# TP Sécurité Symfony

## Table des matières



**Auteur :** Mithridate Mathieu
**Relu, validé & visé par :**
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

**Date création :** 22/10/2023
**Date révision :** 22/10/2023

# TP Sécurité Symfony

## Description du module :

Dans ce module nous allons mettre en œuvre la sécurisation de la connexion d'un projet **Symfony** par :

-Un enregistrement (*Formulaire de création de compte*)

-Une connexion sécurisée (*Formulaire de connexion*),

-Mise en place de l'authentification par un **Token JWT** (*API*).

## Prérequis (outils) :

Pour mettre en œuvre le TP nous allons avoir besoin des outils et techno suivantes :

-Base de données MySQL (*ou MariaDB*),

-PHP 8 +,

-Composer,

-Git (Git Bash),

-Symfony CLI,

-Un compte de messagerie (*type Outlook, Laposte etc.…*)

## 1 Mise en place du projet :

### 1.1 Création du projet Symfony

Créer un répertoire qui va contenir les projets Symfony,
Ouvrir un terminal **git bash** dans le répertoire et saisir les lignes de commande ci-dessous :

```
symfony new projet –webapp
cd projet
```

### 1.2 Configuration du projet Symfony

1 Ouvrir le fichier **.env** à la racine du projet et ajouter la ligne suivante (configuration de la BDD) :

```
DATABASE_URL="mysql://loginBdd:motPasseBdd@127.0.0.1:3306/nomBdd?serverVersion=10.11.2-MariaDB&charset=utf8mb4"
```

2 Commenter la ligne :

```
#
DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=15&charset=utf8"
```

# TP Sécurité Symfony

**NB** : Assurez-vous que votre serveur MySQL ou MariaDB est bien lancé.

## 1.3 Création des entités :

Pour créer l'entité utilisateur nous allons utiliser dans le terminal la commande ci-dessous :

```
symfony console make:user
```

Relancer la commande pour ajouter les attributs suivants à notre entité (*name, firstname*)

## 1.4 Mise en place de la classe Sécurité :

Pour mettre en place l'authentification nous allons utiliser le module **auth** de Symfony à l'aide de la commande ci-dessous dans le terminal :

```
symfony console make:auth
```

## 1.5 Création du controller RegisterController pour gérer l'inscription :

Afin de gérer l'inscription (création de compte) nous allons créer un controller que nous allons nommer **RegisterController** à l'aide de la commande ci-dessous dans le terminal :

```
symfony console make:controller
```

**NB** : Attention à bien nommer les classes avec une **majuscule** (*première lettre*).

## 1.6 Création du formulaire d'inscription :

Pour permettre à nos utilisateurs de créer un compte nous allons générer un formulaire symfony (classe Type) qui va contenir les attributs suivants (*name, firstname, email, password et confirmation de password*). Afin de générer celui-ci nous allons utiliser la commande ci-dessous dans un terminal :

```
symfony console make:form
```

**NB** : Dans les options nous allons choisir nom : **Register** et **User** pour l'entité liée.

## 1.7 Configuration du formulaire RegisterType :

Nous allons éditer la méthode buildForm de la classe **src/Form/RegisterType.php**, afin d'ajouter les différents inputs HTML ainsi que leurs configurations comme ci-dessous :

```
<?php
namespace App\Form;
use App\Entity\User;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;
use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
```

# TP Sécurité Symfony

```
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;

class RegisterType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array
$options): void
    {
        $builder
            ->add('firstname', TextType::class,
            [
                'attr'=>['class'=>'register'],
                'label' => 'Saisir votre Prénom : ',
                'required'=>false
            ])
            ->add('name', TextType::class,
            [
                'attr'=>['class'=>'register'],
                'label' => 'Saisir votre nom : ',
                'required'=>false
            ])
            ->add('email', EmailType::class,
            [
                'attr'=>['class'=>'form'],
                'label' => 'Saisir votre email : ',
                'required'=>true
            ])
            ->add('password', RepeatedType::class, [
                'type' => PasswordType::class,
                'invalid_message' => 'Les mots de passe ne correspondent
pas',
                'options' => ['attr' => ['class' => 'password-field']],
                'required' => true,
                'first_options'  => ['label' => 'Password'],
                'second_options' => ['label' => 'Repeat Password'],
            ])
            ->add('Ajouter', SubmitType::class)
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
```

