as index file. It serves as a fallback solution when

# every other rewrite/redirect fails (e.g. in an aliased environment without

# mod\_rewrite). Additionally, this reduces the matching process for the

# start page (path "/") because otherwise Apache will apply the rewriting rules

# to each configured DirectoryIndex file (e.g. index.php, index.html, index.pl).

DirectoryIndex index.php

# By default, Apache does not evaluate symbolic links if you did not enable this

# feature in your server configuration. Uncomment the following line if you

# install assets as symlinks or if you experience problems related to symlinks

# when compiling LESS/Sass/CoffeScript assets.

# Options +FollowSymlinks

# Disabling MultiViews prevents unwanted negotiation, e.g. "/index" should not resolve

# to the front controller "/index.php" but be rewritten to "/index.php/index".

<IfModule mod\_negotiation.c>

Options -MultiViews

</IfModule>

<IfModule mod\_rewrite.c>

RewriteEngine On

# Determine the RewriteBase automatically and set it as environment variable.

# If you are using Apache aliases to do mass virtual hosting or installed the

# project in a subdirectory, the base path will be prepended to allow proper

# resolution of the index.php file and to redirect to the correct URI. It will

# work in environments without path prefix as well, providing a safe, one-size

# fits all solution. But as you do not need it in this case, you can comment

# the following 2 lines to eliminate the overhead.

RewriteCond %{REQUEST\_URI}::$0 ^(/.+)/(.\*)::\2$

RewriteRule .\* - [E=BASE:%1]

# Sets the HTTP\_AUTHORIZATION header removed by Apache

RewriteCond %{HTTP:Authorization} .+

RewriteRule ^ - [E=HTTP\_AUTHORIZATION:%0]

# Redirect to URI without front controller to prevent duplicate content

# (with and without `/index.php`). Only do this redirect on the initial

# rewrite by Apache and not on subsequent cycles. Otherwise we would get an

# endless redirect loop (request -> rewrite to front controller ->

# redirect -> request -> ...).

# So in case you get a "too many redirects" error or you always get redirected

# to the start page because your Apache does not expose the REDIRECT\_STATUS

# environment variable, you have 2 choices:

# - disable this feature by commenting the following 2 lines or

# - use Apache >= 2.3.9 and replace all L flags by END flags and remove the

# following RewriteCond (best solution)

RewriteCond %{ENV:REDIRECT\_STATUS} =""

RewriteRule ^index\.php(?:/(.\*)|$) %{ENV:BASE}/$1 [R=301,L]

# If the requested filename exists, simply serve it.

# We only want to let Apache serve files and not directories.

# Rewrite all other queries to the front controller.

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^ %{ENV:BASE}/index.php [L]

</IfModule>

<IfModule !mod\_rewrite.c>

<IfModule mod\_alias.c>

# When mod\_rewrite is not available, we instruct a temporary redirect of

# the start page to the front controller explicitly so that the website

# and the generated links can still be used.

RedirectMatch 307 ^/$ /index.php/

# RedirectTemp cannot be used instead

</IfModule>

</IfModule>

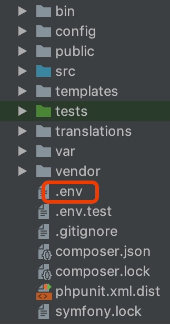
**Basculement du mode développement au mode production sur Symfony**

Objet :

Sur Symfony en mode développement il y a des informations confidentiels qui apparaissent et qui ne concernent que le développeur et non les utilisateurs. Il y a par exemple les messages d'erreurs et la barre de debug. Pour les faire disparaître avant de mettre en ligne un projet Symfony, il est important de changer et de passer en mode production (sans message d'erreur et sans barre de debug).

**Étapes pour passer au mode production**

**1** Ouvrez le fichier ".env" à la racine du projet de Symfony.



À la ligne 17 le fichier se présente en mode développement sous la forme suivante.

APP\_ENV=dev

**2** Modifier le code en passant le projet en mode production pour vos utilisateurs.

APP\_ENV=prod

APP\_DEBUG=0

Pour information, rajouter "APP\_DEBUG=0" permet d'être sûre de ne plus recevoir les messages d'erreurs.

Déploiement vers le portail à partir de Visual Studio Code :

Objet : Lancer un projet local sur un serveur. C’est-à-dire déplacer les fichiers di site web depuis le pc de développement vers le portail public.

Vidéo sur les extensions et sur le déploiement :

<https://www.youtube.com/watch?v=dc21bhbnrZk>

**Deploy :** extension pour déployer le projet.

Cet outil prépare le déploiement pour que les fichiers soient ensuite transférés vers un outil tel Filezilla.

