INSTALLER ET CONFIGURER SYMFONY

DANS LE TERMINAL

- **Installation de symfony (sous windows)**

  $ scoop install symfony-cli

scoop : nom de l'outil de ligne de commande utilisé pour installer des programmes. Scoop est un gestionnaire de paquets pour Windows, qui simplifie l'installation et la gestion des applications sans nécessiter des droits administratifs ou des modifications du système.

install : commande utilisée avec Scoop pour indiquer que vous souhaitez installer un programme. Cela dit à Scoop de télécharger et d'installer un paquet spécifique.

symfony-cli : nom du paquet que vous souhaitez installer avec Scoop. symfony-cli se réfère à l'interface de ligne de commande de Symfony, un framework populaire pour construire des applications web en PHP. L'outil CLI permet aux développeurs de démarrer et gérer leurs projets Symfony, d'exécuter des tâches de développement, et plus encore.

- **Installation de symfony (sous fedora)**

  $ curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.rpm.sh' | sudo -E bash

  $ sudo dnf install symfony-cli

Commande 1: curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.rpm.sh' | sudo -E bash

curl : C'est un outil de ligne de commande utilisé pour transférer des données à partir ou vers un serveur en utilisant divers protocoles. Ici, il est utilisé pour télécharger un script depuis Internet.

-1sLf : Ce sont des options utilisées avec curl :

-1 : Force curl à utiliser le protocole SSL/TLS pour la sécurité (probablement une erreur de frappe, et devrait être -k pour permettre les connexions non sécurisées ou -1 pour utiliser uniquement TLS 1.x).

-s : Mode silencieux. Ne montre pas de barre de progression ni d'erreur.

-L : Si la requête retourne une redirection, curl suivra cette redirection.

-f : Échoue silencieusement (sans sortie d'erreur) sur les codes d'erreur HTTP.

'https://dl.cloudsmith.io/public/symfony/stable/setup.rpm.sh' : URL d'où curl télécharge le script. Ce script est destiné à configurer un dépôt pour les paquets RPM.

| : Un "pipe". Il prend la sortie de curl et la redirige vers une autre commande.

sudo : Exécute la commande suivante avec des privilèges d'administrateur.

-E : Préserve l'environnement lors de l'exécution de la commande suivante. Cela signifie que les variables d'environnement actuelles seront disponibles dans le script exécuté.

bash : Exécute le script téléchargé par curl en utilisant le shell Bash.

Commande 2: sudo dnf install symfony-cli

sudo : Exécute la commande suivante avec des privilèges d'administrateur.

dnf : Le gestionnaire de paquets utilisé par Fedora et certaines autres distributions basées sur Red Hat. Utilisé ici pour installer des logiciels.

install : Indique à dnf de procéder à l'installation d'un ou plusieurs paquets.

symfony-cli : Le nom du paquet à installer. Symfony CLI est un outil de ligne de commande pour gérer des applications Symfony.

En résumé, la première commande configure un dépôt pour les logiciels et la seconde install symfony-cli à partir de ce dépôt.

- **Création d'un nouveau projet symfony**

symfony new « le nom du projet » --webapp

(ex avec un projet qui s'appelle easyC, cela donne symfony new easyC –webpack)

- **Ouverture du projet avec vs code**

$ cd easyC (se mettre dans le main (dossier easyC))

$ code . (ouverture vs code)

- **Installation du bundle Webpack Encore dans le projet Symfony (ce qui permet de gérer les assets plus facilement et efficacement)**

  $ composer require symfony/webpack-encore-bundle

composer : C'est le gestionnaire de dépendances pour PHP. Il est utilisé pour installer et gérer les bibliothèques et les packages PHP dans un projet.

require : C'est la commande pour demander à Composer d'installer un nouveau package.

symfony/webpack-encore-bundle : C'est le nom du package à installer. Il s'agit du bundle **Webpack Encore** pour Symfony. Webpack Encore est un outil utilisé pour gérer les assets (JavaScript, CSS, etc.) dans les projets Symfony de manière moderne et efficace.