# TP Sécurité Symfony

```
        $resolver->setDefaults([
            'data_class' => User::class,
        ]);
    }
}
```

## 1.8 Configuration du template base Twig :

Afin d'afficher de gérer simplement les imports **JS** et **CSS** sur chacune de nos pages nous allons paramétrer le fichier **templates/base.html.twig**.

1 Ajouter vos fichiers **CSS** et **JS** dans le répertoire **public/asset/style/** (*CSS*) et **public/asset/script/** (*JS*) :

2 Configurer le fichier **templates/base.html.twig** comme ci-dessous :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Base{% 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>">
        {% block stylesheets %}
            <link rel="stylesheet"
href="{{ asset('asset/style/bootstrap.min.css') }}">
            <link rel="stylesheet"
href="{{ asset('asset/style/main.css') }}">
            <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.0/css/all.min.css">
        {% endblock %}
        {% block javascripts %}
            <script src="{{ asset('asset/script/bootstrap.min.js') }}"
async></script>
            <script src="{{ asset('asset/script/main.js') }}"
async></script>
        {% endblock %}
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

# TP Sécurité Symfony

## 1.9 Edition du template templates/register/index.html.twig :

Nous allons éditer le template pour intégrer le **formulaire** et les éléments **HTML** de la page. Pour cela nous allons ajouter le code ci-dessous dans le block **body** :

```
 {{ form_start(form) }}
        {{ form_row(form.firstname)}}
        {{ form_row(form.name)}}
        {{ form_row(form.email)}}
        {{ form_row(form.password.first) }}
        {{ form_row(form.password.second) }}
    {{ form_end(form)}}
    <p>{{message}}</p>
```

## 1.10 Ajout de la méthode addUser dans RegisterController :

Nous allons ajouter la méthode **addUser** dans le controller **RegisterController.php** afin d'ajouter un compte en base de données. La méthode va devoir vérifier si le compte n'existe déjà dans la base de données (*pour ne pas créer des doublons*).

1 Importer les dépendances suivantes dans le controller :

```
use App\Entity\User;
use App\Repository\UserRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;
use App\Form\RegisterType;
use Symfony\Component\HttpFoundation\Request;
```

2 Ajouter la méthode **addUser** dans le controller qui va générer le formulaire et enregistrer le compte en base de données. En utilisant le code ci-dessous :

```
#[Route('/register', name: 'app_register')]
    public function adduser(EntityManagerInterface $em, UserRepository
$repo,
    UserPasswordHasherInterface $hash, Request $request): Response
    {
        $msg = "";
        $user = new User();
        $form = $this->createForm(RegisterType::class, $user);
        $form->handleRequest($request);
        if($form->isSubmitted() AND $form->isValid()){
            if($repo->findOneBy(['email'=>$user->getEmail()])){
                $msg ="Le compte : ".$user->getEmail()." existe déjà";
            }
```

# TP Sécurité Symfony

```
            else{
                $pass = $request->request->all('register')['password']['first'];
                $hash = $hash->hashPassword($user, $pass);
                $user->setPassword($hash);
                $user->setRoles(["ROLE_USER"]);
                $user->setActivated(false);
                $em->persist($user);
                $em->flush();
                $msg = "Le compte : ".$user->getEmail()." a été ajouté";
            }
        }
        return $this->render('register/index.html.twig', [
            'message' => $msg,
            'form' => $form->createView(),
        ]);
    }
```

## 1.11 Migrer la base de données :

Pour déployer la base de données nous allons utiliser les commandes suivantes dans le terminal :

