Projet Panthère

Symfony / Doctrine / Twig

1. Le framework Symfony

Créer un répertoire tp-symfony/ sous Wampserver

Installer composer si ce n’est pas déjà fait

<https://getcomposer.org/>

Installer symfony-CLI si ce n’est pas déjà fait.

<https://symfony.com/download>

Dans le répertoire tp-symfony, avec le terminal windows PowerShell, créer le projet projet-symfony :

* Symfony new projet-symfony --full

Dans le dossier projet-symfony, ajouter les composants suivants :

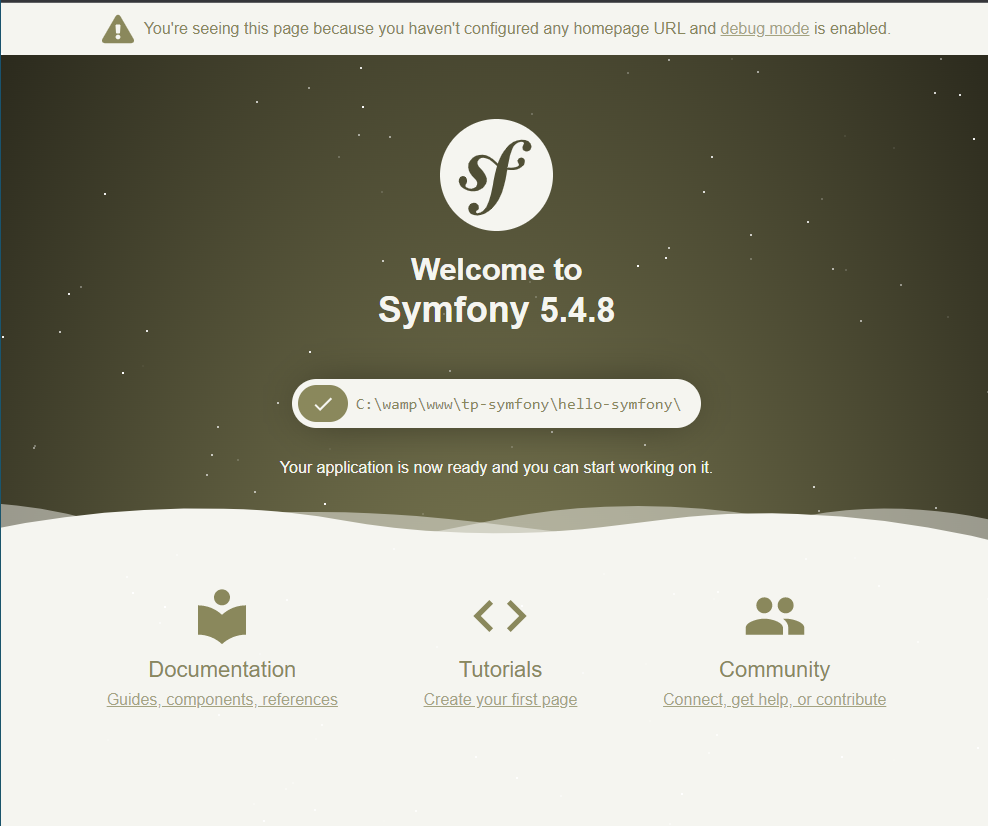
* composer require doctrine/annotations
* composer require twig/twig

Ouvrir un deuxième terminal.

Toujours dans le dossier projet et lancer le serveur :

* symfony server:start

Depuis un navigateur, se rendre à l’adresse http://127.0.0.1:8000



1. L’ORM Doctrine

L'utilisation de la [programmation orientée objet](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet) avec une base de données relationnelle nécessite de convertir les données relationnelles en objets et vice-versa.

Un **mapping objet-relationnel** (en anglais **object-relational mapping** ou **ORM**) définit des correspondances entre les schémas de la base de données et les classes du programme applicatif. On pourrait le désigner par-là, « comme une couche d'abstraction entre le monde objet et monde relationnel »

Cette technique permet à l’utilisateur d’utiliser les tables d’une base de données comme des objets.