- **Installation automatique de toutes les dépendances répertoriées dans le fichier package.json du projet, ainsi que leurs dépendances, en créant un dossier nommé "node_modules".**

  $ npm install

  $ npm run dev

  npm : C'est le gestionnaire de paquets pour JavaScript. Il est utilisé pour installer et gérer les dépendances d'un projet Node.js.

  install : C'est la commande pour installer les dépendances définies dans le fichier package.json de votre projet.

- **Installation de Bootstrap**

  $ npm install bootstrap --save -dev

  npm : Comme précédemment, c'est le gestionnaire de paquets pour JavaScript.

  install : C'est la commande pour installer des paquets.

  bootstrap : C'est le nom du package à installer. Bootstrap est un framework front-end populaire pour la conception de sites Web et d'applications Web.

  --save-dev : c'est un drapeau qui indique à npm d'ajouter Bootstrap en tant que dépendance de développement dans le fichier package.json. Cela signifie que Bootstrap est nécessaire uniquement pour le développement du projet, pas pour son exécution en production.

  En résumé, cette commande installe Bootstrap dans le projet JavaScript, le déclarant comme une dépendance de développement. Cela permettra d'utiliser les fonctionnalités de Bootstrap lors du développement de l'application.

- **Installation de jQuery et Popper.js dans le projet JavaScript, les ajoutant en tant que dépendances de développement.**

  $ npm install jquery @popperjs/core --save -dev

  Cette commande est utilisée dans les projets JavaScript, en particulier avec Node.js. Voici ce qu'elle fait :

  npm : C'est le gestionnaire de paquets pour JavaScript. Il est utilisé pour installer et gérer les dépendances d'un projet.

  install : C'est la commande pour installer des paquets. On lui indique les paquets à installer.

jquery : C'est le nom du premier paquet à installer. jQuery est une bibliothèque JavaScript populaire utilisée pour simplifier la manipulation du DOM et l'interaction avec celui-ci.

@popperjs/core : C'est le nom du deuxième paquet à installer. Popper.js est une bibliothèque JavaScript utilisée pour gérer le positionnement de popups et de tooltips dans une interface utilisateur.

--save-dev : C'est un drapeau indiquant à npm d'ajouter ces paquets en tant que dépendances de développement dans le fichier package.json. Les dépendances de développement sont des paquets nécessaires uniquement pour le développement du projet, pas pour son exécution en production.

DANS LES FICHIERS

- Dans le **dossier style** créer un fichier **global.scss** **et y importer ce code** :

```
// assets/styles/global.scss
// customize some Bootstrap variables
$primary: darken(#428bca, 20%);

// the ~ allows you to reference things in node_modules
@import "~bootstrap/scss/bootstrap";
```

- Ensuite, aller dans le **dossier assets** puis dans le fichier **app.js,** **ce code doit figurer** :

```
// app.js
import './bootstrap.js';
/*
* Welcome to your app's main JavaScript file!
*
* This file will be included onto the page via the importmap() Twig function,
* which should already be in your base.html.twig.
*/
import './styles/global.css';
// app.js
const $ = require('jquery');
// this "modifies" the jquery module: adding behavior to it
// the bootstrap module doesn't export/return anything
require('bootstrap');
```

```
// or you can include specific pieces
// require('bootstrap/js/dist/tooltip');
// require('bootstrap/js/dist/popover');
$(document).ready(function() {
$('[data-toggle="popover"]').popover();
});
```

- Dans le fichier app.js, **changer à la ligne 8 le «app.css» pour «global.css»** :
  import './styles/global.css';

- Aller dans le **fichier bootstrap.js et mettre en commentaire le contenu total de la page**

- Aller dans le fichier **webpack.config.js (qui est dans la racine) et « DECOMMENTER » la ligne 57 (comme ci-dessous) :**
  .enableSassLoader()

