**Cours SYMFONY 5**

|  |
| --- |
| 1. **Introduction** |

Symfony est un *Framework* coté serveur (backend), qui est basé sur le langage PHP.

|  |
| --- |
| 1. **Environnement logiciel et installations** |

1. **Xampp** :

* Il offre le serveur Local (apache)
* Il offre la base de données (MySql).
* Il offre le langage PHP

1. **VSCODE** :

* Il joue le rôle de l’IDE pour pouvoir développer, exécuter, compiler….

1. **Composer** :

* Il permet d’installer des packages PHP en cas de besoin.

1. **Symfony CLI (Comand Line Interface):**

**ATTENTION :**

* Le fichier .exe de CLI on le lance à l’emplacement où on veut créer le projet (Pour Windows).
* Ou par la commande :

**scoop install symfony-cli**

A l’emplacement où je vais créer mon projet.

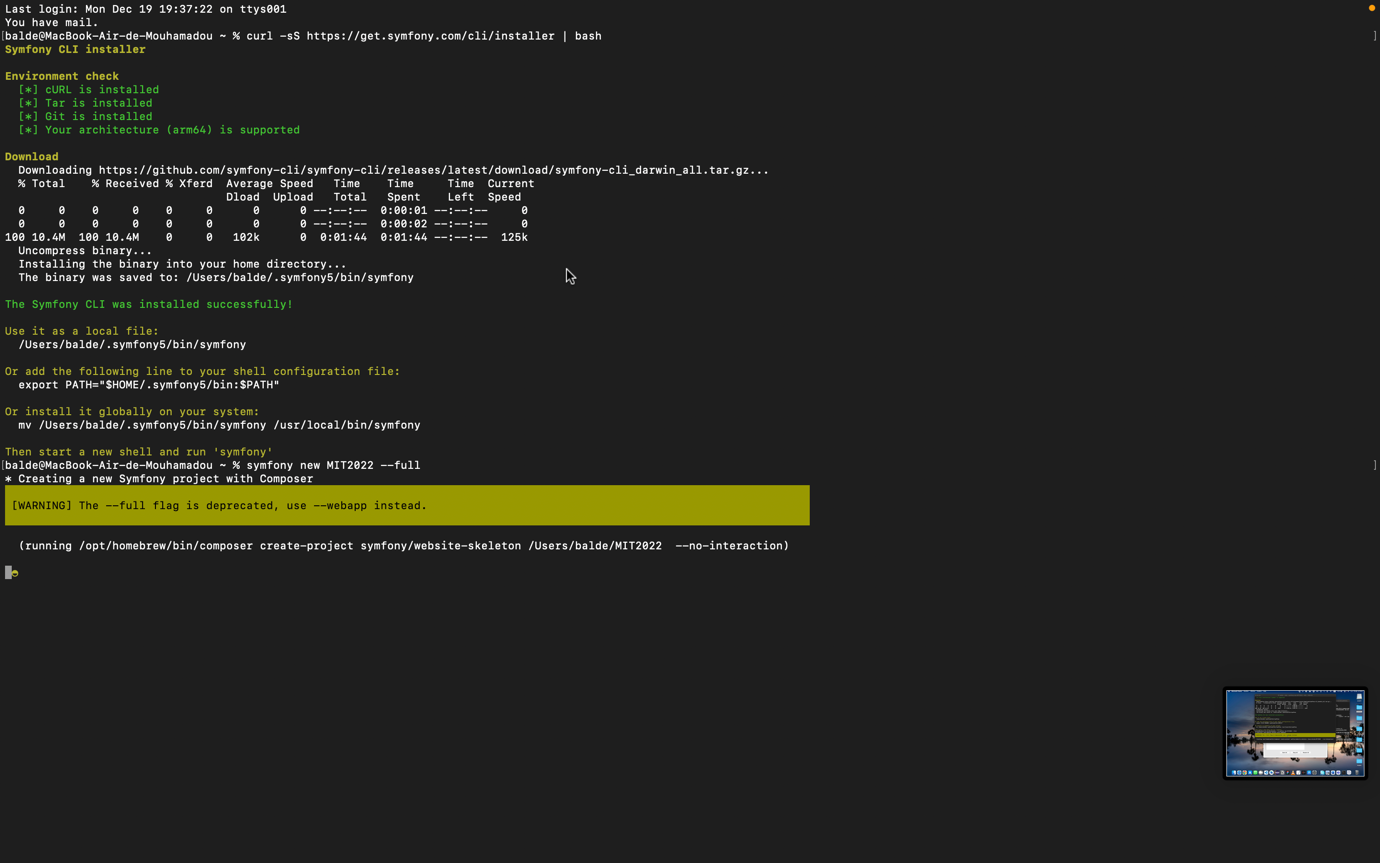
|  |
| --- |
| 1. **Création de notre premier projet Symfony** |

Pour la création de projet, on doit lancer la **commande suivante** :