<https://github.com/mkloubert/vscode-deploy-reloaded/wiki>

Ouvrez (ou créez) votre fichier de settings.json dans votre .vscode, sous dossier de votre espace de travail.

Ajouter une deploy.reloadedsection dans le dossier setting.json.

Exemple pour le protocole SFTP.

{

" deploy.reloaded " : {

" packages " : [

{

" name " : " Version 2.3.4 " ,

" description " : " Package version 2.3.4 " ,

" fichiers " : [

" \*\* / \*. php " ,

" /\*.json "

],

" exclure " : [

" tests / \*\* "

],

" deployOnSave " : [ " Mon dossier SFTP " ]

}

],

" cibles " : [

{

" type " : " sftp " ,

" name " : " Mon dossier SFTP " ,

" description " : " Un dossier SFTP " ,

 " host " : " localhost " , " port " : 22 **, //url du serveur**

                 " user " : " tester " , " password " : " password " **,// identifiant de connexion**

                " dir " : " / my\_package\_files **"// chemin dans lequel on veut copier le dossier**

           }

}

]

}

}

Afin de ne pas surcharger le serveur, nous pouvons effacer le cache pour que les fichiers temporaires soient effacés. Voici la commande qui est utile :

php bin/console cache:clear

**Préparation à la lecture du livre « En route pour Symfony 5 » écrit par Fabien Potencier**

**Téléchargement de Hyper :**

Objet :

Hyper est un terminal à base d'électrons, il est construit sur HTML / CSS / JS.

<https://hyper.is/>

Vérification des installations sur PHP :

Php -m

Les extensions nécessaires pour pouvoir utiliser Symfony sont :

* pdo\_pgsql
* xsl
* amqp
* gd
* openssl
* sodium
* redis
* curl

Voici la page où l’on peut se procurer les extensions manquantes pour PHP. La procédure qui suit est la même pour toutes les extensions que l’on souhaite installer.

<https://pecl.php.net/package/redis/5.3.2/windows>

Une fois le téléchargement terminé, il faut suivre la procédure suivante :

Lien utile pour l’installation :

<https://stackoverflow.com/questions/13817714/how-to-install-amqp-on-windows>

* Copiez php\_amqp.dll dans votre dossier php ext
* Mettez à jour votre php.ini avec: extension = php\_amqp.dll
* Copiez rabbitmq.4.dll dans votre dossier SysWOW64.

Pour vérifier que l’on a tout ce qu’il faut pour travailler sur Symfony, tapez la commande suivante :

symfony book:check-requirements

**Installation de Docker :**

Objet :

Docker est une plate-forme ouverte pour le développement, l'expédition et l'exécution d'applications. Docker vous permet de séparer vos applications de votre infrastructure afin de pouvoir livrer rapidement des logiciels. Avec Docker, vous pouvez gérer votre infrastructure de la même manière que vous gérez vos applications. En tirant parti des méthodologies de Docker pour expédier, tester et déployer rapidement le code, vous pouvez réduire considérablement le délai entre l'écriture du code et son exécution en production.

<https://hub.docker.com/editions/community/docker-ce-desktop-windows/>

Suivre les instructions…

Il faut se créer un compte sur dockerhub.

Commande pour démarrer docker :

docker run -d -p 80:80 docker/getting-started

Qu’est-ce qu’un conteneur :

Un conteneur est simplement un autre processus sur votre machine qui a été isolé de tous les autres processus sur la machine hôte. Cet isolement tire parti des espaces de noms et des groupes de contrôle du noyau , des fonctionnalités qui existent sous Linux depuis longtemps. Docker a travaillé pour rendre ces fonctionnalités accessibles et faciles à utiliser.

Exécution de l’application :

Téléchargez le zip sur la page suivante :

<http://localhost/tutorial/our-application/>

Une fois extrait, utilisez votre éditeur de code préféré pour ouvrir le projet. Si vous avez besoin d'un éditeur, vous pouvez utiliser Visual Studio Code . Vous devriez voir les package.jsonsous-répertoires et deux ( srcet spec).

Créez un fichier Dockerfile pour y insérer le code suivant :

FROM node:12-alpine

WORKDIR /app

COPY . .

RUN yarn install --production

CMD ["node", "src/index.js"]

Récupération du code source du projet Symfony :

symfony new –version=5.0-6 –book guestbook

Commande utile :

À partir du répertoire du projet, démarrez le serveur web en arrière-plan (option -d)

symfony server:start -d

**Rapatriement d’un site depuis un ordinateur de développement vers un nouveau pc**

Préparation de la configuration de Symfony pour l’accueil du site sur le pc de développement.