  DANS LE TERMINAL

- **Installation de Sass et Sass-loader**

  $ npm install sass-loader@^14.0.0 sass --save -dev

  npm : Comme d'habitude, c'est le gestionnaire de paquets pour JavaScript.

  install : C'est la commande pour installer des paquets.

  sass-loader@^14.0.0 : C'est le nom du premier package à installer. sass-loader est un loader pour Webpack, un outil de build pour JavaScript, qui permet de compiler les fichiers Sass en CSS lors de la construction de votre application. L'indication "^14.0.0" signifie que vous souhaitez installer la version 14.0.0 ou une version ultérieure, mais pas la 15.0.0 ou toute autre version majeure ultérieure.

  sass : C'est le nom du deuxième package à installer. Sass est un préprocesseur CSS qui ajoute des fonctionnalités comme les variables, les mixins et les fonctions à CSS.

  --save -dev : Encore une fois, c'est un drapeau qui indique à npm d'ajouter sass-loader et Sass en tant que dépendances de développement dans le fichier package.json. Cela signifie qu'ils sont nécessaires uniquement pour le développement du projet, pas pour son exécution en production.

  En résumé, cette commande installe sass-loader et Sass dans votre projet JavaScript, les déclarant comme des dépendances de développement. Cela vous permettra d'utiliser Sass

pour écrire du CSS de manière plus efficace et de compiler les fichiers Sass en CSS à l'aide de Webpack.

DANS LES FICHIERS

- **Créer un fichier .env.local à la racine** du projet

- Toujours dans le fichier .env **décommenter la ligne 27** # DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8.0.32&charset=utf8mb4"

- Puis **commenter la ligne 29** # DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"

- Puis **copier la ligne 27 (ligne indiquée plus haut) et le coller dans le fichier .env.local**

  Puis modifier **uniquement** dans le fichier .env.local comme ceci :

  DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8.0.32&charset=utf8mb4"

  app = nom d'utilisateur de PhPMyAdmin (root)

  !ChangeMe! = password utilisateur de php my admin

  Puis remplacer le app par le nom du dossier (nouveau dossier cree avec symfony)

DANS LE TERMINAL

- **Créer une base de donnée**

  $ symfony console d:d:c (création de la database dans php my admin)

- **Créer des controller**

  $ symfony console make:controller homeController

  $ symfony console make:controller ProfilController

- **Installer un certificat SSL**

$ symfony server:ca:install

La commande "symfony server:ca:install" est utilisée dans le framework Symfony pour installer un certificat SSL local pour le serveur de développement intégré :

symfony : C'est le nom de la commande de la ligne de commande Symfony CLI.

server:ca:install : C'est la sous-commande spécifique de Symfony CLI qui gère l'installation du certificat SSL local pour le serveur de développement.

En résumé, cette commande permet d'installer un certificat SSL local pour le serveur de développement Symfony, ce qui permet d'accéder à l'application via HTTPS lors du développement local. Cela peut être utile pour simuler un environnement de production sécurisé lors du développement et du test de votre application Symfony.

DANS LES FICHIERS

- Aller dans le fichier homecontroller.php **(dossier : src/controller)** et **effacer le mot « home » dans cette ligne** :

  #[Route('/home', name: 'app_home')]

DANS LE TERMINAL

- C**réation d'une entity (c'est-à-dire une table dans une base de donnée) spécifique de symfony (user)**

  $ symfony console make:user User

  $ symfony console make:migration

  $ symfony console doctrine:migrations:migrate

Ces trois commandes sont utilisées dans le framework Symfony pour gérer les utilisateurs et les migrations de base de données :

symfony console make:user User : Cette commande génère automatiquement le code nécessaire pour créer une nouvelle classe d'utilisateur Symfony. Elle crée une classe User dans le répertoire de votre choix (généralement src/Entity ou src/Entity/User) avec les propriétés et méthodes de base d'un utilisateur. Cela facilite grandement la gestion des utilisateurs dans votre application Symfony.