Ajouter l’ORM Doctrine au projet

* composer require symfony/profiler-pack
* composer require symfony/orm-pack

Ouvrir le fichier .env à la racine du projet. Décommenter et modifier la ligne selon les paramètres de connexion à la base de données.

DATABASE\_URL="mysql://db\_user:db\_password@127.0.0.1:3306/db\_name?serverVersion=5.7&charser=utf8mb4"

1. Template Twig (un court aperçu)

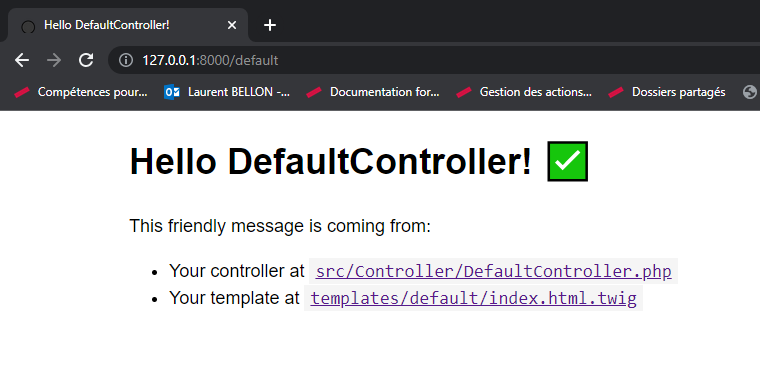
Depuis le terminal, se rendre dans le répertoire projet et installer le bundle « maker » :

* composer require maker --dev

Générer le contrôleur nommé ‘’Default’’ :

* php bin/console make:controller Default

Dans un navigateur se rendre à l’adresse : <http://127.0.0.1:8000/default>



Dans le répertoire **projet-symfony/** quatre fichiers nous intéressent.

1. projet-symfony/config/routes.yaml définit les règles de routage.
2. projet-symfony/src/Controller/DefaultController.php est le contrôleur qui vient d’être créé.

* La route pour la méthode index() est définie dans l’annotation @Route qui l’associe au routage **/default** (ligne 12)
* Cette méthode retourne le résultat de la méthode render() qui traite le fichier de template Twig associé au contrôleur. Le paramètre controller\_name prend pour valeur DefaultController.

Modifier le paramètre controller\_name en message.

Remplacer DefaultController par Voici mon site en symfony

1. projet-symfony/templates/base.html.twig définit un modèle de base twig commun à tous les modèles twig du site.
2. projet-symfony/templates/default/index.html.twig est le modèle twig qui vient d’être créé pour le controleur Default.

Remplacer le bloc title par Accueil

Remplacer le contenu de la balise div par

    <h1>Accueil</h1>

    <p>{{message}}</p>

Vérifier le résultat obtenu.

Editer le fichier routes.yaml de façon à définir le controleur DefaultController comme étant le point d’entrée du site. Décommenter les trois lignes :

index:

  path: /

  controller: App\Controller\DefaultController::index

Vérifier que l’url <http://127.0.0.1:8000> pointe bien vers la page d’accueil.

1. L’ORM Doctrine

Créer la base db\_service depuis la console :

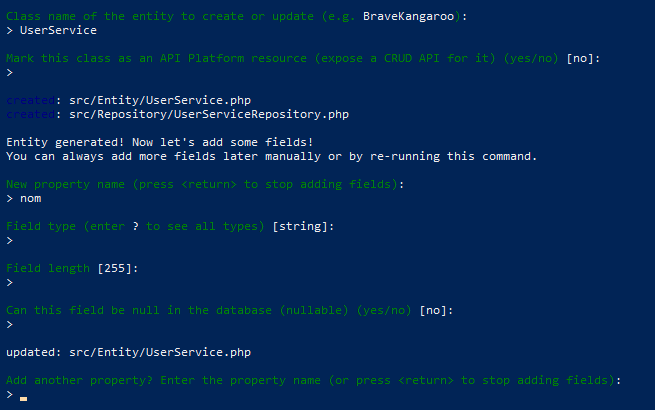
php bin/console doctrine:database:create

Créer une nouvelle entité (schéma de la future table) en utilisant Maker

php bin/console make:entity

Nommer l’entité UserService

Créer 2 propriétés (nom de type string (255) et age de type integer)



Ouvrir le fichier src/Entity/UserService.php et vérifier la correspondance avec le schéma. Notez que la propriété « Id » et son getter associé ont été créés automatiquement.

