

Symfony

Framework = ensemble de composants pour être plus productif, sert à créer les fondations.
C'est une grande bibliothèque

Symfony est un framework web

Il fonctionne suivant le principe REQUEST/ RESPONSE utilisé par le protocole HTTP

Suit l'architecture logicielle MVC

Utilise le langage PHP 7 orienté Objet

Peut se connecter à beaucoup de SGBDR (système de gestion de bdd relationnel)

Il fonctionne suivant le principe REQUEST/RESPONSE utilisé par le protocole HTTP

Doctrine => ORM

1- Installation

Vous devriez au préalable consulter la documentation d'installation

<https://symfony.com/doc/current/setup.html>

Configuration requise :

- Php version 7.2
- Composer

Création dépôt GIT / repositories :

Question 1 : Créez un projet "website-skeleton" *essai*. Observez le contenu du projet... Pas facile !

```
composer create-project symfony/website-skeleton my_project name
```

```
composer create-project symfony/website-skeleton:"^4.4"
```

```
php c:\composer\v2\composer.phar create-project "retour à la ligne"
```

```
symfony/website-skeleton:"^4.4" essai
```

Aller sur le lien donné pour trouver le token

```
php -S localhost:8000 -t public
```

```
php c:\composer\v2\composer.phar create-project  
symfony/website-skeleton:"^4.4" feedback_admin
```

Routage

Pour le moment le message voulu de s'affiche pas

Méthode à base d'annotations

Créer des routes de manière plus souple, en utilisant des annotations.

Pour cela on peut ajouter directement des annotations dans la class controller.

```
/**  
 * @Route  
 */
```



Faire une copie du .env en .env.local

Dans le .env.local mettre le # sur postgresql et l'enlever sur mysql

```
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"  
# DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=13&charset=utf8"
```

ensuite mettre ces informations de base de données

Pour créer une BDD avec symfony en ligne de commande :

Ne pas oublier de mettre en mysql et de créer le .env.local

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

```
PS C:\Users\jenohug\Documents\php_sym\essa1> php bin/console doctrine:database:create
Created database `db_name` for connection named default
```

```
PS C:\Users\jenohug\Documents\php_sym\essa1> php bin/console make:controller
```

Choose a name for your controller class (e.g. **BraveChefController**):

> TableController

created: src/Controller/TableController.php

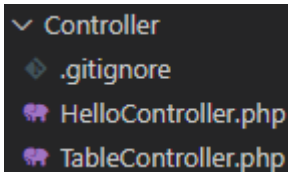
created: templates/table/index.html.twig

Success!

Next: Open your new controller class and add some pages!

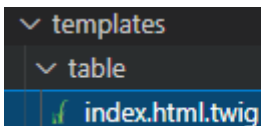
```
PS C:\Users\jenohug\Documents\php_sym\essa1>
```

Ainsi un fichier dans src et Controller c'est créer :



```
Controller
├── .gitignore
├── HelloController.php
└── TableController.php
```

et aussi dans templates :



```
templates
├── table
│   └── index.html.twig
```

Si l'on veut juste afficher toute les commandes : *php bin/console*

faire une requête avec la route :

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

/**
 * @Route("/table", name="table")
 */
class TableController extends AbstractController
{
    /**
     * @Route("/print/{n}/{m}", name="table_print")
     */
    public function index(int $n, int $m, Request $request): Response
    {
        $color = $request->get('c');
        return $this->render('table/index.html.twig', [
            'controller_name' => 'TableController',
            'n' => $n,
            'm' => $m,
            'color' => $color,
        ]);
    }
}
```

Pas besoin de modifier la route pour y avoir accès il faut utiliser Request et récupérer c

Table de 5

Nombre	Multiplieur	Résultat
0	5	0
1	5	5
2	5	10
3	5	15
4	5	20
5	5	25
6	5	30
7	5	35
8	5	40
9	5	45
10	5	50

```
<div class="example-wrapper">
  <h1 style='color:{{color}}'>Table de {{n}}</h1>