symfony console make:migration : Cette commande génère automatiquement une migration basée sur les différences détectées entre vos entités Symfony et votre base de données. Une migration est un fichier PHP qui contient les instructions pour mettre à jour votre base de données afin de refléter les changements dans vos entités. Cela simplifie le

processus de mise à jour de votre schéma de base de données tout en maintenant la cohérence avec votre code.

symfony console doctrine:migrations:migrate : Cette commande exécute les migrations en attente sur votre base de données. Elle applique toutes les migrations qui n'ont pas encore été exécutées, mettant ainsi à jour votre base de données avec les derniers changements définis dans vos migrations. Cela garantit que votre base de données est toujours synchronisée avec votre code.

- **dans le terminal de commande** :

  $ composer require symfonycasts/verify-email-bundle

composer : C'est le gestionnaire de dépendances pour PHP.

require : C'est la commande pour demander à Composer d'installer un nouveau package.

symfonycasts/verify-email-bundle : C'est le nom du package à installer. VerifyEmailBundle est un bundle Symfony développé par SymfonyCasts qui facilite la mise en place de la vérification par e-mail dans les applications Symfony. Il fournit des fonctionnalités pour gérer le processus de vérification des adresses e-mail des utilisateurs, y compris la génération de liens de vérification, la validation des adresses e-mail et la gestion des événements liés à la vérification par e-mail.

En résumé, cette commande installe VerifyEmailBundle dans votre projet Symfony, vous permettant de mettre en place facilement la vérification par e-mail pour les utilisateurs de votre application Symfony. Cela simplifie le processus de gestion des adresses e-mail vérifiées et améliore la sécurité de votre application en confirmant les identités des utilisateurs.

- **Création d'un formulaire d'inscription**

  $ symfony console make:registration-form (Répondre à toutes les questions dans le terminal et à la question suivante «What route should the user be redirected to after registration?»: répondre no 15)

  $ symfony console make :migration

  $ symfony console doctrine :migrations :migrate

make:registration-form : C'est la commande spécifique pour générer un formulaire d'inscription. Elle fait partie du bundle MakerBundle de Symfony, qui est un ensemble d'outils pour accélérer le processus de développement en générant automatiquement du code boilerplate.

Lorsque vous exécutez cette commande, Symfony va vous guider à travers un assistant pour configurer les options du formulaire d'inscription, comme les champs à inclure (par exemple, nom, e-mail, mot de passe), les validations à appliquer, etc. Une fois terminé, Symfony générera automatiquement les fichiers nécessaires pour le formulaire d'inscription, y compris la classe du formulaire, le template Twig associé et les tests unitaires, si vous en avez activé la génération.

En résumé, cette commande simplifie le processus de création d'un formulaire d'inscription dans votre application Symfony en générant automatiquement le code boilerplate nécessaire, ce qui vous permet de vous concentrer sur le développement des fonctionnalités de votre application.

- **Envoyer des mails** :

  $ composer require symfony/google-mailer (pour Gmail)

  $ composer require symfony/google-mailer (autre)

composer : C'est le gestionnaire de dépendances pour PHP.

require : C'est la commande pour demander à Composer d'installer un nouveau package.

symfony/google-mailer : C'est le nom du package à installer. GoogleMailer est un composant Symfony qui fournit une intégration facile avec les services de messagerie de Google, tels que Gmail. Il permet d'envoyer des e-mails via le protocole SMTP de Google.

En résumé, cette commande installe le composant GoogleMailer dans votre projet Symfony, vous permettant d'envoyer des e-mails via les services de messagerie de Google de manière facile et intégrée. Cela peut être utile si vous avez besoin d'envoyer des e-mails depuis votre application Symfony via Gmail ou d'autres services de messagerie de Google.