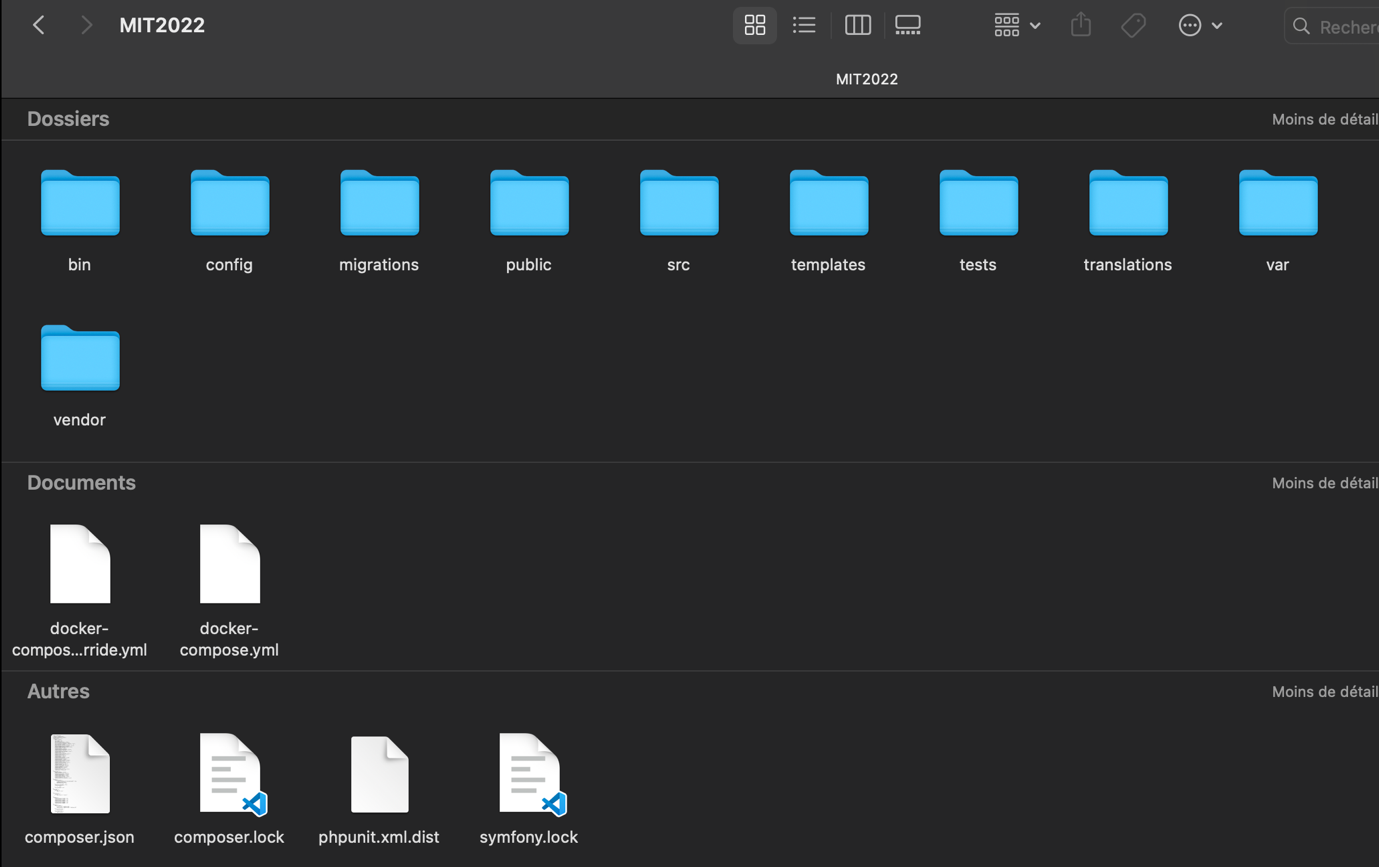
**symfony new nomduprojet –full**

Par exemple : Je veux créer un projet intitulé **MIT2022**.  
Je saisie la commande suivante :

**symfony new MIT2022 –full**



Un dossier intitulé MIT2022 est crée. Ce dossier contient tous les dossiers et les fichiers de mon projet.



ATTENTION : Pour exécuter le projet on doit entrer dedans.

cd MIT2022

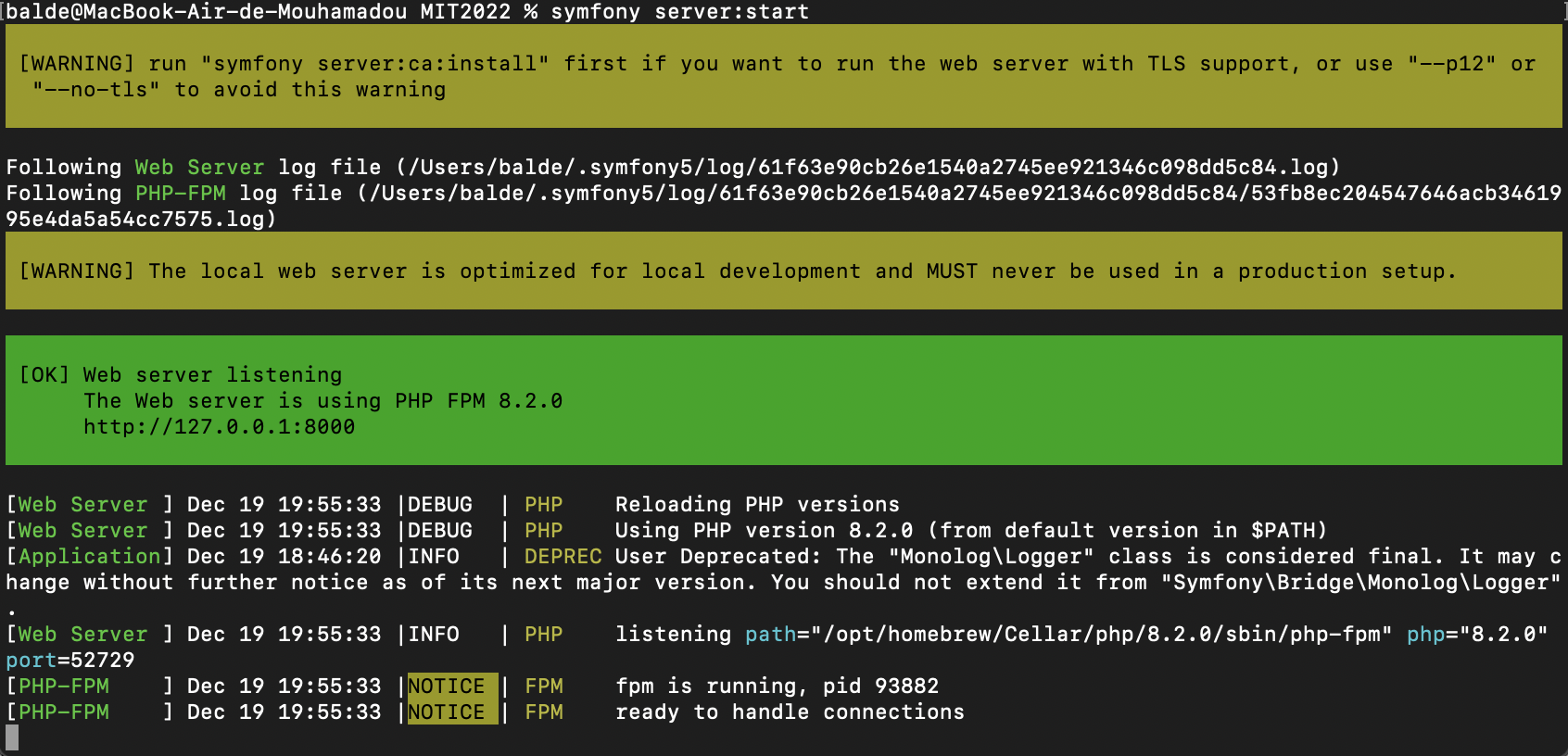
balde@MacBook-Air-de-Mouhamadou MIT2022 %

|  |
| --- |
| 1. **Exécution du projet** |

Après la création du projet on doit l’exécuter pour voir le résultat.