1 Créer la base de données :

```
symfony console doctrine:database:create
```

2 Créer un fichier de migration (structure de la base de données)

```
symfony console make:migration
```

3 Migrer la structure sur le serveur de base de données :

```
symfony doctrine:migrations:migrate
```

**NB** : Vérifier si la base de données et la table user à bien été ajouté en BDD.

## 1.12 Tester la création de compte :

Pour vérifier si notre méthode addUser fonctionne correctement nous allons tester la méthode dans le navigateur internet :

1 Installer le certificat SSL en utilisant la commande suivante dans le terminal :

```
symfony server:ca:install
```

# TP Sécurité Symfony

2 Démarrer le serveur en utilisant la commande suivante dans le terminal :

```
symfony server:start -d
```

3 Saisir l'url suivante dans le navigateur :

https://localhost/register ou https://127.0.0.1/register

4 Tester d'ajouter un compte :

Vérifier en ajoutant un compte dans le formulaire si le message ci-dessous apparait bien :

**Le compte : ***@***.*** a été ajouté**

Vérifier en saisissant le même **email** si le message ci-dessous apparait bien :

**Le compte : ***@***.*** existe déjà**

## 1.12 Création d'un service :

Dans cette partie nous allons créer un service Utils pour intégrer nos fonctions de sécurité.

1 Ajouter un répertoire **src/Service**.

2 Créer une classe **Utils.php** avec le code ci-dessous :

```
<?php
namespace App\Service;
class Utils{
    /**
     * @param string $value
     * @return null|string
     */
    public static function cleanInputStatic(string $value):?string{
        return htmlspecialchars(strip_tags(trim($value)));
    }
    /**
     * @param string  $date
     * @param string  $format
     * @return bool
     */
    public static function isValid($date, $format = 'Y-m-d'):bool{
        $dt = \DateTime::createFromFormat($format, $date);
        return $dt && $dt->format($format) === $date;
    }
}
?>
```

# TP Sécurité Symfony

## 1.13 Mise à jour de la fonction addUser :

Nous allons utiliser la fonction **cleanInputStatic** dans la méthode **addUser** pour nettoyer les entrées utilisateur dans le formulaire. Pour cela nous allons modifier la méthode addUser avec le code ci-dessous :

1 Ajouter la dépendance :

```
use App\Service\Utils ;
```

2 Ajouter le code ci-dessous dans la méthode :

A l'intérieur du else :

```
$pass =
Utils::cleanInputStatic($request->request->all('register')['password']['fir
st']);
$email =
Utils::cleanInputStatic($request->request->all('register')['email']);
$name =
Utils::cleanInputStatic($request->request->all('register')['name']);
$firstname =
Utils::cleanInputStatic($request->request->all('register')['firstname']);
$user->setName($name);
$user->setFirstName($firstname);
$user->setEmail($email);
```

## 1.14 Ajouter la librairie phpmailer au projet Symfony :

Nous allons télécharger et importer la librairie phpmailer à notre projet Symfony en utilisant dans le terminal la commande ci-dessous :

```
composer require phpmailer/phpmailer
```

## 1.15 Création du service Messagerie :

Nous allons créer un nouveau service qui va nous permettre d'envoyer des e-mails. Pour se faire nous allons ajouter un nouveau fichier **src/Service/MessagerieService.php**. Nous avons besoin pour d'un compte de messagerie et de configurer le projet avec :

# TP Sécurité Symfony

1 Ajouter vos informations de connexion dans le fichier **.env** comme ci-dessous :

```
LOGIN_MAIL=monlogin
PASSWORD_MAIL=monpassword
SERVEUR_MAIL=smtp.monserveur.com
PORT_MAIL=465
```

Nous allons modifier ces clés avec les valeurs de notre serveur SMTP (Serveur Mail).