  <table border='1px' style='border-color:{{color}}'>
    <thead>
      <th style='color:{{color}}'>Nombre</th>
      <th style='color:{{color}}'>Multiplieur</th>
      <th style='color:{{color}}'>Résultat</th>
    </thead>
    <tbody>

      {% for i in 0..m %}
      <tr><td>{{i}}</td><td>{{n}}</td><td>{{n*i}}</td></tr>
      {% endfor %}

    </tbody>
  </table>
```

Formulaire

php bin/console make:form

Une classe formulaire se termine par Type

pour générer des formulaire et en faire

Appeler le de le class TableChoiceType

Ne rien mettre à la 2ème proposition

```
PS C:\Users\jenohug\Documents\php_sym\essa1> php bin/console make:form

The name of the form class (e.g. OrangePopsicleType):
> TableChoiceType

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
>

created: src/Form/TableChoiceType.php

Success!

Next: Add fields to your form and start using it.
Find the documentation at https://symfony.com/doc/current/forms.html
PS C:\Users\jenohug\Documents\php_sym\essa1>
```

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

{% block title %}Choise table number{% endblock %}

{% block body %}

    {{ form(formulaire)}}

{% endblock %}

/**
 * @Route("/select", name="table_select")
 */
public function select(){

    $form = $this->createForm(TableChoiceType::class);

    return $this->render('table/vue.html.twig',[
        'formulaire' => $form->createView(),
    ]);
}
```

Faire une liste

```
$builder->add('table_member_list', ChoiceType::class,[
    'choices' => [
        'Table de 1' => 1,
```

```
'Table de 2' => 2,  
'Table de 3' => 3,  
'Table de 4' => 4,  
'Table de 5' => 5,  
'Table de 6' => 6,  
'Table de 7' => 7,  
'Table de 8' => 8,  
'Table de 9' => 9,  
'Table de 10' => 10,  
'Table de 11' => 11,  
],  
]);
```

Pour installer plusieurs projets dans Symfony, il faut utiliser VirtualHost.

```
> php bin/console make:entity
```

```
php bin/console make:migration
```

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

```
yes
```

Permet de générer une migration

```
PS C:\Users\jenohug\Documents\202-symfony-tp-orm\2021-symfony-tp-orm> php bin/console make:crud
```

```
The class name of the entity to create CRUD (e.g. TinyElephant):
```

```
> User
```

```
Choose a name for your controller class (e.g. UserController) [UserController]:
```

```
>
```

Injection de Dépendance

Design pattern "Singleton"

But : 1 seule instanciation

Chaque new crée un objet

il faut pouvoir utiliser plusieurs implémentations d'une dépendance

ne pas gérer les instances dans la classe qu'il utilise

Injection de dépendance par setter (mutateur)

par propriété

- plus flexible que singleton
- possibilité d'utiliser un mock object
- jsp
-

Service container -> permet de gérer les services

- type hinting = spécifier le type de variable transmise à la méthode
- autowiring = injecter automatiquement les paramètres connus

```
public function new(Request $request): Response
```

le service container instancie à notre place avec Request

php bin/console make:user -> implémenter la gestion des utilisateur

```
PS C:\Users\jenohug\Documents\symfony tp security username> php bin/console make:user
```

```
The name of the security user class (e.g. User) [User]:
```

```
> User
```

```
Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
```

```
> yes
```

```
Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
```

```
> username
```

```
Does this app need to hash/check user passwords? (yes/no) [yes]:
```

```
> yes
```

```
created: src/Entity/User.php
```

```
created: src/Repository/UserRepository.php
```

```
updated: src/Entity/User.php
```

```
updated: config/packages/security.yaml
```

Permet de créer un utilisateur en utilisant un username (pas email) et de pouvoir hasher un mdp

DOC SYMFONY :

<https://symfony.com/doc/current/reference/configuration/security.html#form-login-authentication>

```
Entity > User.php
<?php

namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;

/**
 * @ORM\Table(name="tbl_user")
 * @ORM\Entity(repositoryClass=UserRepository::class)
 */
class User implements UserInterface
{
    /**
     * @ORM\Table(name="tbl_user")