Avec la commande :

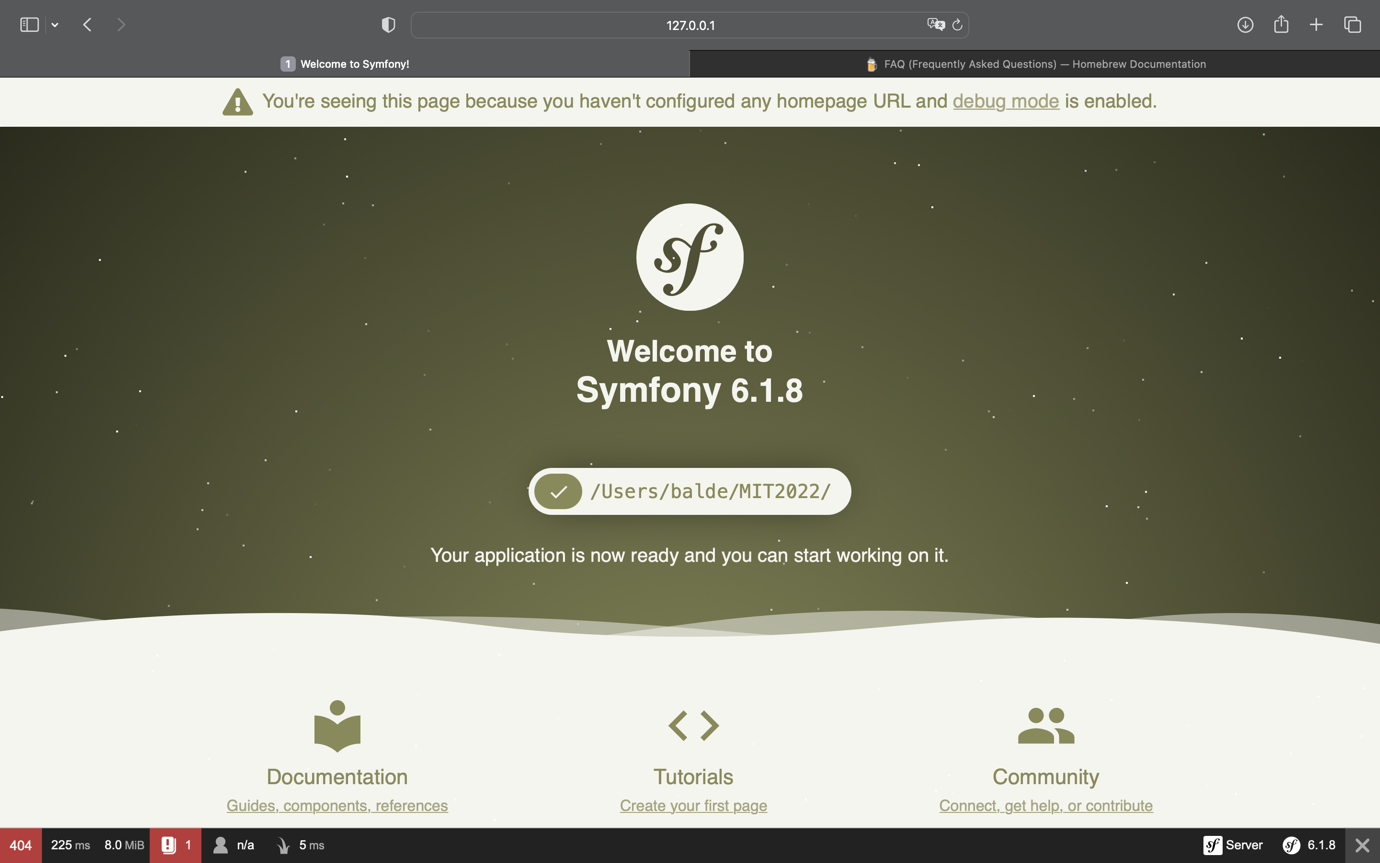
**symfony server:start**



ATTENTION : N’oublier d’ouvrir xampp(xampp panel) et d’activer le serveur apache.

Le projet s’exécute et se lance à travers l’url suivante :

http://127.0.0.1:8000  
Pour voir le résultat on copie cette URL et on la met dans le navigateur.



|  |
| --- |
| 1. **Création du Controller** |

* Après la création du projet on doit commencer par la création des Controlleurs.
* Le controller va gérer la communication entre :

**Les Modèles** (classe, entité, table ………)

ET

**Les Vues** (les pages HTML)

* Notre premier Controller sera intitulé TaskController.php

**Voilà le Code de notre Controller**

<?php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

class TaskController

{

/\*\*

\* @Route("/1")

\*/

public function Mouhamadou():Response{

return new Response("Je suis la fonction Mouhamadou");

}

/\*\*

\* @Route("/2")

\*/

public function Latif():Response{

return new Response("Je suis la fonction Latif");

}

/\*\*

\* @Route("/3")

\*/

public function Souleymane():Response{

return new Response("Je suis la fonction Souleymane");

}

}

?>

|  |
| --- |
| 1. **Router** |

On veut lancer par exemple le contenu d’une fonction AfficheMenu() à travers la **path** : **/menu**

**Voilà le Code de notre Controller**

/\*\*

\* @Route("/menu")

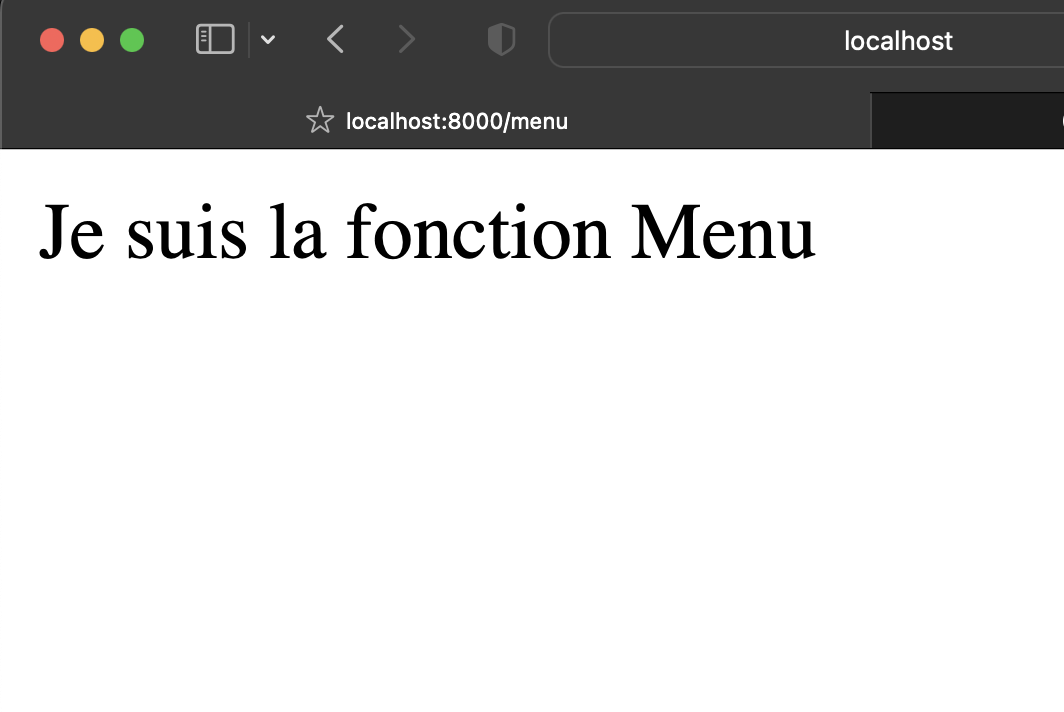
\*/

public function Affichemenu():Response{

return new Response("Je suis la fonction Menu");