Création d’un dossier TEKXIA pour y intégrer le dossier original et le dossier à travailler.

Préparer le contexte

Chemin pour le site d’origine :

D:\Tekxia\Tekxia-origine

Nom du chemin pour le dossier à reprendre :

D:\Tekxia\Tekxia-VSC\symf-txa-com

**Nouveau projet symfony**

Symfony new projet -full

Composer require symfony/webpack-encore-bundle

npm install

Seconde période de stage

Commande pour installer le certificat sous symfony est

Symfony server:ca:install

Git

**Création d’un nouveau repo**

Sur le compte github, il faut aller dans Repositories et cliquer sur nouveau

Allez sur le dossier racine du projet

Echo “#Entrainements” >> README.md

Git init

Git add README.md

Git commit -m “first commit”

Git branch -M main

Git remote add origin <https://github.com/jpicard22/> nom du projet.git

Git push -u origin main

Pour ajouter des fichiers

Git add nom du fichier

Pour commiter et ajouter

Git status

Git add .

Git commit -m “modification”

Git push

Pour créer une branche

Git branch nom de la branche

Savoir dans quelle branche je suis

Git branch

Pour changer de branche

Git checkout nom de la branche

Merger avec une branche ( Les modifications apportées dans la branche vous conviennent. Il faut alors fusionner votre branche et votre master. Placer vous dans la branche master )

Git checkout master

Puis

Git merge nom de la branche

Pour effacer la branche inutile

Git branch -D nom de la branche

Pour récupérer les dernières modifications

Git pull origin master

MP-NET-S5

Symfony new MP-NET-S5-2 -full

Composer require symfony/webpack-encore-bundle

npm install

composer req maker --dev

Composer require doctrine/annotations

symfony console make:controller home

composer require symfony/asset

**Installation de yarn**

npm install --global yarn

yarn add --dev @symfony/webpack-encore

yarn encore dev

Pour créer le manifest.json

yarn encore dev-server –hot

Mise en place de la home page avec un carousel. Le site ci-dessous donne un très bon exemple de carousel

<https://codepen.io/Ziratsu/pen/XGbVVL>

**Pour créer le formulaire :**

Pour créer le fichier security.yaml

composer require security

Création de la base de données :

php bin/console doctrine:database:create

php bin/console make:entity

Création de la table u\_iden en ligne de commande

Validation via les commandes symfony

php bin/console make:migration

php bin/console doctrine:migrations:migrate

Création du Controller user :

symfony console make:controller user

Création de l’entité user

symfony console make:user

Pour mettre en place l’authentification des utilisateurs

symfony console make:auth

Création du formulaire d’inscription

symfony console make:registration-form

Pour vérifier les comptes par e-mail, il faut installer le bundle « verify e-mail »

composer require symfonycasts/verify-email-bundle

Pour la réinitialisation des mots de passe, il faut installer le bundle reset-password

composer require symfonycasts/reset-password-bundle

symfony console make:reset-password

Il faut mettre à jour la base de données

symfony console make:migration

symfony console doctrine:migrations:migrate

Mettre à jour la table user en rajoutant le nom et le prénom

symfony console make:entity + ajout du nom et prénom via le terminal de commande

Mise à jour de la base de données

symfony console make:migration

symfony console doctrine:migrations:migrate

Dans le fichier security.yaml, il faut décocher la route de l’utilisateur

access\_control:

# - { path: ^/admin, roles: ROLE\_ADMIN }

- { path: ^/users, roles: ROLE\_USER }

La route users est protégée

Modifier dans le contrôleur usersController -> ajouter la méthode

/\*\*

\* @Route("/users", name="users")

\*/

public function index()

{

return $this->render('users/index.html.twig');

}

Mise en place de la vue dans le fichier twig que l’on veut faire apparaitre les données de la base

<p>Nom : {{ app.user.name }}</p>

<p>Prénom : {{ app.user.firstname }}</p>

<p>E-mail : {{ app.user.email }}</p>

Mettre en place Symfony form

composer require symfony/form

namespace App\Controller\Admin

Pour créer l’interface administrateur

Changer en haut au niveau du namespace

namespace App\Controller\Admin

Création du controller admin

symfony console make:controller admin

Décocher dans le fichier security.yaml -> # path admin

symfony console make:form ReCaptchaType

PATH=$PATH:/path/to/pg/bin pgmgr db create