```

ajouter cette ligne avant la migration pour le nom de la table

Implémenter des fixtures qui vont créer 3 user pour les test

SuperAdmin

Admin

User

composer require orm-fixtures --dev → install composant orm-fixtures

les composant se situe dans packagist.org

```
PS C:\Users\jenohug\Documents\symfony_tp_security_username> composer require orm-fixtures --dev
Warning from https://repo.packagist.org: Support for Composer 1 is deprecated and some packages will
PS C:\Users\jenohug\Documents\symfony_tp_security_username> php bin/console make:fixtures

The class name of the fixtures to create (e.g. AppFixtures):
> UserFixtures
```

Injection de dépendances dans la classe UserFixtures

```
class UserFixtures extends Fixture
{
    private $passwordEncoder = null;

    public function load(ObjectManager $manager): void
    {
        // $product = new Product();
        // $manager->persist($product);

        $manager->flush();
    }

    public function __construct(UserPasswordEncoderInterface $passwordEncoder)
    {
        $this->passwordEncoder = $passwordEncoder;
    }
}
```

Implémenter la méthode *load()*

dans la classe UserFixtures pour créer un user “superadmin”

```
$superadmin = new User();
$superadmin->setUsername('superadmin');
$superadmin->setRoles(array('ROLE_SUPER_ADMIN'));
$password = $this->passwordEncoder->encodePassword($superadmin,
'superadmin');
$superadmin->setPassword($password);
$manager->persist($superadmin);
```

remplacer le superadmin par admin pour créer un user admin

user

user

```
public function load(ObjectManager $manager): void
{
    // $product = new Product();
    // $manager->persist($product);

    $superadmin = new User();
    $superadmin->setUsername('superadmin');
    $superadmin->setRoles(array('ROLE_SUPER_ADMIN'));
    $password = $this->passwordEncoder->encodePassword($superadmin, 'superadmin');
    $superadmin->setPassword($password);
    $manager->persist($superadmin);

    $admin = new User();
    $admin->setUsername('admin');
    $admin->setRoles(array('ROLE_ADMIN'));
    $password = $this->passwordEncoder->encodePassword($admin, 'admin');
    $admin->setPassword($password);
    $manager->persist($admin);

    $user = new User();
    $user->setUsername('user');
    $user->setRoles(array('ROLE_USER'));
    $password = $this->passwordEncoder->encodePassword($user, 'user');
    $user->setPassword($password);
    $manager->persist($user);

    $manager->flush();
}
```

pour changer le type d'encodage config/packages/security.yaml

```
> php bin/console doctrine:fixtures:load
```

```
"App\DataFixtures\UserPasswordEncoderInterface" but this class was not found.
```

```
use App\Entity\User;
use Doctrine\Persistence\ObjectManager;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;
```

penser à importer la class si il y a cette erreur et refaire commande
sinon

```
PS C:\Users\jenohug\Documents\symfony_tp_security_username> php bin/console doctrine:fixtures:load

Careful, database "symfony_tp_security_username" will be purged. Do you want to continue? (yes/no) [no]:
> yes
```

mettre yes pour charger les fixtures

```

role_hierarchy:
  ROLE_ADMIN:       ROLE_USER
  ROLE_SUPER_ADMIN: ROLE_ADMIN

```

dans le security.yaml ajouter cela

cela permet que le role SA à aussi le role ADMIN
et que le le role admin à aussi le role USER

```
php bin/console make:auth
```

permet de créer le form de login

```

What style of authentication do you want? [Empty authenticator]:
[0] Empty authenticator
[1] Login form authenticator
> 1

```

```

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> LoginFormAuthenticator

```

```

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
> SecurityController

```

```

Do you want to generate a '/logout' URL? (yes/no) [yes]:
> yes

```

ensuite pour voir si cela a fonctionner : ./login pour accéder au form et /logout pour se déco

src/security/Login...