}

**Voilà le Résultat**



Pour pouvoir utiliser les routes on doit Lancer la commande suivante :

**composer require annotations**

Remarque : N’oublier pas d’importer les classes nécessaires dans vos codes, exemple dans notre controlleur on a fait les importations suivantes :

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

|  |
| --- |
| 1. **Type des fonctions du contrôleur** |

**7.1. Affichage d’un simple message :**

return new Response ('bonjour je suis la fonction de mohamed');

* 1. **Affichage d’une simple page HTML**

return $this->render("page.html.twig");

    }

* 1. **Affichage d’une page html en passant des paramètres :**

        $firstVariable = "MIT";

        return $this->render(

            "ecole.html.twig",

            ["secondVariable"=>$firstVariable]

        );

**NB :**

**pour les point 7.2 et 7.3 on doit rendre la classe TaskController comme :**

class TaskController *extends* AbstractController {

On doit aussi lancer la commande :

**composer require twig**

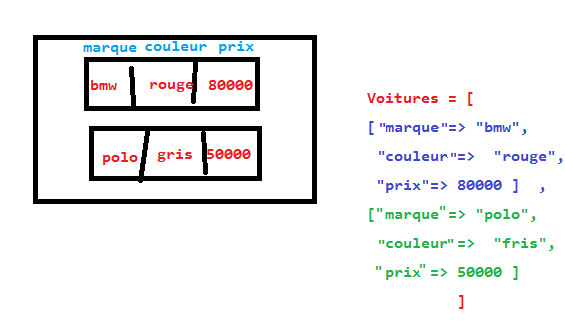
|  |
| --- |
| 1. **Les boucles avec Symfony** |

Créer dans notre Controller une fonction **ventevoiture().**

Cette fonction contient une variable **voiture**, cette variable est sous la forme de **tableau d’objet (voiture [ [ ],[ ],[ ]....    ]; ).**

Dans une autre page **shop.html.twig** afficher les voitures sous la forme d’un tableau.

Chaque voiture est caractérisée par sa marque, sa couleur et son prix.



**Dans le controller (Taskcontroller.php)**

**Voilà le code concret de cette fonction :**

/\*\*

\* @Route("/voiture")

\*/

public function VenteVoiture(){

$Voiture = [

[

"Marque" => 'peugot',

"Couleur" => 'gris',

"Prix" => 80000,

],

[

"Marque" => 'ferrari',

"Couleur" => 'noir',

"Prix" => 10000000,

],

[

"Marque" => 'nissan',

"Couleur" => 'jaune',

"Prix" => 300000,

],

[

"Marque" => 'Mercedes',

"Couleur" => 'rose',

"Prix" => 100000000,

],

[

"Marque" => 'renault',

"Couleur" => 'vert',

"Prix" => 20000,

],

[

"Marque" => 'toyota',

"Couleur" => 'blanc',

"Prix" => 450000,

],

[

"Marque" => 'range rover',

"Couleur" => 'gris',

"Prix" => 100000000000,

]

] ;

return $this->render('shop.html.twig',

['varpagehtml'=>$Voiture]);

}

**Voilà le code de notre fichier shop.html.twig**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>{% block title %} Voiture!{% endblock %}</title>

<link rel="icon" href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22 viewBox=%220 0 128 128%22><text y=%221.2em%22 font-size=%2296%22>⚫️</text></svg>">

{# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #}

{% block stylesheets %}

{{ encore\_entry\_link\_tags('app') }}

{% endblock %}

{% block javascripts %}

{{ encore\_entry\_script\_tags('app') }}

{% endblock %}

<style>

thead { background-color:black;color:green}

tbody { background-color:pink;color:black}

</style>

</head>

<body>

{% block body %}

<h1>la liste des voitures est: </h1>

<table>

<thead>

<tr>

<th>Marque</th>

<th>Couleur</th>

<th>Prix</th>

</tr>

</thead>

<tbody>

{% for oneVoiture in varpagehtml %}

<tr>

<td>{{ oneVoiture.Marque }}</td>

<td>{{oneVoiture.Couleur}}</td>

<td>{{oneVoiture.Prix}}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% endblock %}

</body>

</html>

**Et voilà le résultat en utilisant le chemin /voiture :**

****

|  |
| --- |
| 1. **Création de la base de données** |

On va créer une base de donnée intitulée **Tasks**, on écrit la commande suivant :

**composer require symfony/orm-pack**

Puis la commande :

**composer require –dev symfony/maker-bundle**

Maintenant on se dirige vers le ficher**. env**: pour faire la configuration nécessaire pour la création de la base de données.

Dans ce fichier**. env** on modifie les deux lignes 31 et 32 de la façon suivante :

DATABASE\_URL="mysql://root:@localhost/Tasks?serverVersion=8&charset=utf8mb4"

*# DATABASE\_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?*

Maintenant on va créer la base grâce à la commande :

**php bin/console doctrine:database:create**

|  |
| --- |
| 1. **Création de la table Task** |

En symfony, les tables sont désignées par le mot **Entity** (coté code). On va créer notre première table avec la commande suivante :

**php bin/console make:entity**

|  |
| --- |
| 1. **Migration de l’entité vers la table correspondante** |

Pour faire cela, on lance la commande :

**php bin/console make:migration**

Après cette commande, un fichier dans le dossier migration est créé, Maintenant pour faire concrètement la migration, on écrit la commande:

**php bin/console doctrine:migrations:migrate**

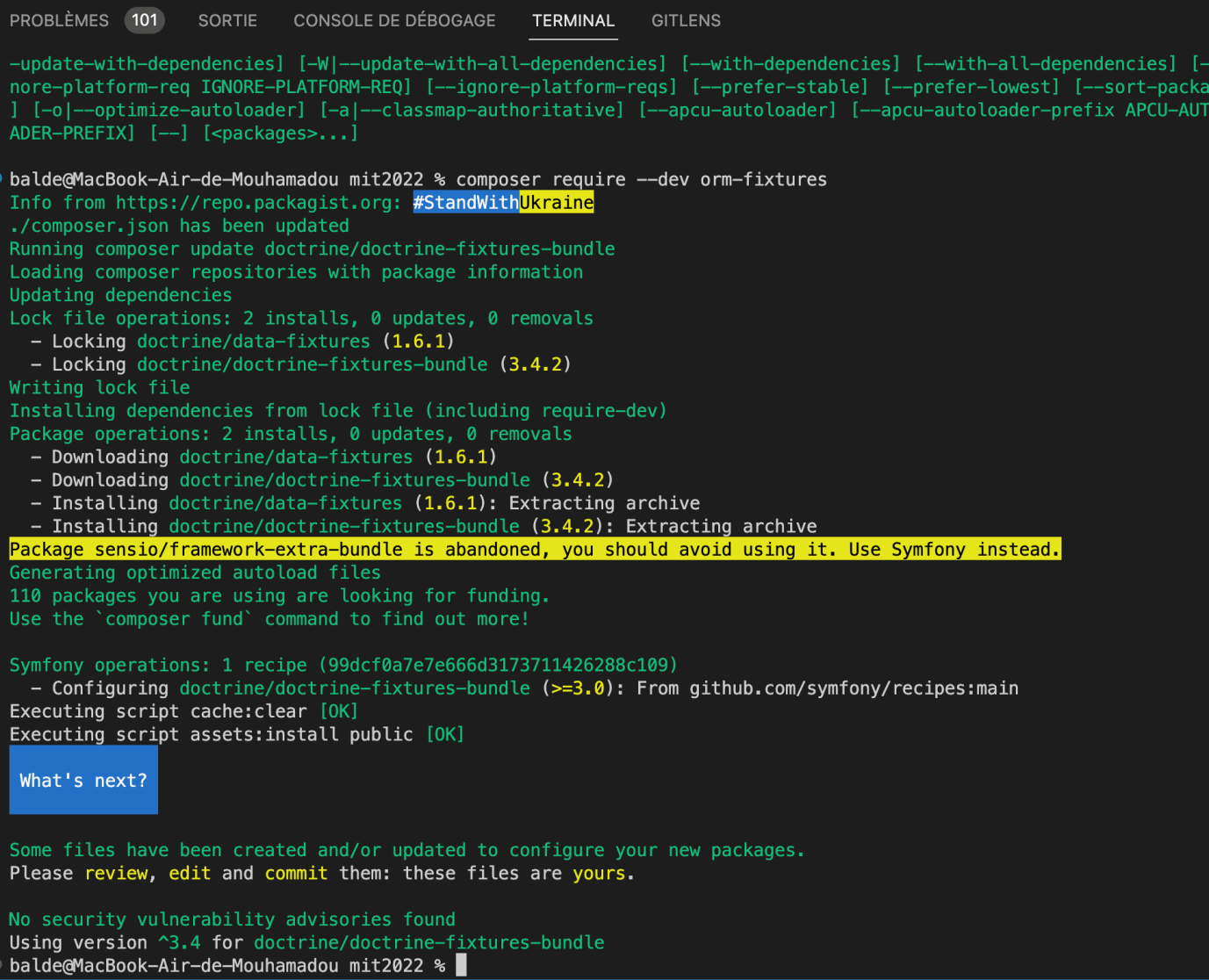
|  |
| --- |
| 1. **Fixtures** |

Les fixtures sont des classes utiliseées pour générer des données fictives (fake data/ou des données de votre choix) et remplir nos tables.

Nous allons créer notre première fixtures intitulée  **TaskFixture** pour remplir notre table Task.

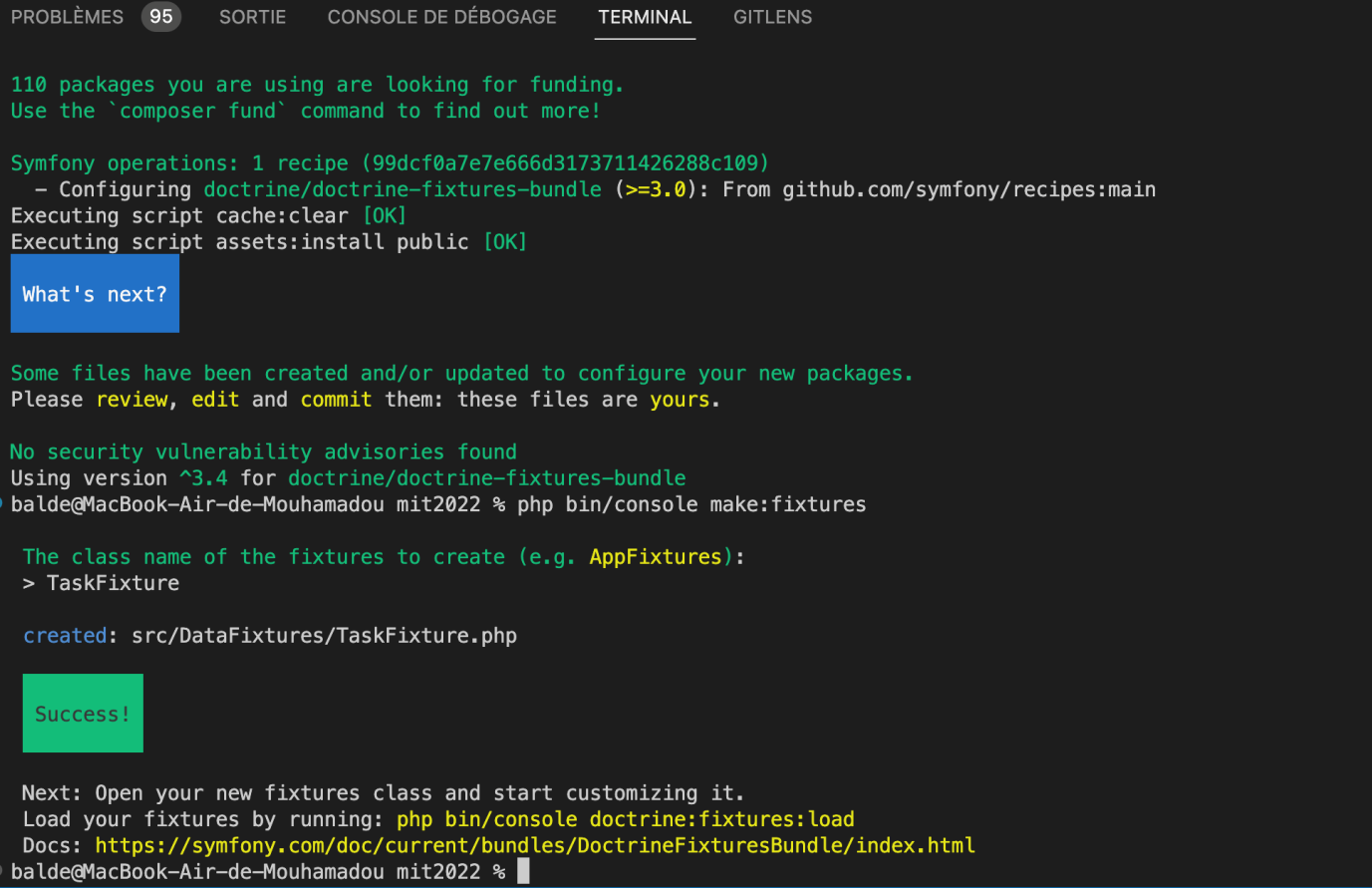
Pour faire cela on écrit la commande :

**composer require –dev orm-fixtures**



Puis la commande :

**php bin/console make:fixtures**



Les commandes ci-desus permettent de créer un dossier DataFixtures dans le dossier src.