Depuis la console, faire la migration vers la BDD pour créer la table selon le schéma décrit par l’entité.

php bin/console make:migration

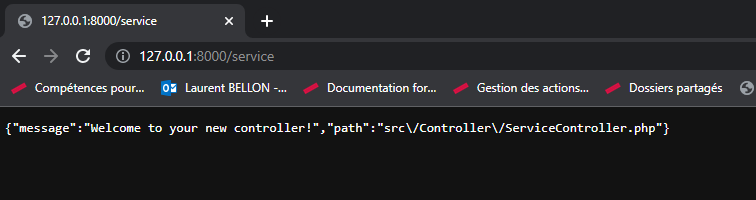
php bin/console doctrine:migrations:migrate

Créer un nouveau contrôleur nommé Service sans template :

php bin/console make:controller -–no-template Service

Tester le résultat dans un navigateur web :

<http://127.0.0.1:8000/service>



Notez qu’on affiche un objet JSON, car il n’y a pas de template associé à ce contrôleur.

Modifier le contrôleur ServiceController pour insérer quelques utilisateurs dans la table user\_service. Les données à insérer seront codées en ‘dur’.

Need Help ? See link: : https://symfony.com/doc/current/doctrine.html

* Inclure le manager d’Entités : use Doctrine\ORM\EntityManagerInterface ;
* Inclure l’entité UserService : use App\Entity\UserService;
* Créer une nouvelle méthode createAction. Lui associer une nouvelle route /service/create au moyen des annotations.
* Passer en paramètre de la méthode une variable de type EntityManagerInterface $em
* Créer une instance de UserService
* Utiliser les méthodes setName() et SetAge() de ce nouvel objet
* Ouvrir la persistence de l’entité : $em->persist($user);
* Insérer les valeurs dans la base : $em->flush();

Voici le résultat :

<?php

namespace *App\Controller*;

use *Symfony\Bundle\FrameworkBundle\Controller\AbstractController*;

use *Symfony\Component\HttpFoundation\JsonResponse*;

use *Symfony\Component\Routing\Annotation\Route*;

use *App\Entity\UserService*;

use *Doctrine\ORM\EntityManagerInterface*;

class ServiceController extends *AbstractController*

{

    /\*\*

     \* @Route("/service", name="app\_service")

     \*/

    public function index(): *JsonResponse*

    {

        return *$this*->json([

            'message' => 'Welcome to your new controller!',

            'path' => 'src/Controller/ServiceController.php',

        ]);

    }

    /\*\*

     \* @Route("/service/create", name="app\_service\_create")

     \*/

    public function createAction(*EntityManagerInterface* $em)

    {

        $user = new *UserService*();

        $user->setName('Sophie');

        $user->setAge('55');

        $em->persist($user);

        $em->flush();

        return *$this*->json([

            'message' => 'New user',

            'value' => $user->getName()

        ]);

    }

}

Modifier le contrôleur ServiceController pour chercher des utilisateurs dans la table user\_service en appelant le repository associé à l’entité UserService.

* Créer une méthode searchAction. Lui associer une route /service/search au moyen des annotations.
* Passer en paramètre de la méthode une variable de type EntityManagerInterface $em
* Appeler le UserServiceRepository au moyen de la méthode   
  $em->getRepository(UserService ::class) de l’objet $em.
* Tester les différentes méthodes find(), findOneBy(), findBy(), findAll()

Modifier le controleur ServiceController pour modifier un élément de la base.