DANS LES FICHIERS

(en amont Il va faloir cree un mail avec une authantification a deux facteur et generer un code .)

- **Aller dans le fichier .env (tout en bas) copier puis coller le bloc ci-dessous (ou le block adapter au server mail*)** dans le fichier .env.local :

  ###> symfony/google-mailer ###

  # Gmail SHOULD NOT be used on production, use it in development only.

  # MAILER_DSN=gmail://MAIL:PASSWORD @default

  ###< symfony/google-mailer ###

  *Le cas échant ne pas oublier de changer le port adapter a son serveur mail

- **Remplacer son adresse mail et le code généré par google** comme ci-dessous :

  # MAILER_DSN=gmail://robbianoremi@gmail.com:qgpasfrwcfhklgts@default

- Puis **coller et décommenter cette ligne dans le fichier .env.local**

DANS LE TERMINAL

$ symfony console make:auth

(Puis rp au question)

Donner la réponse ci-dessous à cette question «The class name of the authenticator to create (e.g. AppCustomAuthenticator)»:

Rp : > AppAuthenticator

$ symfony list

$ symfony console list

symfony console make:auth : Cette commande est utilisée pour générer rapidement des fonctionnalités d'authentification dans une application Symfony. Elle fait partie du bundle MakerBundle de Symfony. Lorsque vous exécutez cette commande, Symfony vous guide à travers un assistant interactif pour choisir les fonctionnalités d'authentification à inclure, telles que l'authentification par formulaire, l'authentification par API, etc. Elle génère ensuite automatiquement le code nécessaire, y compris les contrôleurs, les templates, les entités et les services, pour mettre en œuvre ces fonctionnalités d'authentification.

symfony list : Cette commande affiche une liste de toutes les commandes disponibles dans votre application Symfony, y compris les commandes système fournies par Symfony et les commandes personnalisées que vous avez ajoutées ou installées à l'aide de bundles tiers. Elle est utile pour voir toutes les fonctionnalités disponibles dans votre application Symfony et pour obtenir des informations sur leur utilisation.

symfony console list : Cette commande est similaire à "symfony list", mais elle affiche uniquement une liste des commandes disponibles dans la console Symfony (c'est-à-dire les commandes exécutables via la CLI). Cela inclut à la fois les commandes système de Symfony et les commandes personnalisées spécifiques à votre application Symfony.

En résumé, ces commandes sont utiles pour générer des fonctionnalités d'authentification, afficher une liste des commandes disponibles dans votre application Symfony et afficher une liste des commandes disponibles dans la console Symfony. Elles peuvent vous faire gagner du temps et vous aider à explorer les fonctionnalités disponibles dans Symfony.

DANS LES FICHIERS

- Dans le fichier registrationformtype.php **(dossier src /form)** ajouter cette ligne →add('name') comme ci-dessous :

```
$builder
->add('name')
->add('email')
...
```

- Puis, dans le fichier register.html.twig **(dossier template/registration)** ajouter cette ligne {{ form_row(registrationForm.name) }} comme ci-dessous :

```
{{ form_errors(registrationForm) }}
{{ form_start(registrationForm) }}
{{ form_row(registrationForm.name) }}
{{ form_row(registrationForm.email) }}
…
```

DANS LE TERMINAL

- **Créer une table dans la base de donnée**

```
symfony console make : entity User
symfony console make :migration
symfony console doctrine :migrations :migrate
```

symfony console make:user User : Cette commande génère automatiquement le code nécessaire pour créer une nouvelle classe d'utilisateur Symfony. Elle crée une classe User dans le répertoire de votre choix (généralement src/Entity ou src/Entity/User) avec les propriétés et méthodes de base d'un utilisateur. Cela facilite grandement la gestion des utilisateurs dans votre application Symfony.

symfony console make:migration : Cette commande génère automatiquement une migration basée sur les différences détectées entre vos entités Symfony et votre base de données. Une migration est un fichier PHP qui contient les instructions pour mettre à jour votre base de données afin de refléter les changements dans vos entités. Cela simplifie le

processus de mise à jour de votre schéma de base de données tout en maintenant la cohérence avec votre code.

symfony console doctrine:migrations:migrate : Cette commande exécute les migrations en attente sur votre base de données. Elle applique toutes les migrations qui n'ont pas encore été exécutées, mettant ainsi à jour votre base de données avec les derniers changements définis dans vos migrations. Cela garantit que votre base de données est toujours synchronisée avec votre code.