Ce dossier va contenir :

* Notre fixture sous la forme d’une page php **TaskFixture.php**.
* Un exemple pour comprendre : **AppFixture.php**

Voilà ci-dessous le code de notre Fixture:

<?php

namespace App\DataFixtures;

use Doctrine\Bundle\FixturesBundle\Fixture;

use Doctrine\Persistence\ObjectManager;

class TaskFixture extends Fixture

{

public function load(ObjectManager $manager): void

{

// $product = new Product();

// $manager->persist($product);

$manager->flush();

}

}

On doit utiliser la fonction **load()** de notre fixture pour remplir notre table Task.

Voilà le contenue de la fonction l**oad()** qui permet de créer 10 Tasks :

public function load(ObjectManager $manager): void

{

//ici on va préparer nos données pour charger la table Task

for ($i = 1; $i <= 10;$i++){

$T = new Task();

//id de l'élément i

//title de l'élément i

$T->setTitle('Le titre'.$i);

//le body de l'élément i

$T->setBody('Le body'.$i);

$manager->persist($T);

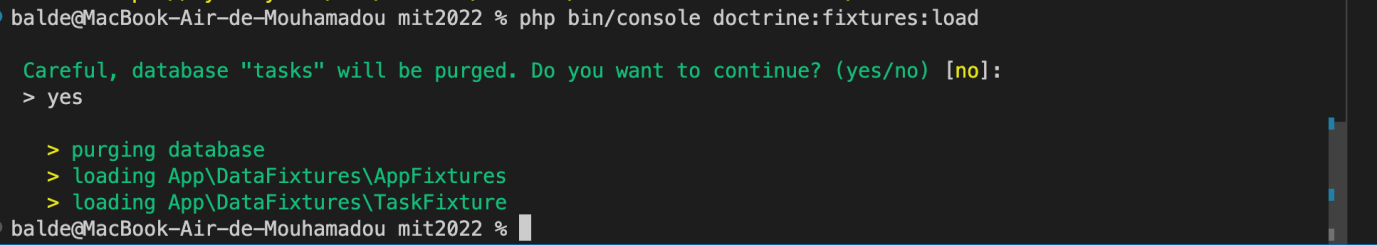
}

$manager->flush();

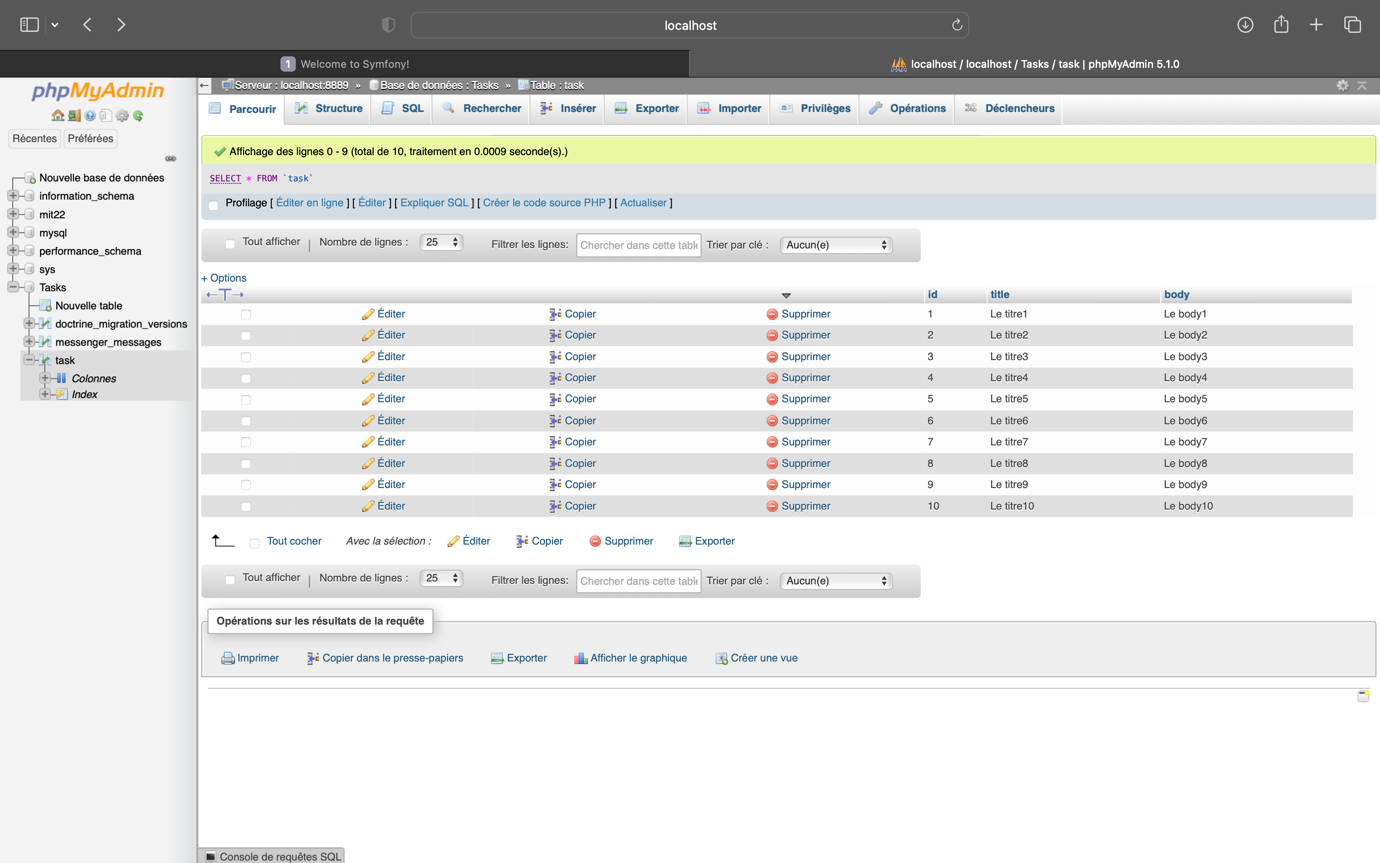
}

Pour remplir la table avec les 10 Tasks on doit exécuter la commande suivante**:**