* Créer une méthode updateAction. Lui associer une route /service/update au moyen des annotations.
* Passer en paramètre de la méthode une variable de type EntityManagerInterface $em

1. Le bundle API Plateform

L’objectif est de créer une API de gestion des ressources humaines d’une équipe ou d’une entreprise. Elle repose sur deux tables Team (Equipe) regroupant les collaborateurs et Position (fonction) regroupant les fonctions de l’entreprise.

Supprimer toutes les tables précédemment créées

Supprimer l’entité UserService

Supprimer le Repository

Supprimer les fichiers de migrations

Vider le cache symfony

php bin/console cache:clear

Ajouter API Plateform au projet

Composer require api

Composer require migrations

Créer une nouvelle entité nommée Team avec les propriétés firstname:string et lastname:string

Répondre « yes » à la question « Mark this class as an API Plateform resource »

Faire la migration afin de créer la table en BDD

php bin/console make:migration

php bin/console doctrine:migrations:migrate

Créer une nouvelle entité Position avec la propriété label:string

php bin/console make:entity

Répondre « yes » à la nouvelle question « Mark this class as an API Plateform resource »

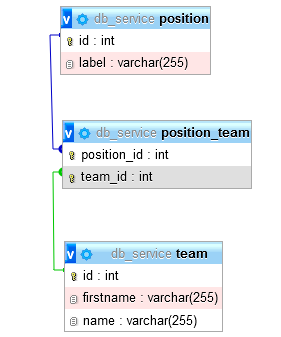
Ajouter un deuxième champ appelé users qui sera de type **ManyToMany** : puis, dire qu'il sera relié à la classe Team, puis répondre **yes** pour « Do you want to add a new property to Team »

Faire la migration pour créer les tables :

php bin/console make:migration

php bin/console doctrine:migrations:migrate

Vérifier dans phpMyAdmin, la création automatique d’une table de liaison (type ManyToMany)



Avec le navigateur web, accéder au centre de gestion de l’API :

* http://127.0.0.1:8000/api

1. Consommer le webservice

Créer les métiers dans la table position au moyen de la requête POST : Gérant, Commercial, Comptabilité, Lead dev, Dev, UI/UX, Testeur, Stagiaire.

Reproduire en base de données l’organigramme suivant en utilisant l’interface API **sans les liens hiérarchiques entre collaborateurs** dans un premier temps.

Paul Martin

(lead Dev)

Arthur Vincent

(Dev)

Justine Dupont

(Dev)

John Deer

(UI / UX)

Charly Goose

(Testeur stagiaire)

Sylvie Durand

(Commercial)

Martine Duck

(Comptabilité)

Paul Stone

(Gérant)

Enrichir le schéma de BD pour intégrer

* la notion de lien hiérarchique entre collaborateurs
* Photo des collaborateurs (chemin vers fichier image)
* Âge
* Adresse
* Tel
* Mail
* CV (chemin vers fichier PDF)

1. Interface Symfony / Twig

Coder une application en Symfony / Twig qui affiche l’organigramme de la structure en tenant compte de l’arrivée ou du départ des collaborateurs qui seront gérés côtés API.

On pourra se contenter d’afficher l’arborescence de l’organigramme au moyen d’indentations :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Paul Stone (gérant) | | |  |  |
|  |  |  |  |  |
|  | Paul Martin (Lead dev) | | |  |
|  |  |  |  |  |
|  |  | Justine Dupont (Dev) | |  |
|  |  |  |  |  |
|  |  |  | Charly Goose (Testeur Stagiaire) | |
|  |  |  |  |  |
|  |  | Arthur Vincent (Dev) | |  |
|  |  |  |  |  |
|  |  | John Deer (UI / UX) | |  |
|  |  |  |  |  |
|  | Sylvie Durand (Commercial) | | |  |
|  |  |  |  |  |
|  | Martine Duck (Comptable) | | |  |

Les photos des collaborateurs seront affichées devant leur nom et au format 50px x 50px.

En cliquant sur le nom du collaborateur (ou sa photo), sa fiche complète est affichée.

**Bonus**: Ecrire une fonction vérifiant la cohérence de la base de données : si un collaborateur part, les collaborateurs dont il est le supérieur sont rattachés à son supérieur immédiat.