DANS LES FICHIERS

- Dans le fichier **security.yaml** **(dossier config/packages/), décommenter** les lignes suivantes comme ci-dessous :

  access_control:

  - { path: ^/admin, roles: ROLE_ADMIN }

  - { path: ^/profile, roles: ROLE_USER }

- Dans le fichier **appauthentificator.php** **(dossier src/security) remplacer** la function onAuthenticationSuccess comme ci-dessous :

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
    {
        if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
            return new RedirectResponse($targetPath);
        }

        // Récupérez l'utilisateur
        $user = $token->getUser();

        // Vérifiez si l'email de l'utilisateur est vérifié
        if (!$user->isVerified()) {
            // Ajoutez un message flash pour informer l'utilisateur de vérifier son e-mail
                $request->getSession()->getFlashBag()->add('warning', 'Veuillez confirmer votre adresse e-mail.');

            // Redirigez l'utilisateur vers la page d'accueil
```

```
            return new RedirectResponse($this->urlGenerator->generate('app_home')); // Remplacez 'home_route' par le nom de votre route d'accueil
        }

        // Récupérez les rôles de l'utilisateur
        $roles = $token->getUser()->getRoles();

        // Redirigez les utilisateurs en fonction de leurs rôles
        if (in_array('ROLE_ADMIN', $roles, true)) {
            return new RedirectResponse($this->urlGenerator->generate('admin')); // Remplacez 'admin_route' par le nom de votre route admin
        } elseif (in_array('ROLE_USER', $roles, true)) {
            return new RedirectResponse($this->urlGenerator->generate('app_profil')); // Remplacez 'profile_route' par le nom de votre route profil
        } else {
            throw new \Exception('No route found for user role.');
        }
    }

    protected function getLoginUrl(Request $request): string
    {
        return $this->urlGenerator->generate(self::LOGIN_ROUTE);
    }
```

DANS LE TERMINAL

- **Installer easy admin**

```
$ composer require easycorp/easyadmin-bundle
```

composer : Comme toujours, c'est le gestionnaire de dépendances pour PHP.

require : C'est la commande pour demander à Composer d'installer un nouveau package.

easycorp/easyadmin-bundle : C'est le nom du package à installer. EasyAdmin est un bundle Symfony qui facilite la création d'une interface d'administration pour vos applications Symfony. Il fournit une interface utilisateur prête à l'emploi pour gérer les entités de votre application, sans nécessiter beaucoup de code.

En résumé, cette commande installe EasyAdmin dans votre projet Symfony, vous permettant de créer rapidement et facilement une interface d'administration pour gérer vos entités Symfony. Cela simplifie le processus de création d'une interface d'administration complète et fonctionnelle pour votre application.

- **Mail asynchrone**

  $ symfony console messenger:consume async

symfony console : C'est la commande pour exécuter des commandes Symfony CLI.

messenger:consume : C'est la commande spécifique pour consommer les messages avec le composant Messenger. Le composant Messenger est utilisé pour implémenter des systèmes de messagerie asynchrones dans Symfony, permettant d'exécuter des tâches en arrière-plan de manière efficace.

async : C'est l'option qui spécifie le nom du transport à utiliser pour la consommation des messages. Dans ce cas, "async" est généralement configuré pour utiliser un transport asynchrone tel que RabbitMQ, Redis ou Amazon SQS. Vous pouvez configurer les transports dans le fichier "messenger.yaml" de votre application Symfony.

Lorsque vous exécutez cette commande, Symfony commence à consommer les messages du transport spécifié de manière asynchrone. Les messages sont récupérés du transport et traités par les handlers de message correspondants que vous avez configurés dans votre application Symfony.

En résumé, cette commande est utilisée pour démarrer le processus de consommation de messages asynchrones dans votre application Symfony à l'aide du composant Messenger. Cela peut être utile pour traiter des tâches en arrière-plan de manière efficace et améliorer les performances de votre application

- **Créer le dashboard controller**

  symfony console make:admin :dashboard

- **RUNNING SERVER**

  symfony server:start

  symfony server:stop

- **RUNNING COMMANDE**

  npm run dev

-> intall docker

modifier le contenu du fichier compose.yaml **(dans la racine)** comme ceci :