**php bin/console doctrine :fixtures :load**



Maintenant voilà le résultat (la table est bien remplie par les 10 tasks) :



|  |
| --- |
| 1. **Affichage du contenue de la table Task dans notre application** |

Pour faire cette étape avec sucée if faut suivre la démarche suivante :

* Création de la fonction (nom de votre choix), son contenue permet de :
* récupérer des données de la partie model (entity => table dans la base de donnée)
* If fait un traitement sur / avec les données.
* Envoyer les données vers la partie vue pour l’affichée.
* Création de la page html qui va afficher et récupère les données de la fonction ci-dessus :

NB : lors de la création de l’entité (qui va créer une classe dans le dossier entity dans notre cas : Task.php) un autre ficher intitulé **nom-de-lentitéRepository.php** (**TaskRepository.php**) sera créé dans le dossier repository, le but de cette classe et de gérer la communication des données entre l’entité et sa table correspondante dans la base. (Les tables concrètes)

Voila la réponse :

Le code de la fonction dans le controller :

*/\*\**

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

*\*/*

*public* function allTask (TaskRepository $tr ) : Response {

        $lesTaches = $tr->findAll();

        return $this->render(

            "allTasks.html.twig",

            ["x"=>$lesTaches]

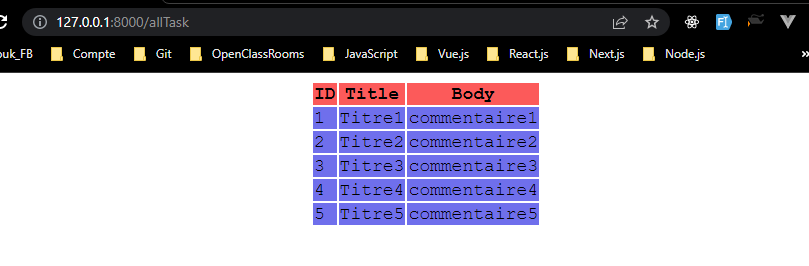
        );

    }

Sans oublier de mettre cette ligne dans le code du controlleur :

use App\Repository\TaskRepository;

Et voilà le résultat :



|  |
| --- |
| 1. **Bootstrap** |

On Ajoute le Bootstrap dans le projet par son ajout dans le fichier **base.html.twig** :

        {% block stylesheets %}

        <link *rel*="stylesheet" *href*="https://bootswatch.com/4/litera/bootstrap.min.css">

Puis on ajoute la navbar :

    <body>

        <nav *class*="navbar navbar-expand-lg navbar-light bg-light">

            <a *class*="navbar-brand" *href*="#">Formation symfony 2022</a>

            <button *class*="navbar-toggler" *type*="button" *data-toggle*="collapse" *data-target*="#navbarNavAltMarkup" *aria-controls*="navbarNavAltMarkup" *aria-expanded*="false" *aria-label*="Toggle navigation">

              <span *class*="navbar-toggler-icon"></span>

            </button>

            <div *class*="collapse navbar-collapse" id="navbarNavAltMarkup">

              <div *class*="navbar-nav">

                <a *class*="nav-item nav-link active" *href*="{{path("jou")}}">Home <span *class*="sr-only">(current)</span></a>

                <a *class*="nav-item nav-link" *href*="{{path("allTask")}}">Show all tasks</a>

              </div>

            </div>

        </nav>

        {% block body %}

        {% endblock %}

    </body>

on intègre le style et le contenu de la page base.html.twig dans toutes les page du projet avec la ligne suivante :

{% extends 'base.html.twig' %}

Le résultat devient comme suit :



**15 Creation de font awsom button :**

La premaire chois en doit ajouter dans le style de la page base.html.twig le code suivant :

        <link *rel*="stylesheet" *href*="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.1/css/all.min.css" *integrity*="sha512-MV7K8+y+gLIBoVD59lQIYicR65iaqukzvf/nwasF0nqhPay5w/9lJmVM2hMDcnK1OnMGCdVK+iQrJ7lzPJQd1w==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"/>

Maintenant en va creer les button devon chaque tache dans la page html.twig qui affiche la tableu de nos taches

Vola le code apret lajout bootstrap et font awsom

{% extends 'base.html.twig' %}

{% block title %}

{% endblock %}

{% block body %}

<style>

    #customers {

        font-family: Arial, Helvetica, sans-serif;

        border-collapse: collapse;

        width: 70%;

        align: center;

        margin: auto

    }

    h1 {

        background-color: blue;

        color: white;

        text-align: center;

        align: center

    }

    #customers td,

    #customers th {

        border: 1px solid #ddd;

        padding: 8px;

        align: center;

        text-align: center;

    }

    #customers tr:nth-child(even) {

        background-color: #f2f2f2;

        align: center

    }

    #customers tr:hover {

        background-color: #ddd;

        align: center

    }

    #customers th {

        padding-top: 12px;

        padding-bottom: 12px;

        text-align: center;

        background-color: blue;

        color: white;

    }

</style>

    <h1>la liste de vos teches</h1>

    <table id="customers" *border*="1">

        <thead>

            <th>ID</th>

            <th>Title</th>

            <th>Body</th>

            <th>Action</th>

        </thead>

        <tbody>

            {% for oneelement in x %}

                <tr>

                    <td>{{oneelement.id}}</td>

                    <td>{{oneelement.title}}</td>

                    <td>{{oneelement.body}}</td>

                    <td>

                        <a *href*="#" *class*="btn btn-sm btn-primary"><i *class*="fa fa-eye"></i></a>

                        <a *href*="#" *class*="btn btn-sm btn-warning"><i *class*="fa fa-edit"></i></a>

                        <a *href*="#" *class*="btn btn-sm btn-danger"><i *class*="fa fa-trash"></i></a>

                    </td>

                </tr>

            {% endfor  %}

        </tbody>

    </table>

{% endblock %}

Est vola le resultat :



**16 Affichage dune seul tache en detail (read by id de la base ):**

Maintenet en clicon sue le button eye en blue devont chaque tache , cette tache sera affiche en detail, seule, dans une autre page html.twig et a traversé un autre route différant de la route de la page de tous les taches

Vola la solution :

Dans la page AllTask.html.twig en modifie le lien du button eye de la facon suivant :

                        <a *href*="{{path("details",{id:oneelement.id})}}" *class*="btn btn-sm btn-primary"><i *class*="fa fa-eye"></i></a>

Maintenant en doit crer une page detail.html.twig qui affiche les details de la tache qont ve consulter

Romarque

cette page doit avoire une route dynamique, exemple :

/detail/1 : ce la page detail.html.twig de la tache 1

/detail/3 : ce la page detail.html.twig de la tache 3

/detail/20 : ce la page detail.html.twig de la tache 20

En doit gerer cette acpee dynamique dans la fonction coté controller .