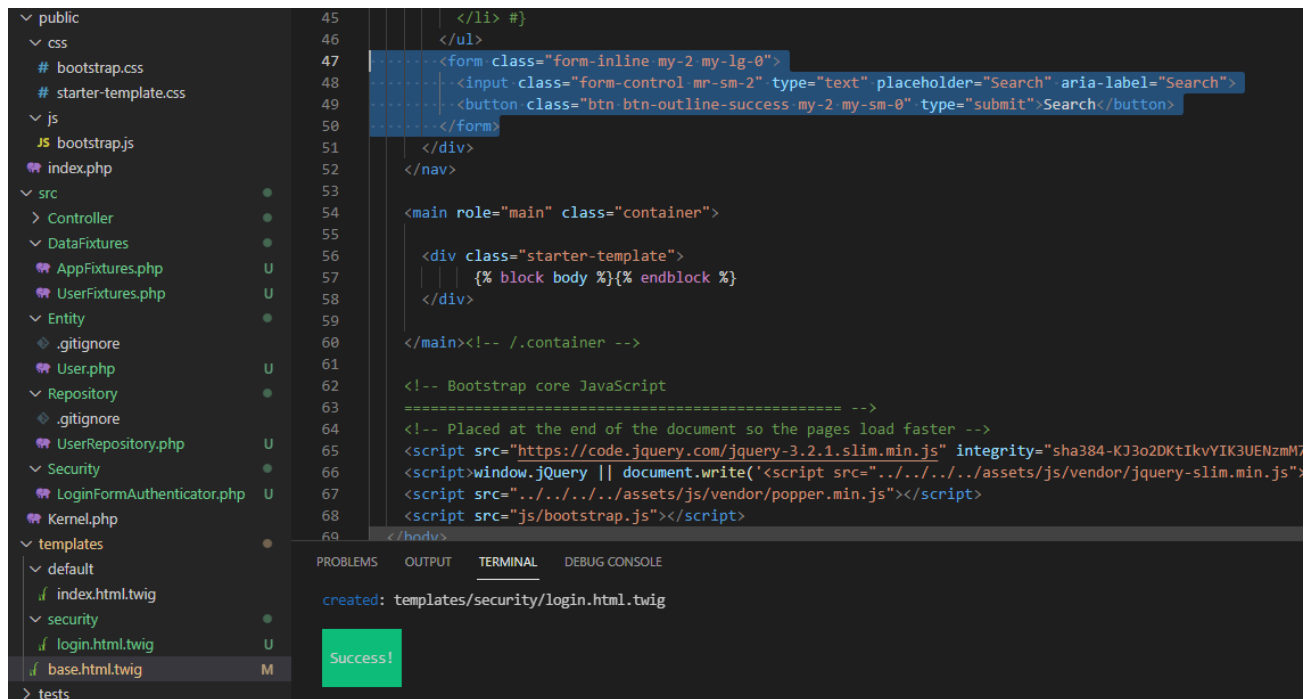
```

92     public function onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)
93     {
94         if ($targetPath = $this->getTargetPath($request->getSession(), $providerKey)) {
95             return new RedirectResponse($targetPath);
96         }
97
98         return new RedirectResponse($this->urlGenerator->generate('home'));
99         // throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
100     }
101
102     protected function getLoginUrl()
103     {
104         return $this->urlGenerator->generate(self::LOGIN_ROUTE);
105     }

```

décommenter la ligne 98 et modif le generate
commenter la ligne 99

permet de préciser la route home utilis après la connexion d'un user



remplacer ce qui est surligner par :

```
<ul class="navbar-nav">
    {% if is_granted('ROLE_USER') %}
        <li>
            <div class="nav-link disabled">
                <i class="fas fa-user"></i>
                {% if app.user %}
                    <strong>{{ app.user.username }}</strong>
                {% endif %}
            </div>
        </li>
        <li>
            <a class="nav-link" href="{{ path('app_logout') }}">{{ 'Logout'|trans }}</a>
        </li>
    {% else %}
        <li>
            <a class="nav-link" href="{{ path('app_login') }}">{{ 'Sign in'|trans }}</a>
        </li>
    {% endif %}
</ul>
```