```
version: '3'

services:
  ###> doctrine/doctrine-bundle ###
  database:
    image: mysql:8.0
    environment:
      MYSQL_DATABASE: ${MYSQL_DATABASE:-symfony}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD:-root}
      MYSQL_USER: ${MYSQL_USER:-symfony}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD:-!ChangeMe!}
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      timeout: 10s
      retries: 5
      start_period: 30s
    volumes:
      - database_data:/var/lib/mysql
  ###< doctrine/doctrine-bundle ###

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    environment:
      PMA_HOST: database
      PMA_PORT: 3306
      PMA_ARBITRARY: 1
    ports:
      - '8080:80'
    depends_on:
      - database

volumes:
  ###> doctrine/doctrine-bundle ###
  database_data:
###< doctrine/doctrine-bundle ###
```

modifier le contenu du fichier compose.overrride.yaml **(dans la racine)** comme ceci :

```
version: '3'

services:
###> doctrine/doctrine-bundle ###
  database:
    ports:
      - "3306"
###< doctrine/doctrine-bundle ###

###> symfony/mailer ###
  mailer:
    image: axllent/mailpit
    ports:
      - "1025"
      - "8025"
    environment:
      MP_SMTP_AUTH_ACCEPT_ANY: 1
      MP_SMTP_AUTH_ALLOW_INSECURE: 1
###< symfony/mailer ###
```

➔ **Dans le terminal de commande :**

$ docker-compose up --build -d

Avec ce model de fichier compose.yml plus besoin de wamp et de php my admin , Tout est dans DOCKER . (base de donées etc).
Ce fichier peu etre installer dans n'importe quel projet

Puis Pour installer docker,

➔ **dans le terminal de commande :**
$ docker-compose up --build -d

```
version: '3.8'

services:
  php:
    image: php:8.3-apache
    container_name: php83
    ports:
      - "8000:80"
    volumes:
      - ./www:/var/www/html
    working_dir: /var/www/html
    command: docker-php-entrypoint apache2-foreground
    depends_on:
      - db

  db:
    image: mysql:5.7
    container_name: mysql57
    command: --default-authentication-plugin=mysql_native_password
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: projectdb
      MYSQL_USER: user
      MYSQL_PASSWORD: password
```

```
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadmin
    depends_on:
      - db
    ports:
      - "8080:80"
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: root

  node:
    image: node:latest
    container_name: node
    volumes:
      - ./www:/var/www/html
    working_dir: /var/www/html
    command: sh -c "npm install && npm run dev"

  composer:
    image: composer:latest
    container_name: composer
    volumes:
      - ./www:/var/www/html
    working_dir: /var/www/html
    command: composer install


volumes:
  db_data:
```

➔ Dans le terminal de commande

$ netstat -aon | findstr 3306

Cree un fichier nginx.conf (dans la racine) et y ajouter le code ci-dessous :

```
server {
    listen 8888;
    server_name localhost;

    root /var/www/html;
```

```
    index index.php index.html index.htm;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```