Vola le code de cette fonction dans le controller :

*/\*\**

*\* @Route("/detail/{id}", name="details")*

*\*/*

*public* function oneTask (TaskRepository $tr, $id) : Response {

        $uneTache = $tr->find($id);

        return $this->render("detail.html.twig", ["y"=>$uneTache]);

    }

Finalement vola le code de la page detail.html.twig :

{% extends 'base.html.twig' %}

{% block title %}

{% endblock %}

{% block body %}

 <style>

#customers {

    font-family: Arial, Helvetica, sans-serif;

    border-collapse: collapse;

    width: 70%;

    align: center;

    margin: auto

}

h1 {

    background-color: blue;

    color: white;

    text-align: center;

    align: center

}

#customers td,

#customers th {

    border: 1px solid #ddd;

    padding: 8px;

    align: center;

    text-align: center;

}

#customers tr:nth-child(even) {

    background-color: #f2f2f2;

    align: center

}

#customers tr:hover {

    background-color: #ddd;

    align: center

}

#customers th {

    padding-top: 12px;

    padding-bottom: 12px;

    text-align: center;

    background-color: blue;

    color: white;

}

</style>

    <h1>les details de votre tache : </h1>

    <table id="customers" *border*="1">

        <thead>

            <th>ID</th>

            <th>Title</th>

            <th>Body</th>

        </thead>

        <tbody>

                <tr>

                    <td>{{y.id}}</td>

                    <td>{{y.title}}</td>

                    <td>{{y.body}}</td>

                </tr>

        </tbody>

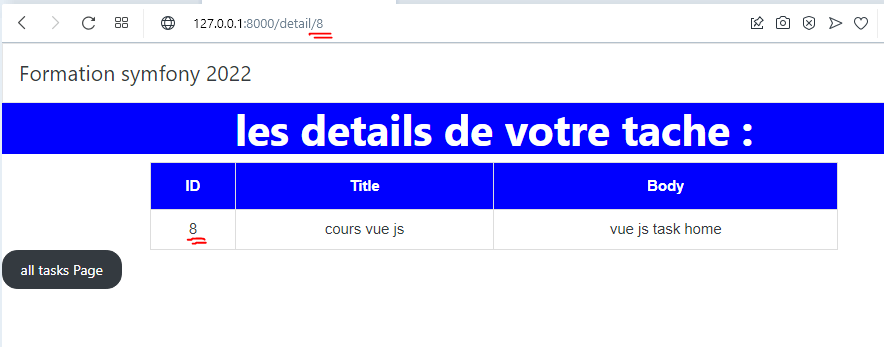
    </table>

    <a *href*="{{path("allTask")}}" *class*="btn btn-dark m5">all tasks Page </a>

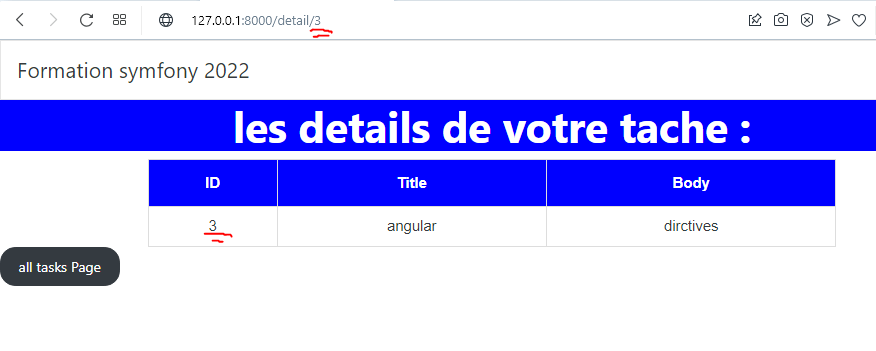
{% endblock %}

Et vola le resulat :

Pour afficher les details de la tache 8



Pour afficher les details de la tache 3



**17 Création d’une nouvelle tâche :**

Dans le fichier base.html.twig en doit ajouter une option (add Task) dans la nav bar qui permet douvrire une nouvelle page html content la **forme** a remplire par l’utilisateur :

Lidée ici est de :

* Creation de la forme qui l’utiisateur va repmlier pour cree une tache
* Cree une page html add.html.twig qui va charger la forme
* La fonction dans le controller qui va geré cette operation

17.1 cration de la forme :

En doit lance la commande suivant :

composer require symfony/form

En doit créer un fichier nom-de-la-tableType.php (dans notre cas TaskType.php) dans un noveaux dossier Form dans le dossier src.

Vola le code :

TaskController.php

*/\*\**

*\* @Route("/add", name="AddTask")*

*\*/*

*public* function addtask(Request $req)

     {

         $task = new Task;

         $form = $this->createForm(TaskType::class, $task);

         $form->handleRequest($req);

         if ($form->isSubmitted() && $form->isValid()) {

             $task = $form->getData();

             $entityManger = $this->$getDoctrine()->getManager();

             $entityManger->persist($task);

             $entityManger->flush();

             return $this->redirectToRoute('allTask');

         }

         return $this->render('add.html.twig', [

             'form' => $form->createView(),

         ]);

     }

}

TaskType.php

<?php

namespace App\Form;

use App\Entity\Task;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextareaType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\Form\FormBuilderInterface;

use Symfony\Component\OptionsResolver\OptionsResolver;

class TaskType *extends* AbstractType

{

*public* function buildForm(FormBuilderInterface $builder, array $options)

    {

        $builder

            ->add('title', TextType::class)

            ->add('body', TextareaType::class)

            ->add('valider', SubmitType::class);

    }

*public* function configureOptions(OptionsResolver $resolver): void

    {

        $resolver->setDefaults([

            'data\_class' => Task::class,

        ]);

    }

}

add.html.twig

{% extends 'base.html.twig' %} {% form\_theme form 'bootstrap\_4\_layout.html.twig'

%} {% block title %}bonjour ! {% endblock %} {% block body %}

<style>

    #customers {

        font-family: Arial, Helvetica, sans-serif;

        border-collapse: collapse;

        width: 70%;

        align: center;

        margin: auto;

    }

    h1,

    h2 {

        background-color: blue;

        color: white;

        text-align: center;

        align: center;

    }

    #customers td,

    #customers th {

        border: 1px solid #ddd;

        padding: 8px;

        align: center;

        text-align: center;

    }

    #customers tr:nth-child(even) {

        background-color: #f2f2f2;

        align: center;

    }

    #customers tr:hover {

        background-color: #ddd;

        align: center;

    }

    #customers th {

        padding-top: 12px;

        padding-bottom: 12px;

        text-align: center;

        background-color: blue;

        color: white;

    }

</style>

<h1>Ajouter une tache:</h1>

{{ form\_start(form) }}

{{ form\_widget(form) }}

{{ form\_end(form) }}

{% endblock %}

Finalement voici la résultat final :