2 Nous allons configurer le fichier config/service.yaml pour lier le fichier **.env** et le service **Messagerie**.

```
App\Service\MessagerieService:
        arguments: ['%env(LOGIN_MAIL)%', '%env(PASSWORD_MAIL)%',
'%env(SERVEUR_MAIL)%', '%env(PORT_MAIL)%']
```

**NB** : Attention à l'indentation.

3 Création de la classe **MessagerieService** ajouter le code ci-dessous :

```
<?php
namespace App\Service;
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;
class MessagerieService{
    private ?string $login;
    private ?string $password;
    private ?string $serveur;
    private ?int $port;
    public function __construct(string $login, string $password, string
$serveur, int $port){
        $this->login = $login;
        $this->password = $password;
        $this->serveur = $serveur;
        $this->port = $port;
    }
    public function sendMail($objet, $content, $addresse){
        //Load Composer's autoloader
        require '../vendor/autoload.php';

        //Create an instance; passing `true` enables exceptions
        $mail = new PHPMailer(true);
        try {
```

# TP Sécurité Symfony

```
            //Server settings
            $mail->SMTPDebug = 0;
            $mail->isSMTP();
            $mail->Host       = $this->serveur;
            $mail->SMTPAuth   = true;
            $mail->Username   = $this->$login;
            $mail->Password   = $this->password;
            $mail->SMTPSecure = PHPMailer::ENCRYPTION_SMTPS;
            $mail->Port       = $this->port;

            //expéditeur
            $mail->setFrom($login, 'Admin');
            //destinataire
            $mail->addAddress($addresse);
            //Content
            $mail->isHTML(true);
            $mail->Subject = $objet;
            $mail->Body = $content;
            //envoi du mail
            $mail->send();
            return 'Le mail à été envoyé avec succès';
        }
        catch (\Exception $e) {
            return "Erreur Mail :{$mail->ErrorInfo}";
        }
    }
}
```

## 1.16 Modification de la fonction addUser (*UserController*)

Nous allons ajouter l'envoi d'un mail après l'ajout du compte en BDD comme ci-dessous :

```
#[Route('/register', name: 'app_register')]
    public function adduser(EntityManagerInterface $em, UserRepository
$repo,
    UserPasswordHasherInterface $hash, Request $request, MessagerieService
$messagerie): Response
    {
        $msg = "";
        $user = new User();
        $form = $this->createForm(RegisterType::class, $user);
        $form->handleRequest($request);
        if($form->isSubmitted() AND $form->isValid()){
            if($repo->findOneBy(['email'=>$user->getEmail()])){
                $msg ="Le compte : ".$user->getEmail()." existe déja";
```

# TP Sécurité Symfony

```
            }
            else{
                $pass =
Utils::cleanInputStatic($request->request->all('register')['password']['fir
st']);
                $email =
Utils::cleanInputStatic($request->request->all('register')['email']);
                $name =
Utils::cleanInputStatic($request->request->all('register')['name']);
                $firstname =
Utils::cleanInputStatic($request->request->all('register')['firstname']);
                $user->setName($name);
                $user->setFirstName($firstname);
                $user->setEmail($email);
                $hash = $hash->hashPassword($user, $pass);
                $user->setPassword($hash);
                $user->setRoles(["ROLE_USER"]);
                $user->setActivated(false);
                $em->persist($user);
                $em->flush();
                $msg = "Le compte : ".$user->getEmail()." a été ajouté";
                $object = "Activation du compte";
                $content = '<p>Pour activer votre compte veuillez cliquer
sur l\'url ci-dessous</p>
                <a
href="https://localhost:8000/register/activate/'.$user->getId().'>Activatio
n</a>';
                $statut = $messagerie->sendMail($object, $content,
$user->getEmail());
            }
        }
        return $this->render('register/index.html.twig', [
            'message' => $msg,
            'form' => $form->createView(),
        ]);
    }
```

**NB** : Ne pas oublier d'ajouter la dépendance du service (*use App\Service\MessagerieService*).

## 1.17 Ajout de la fonction activeUser :

Nous allons ajouter une fonction qui va activer le compte utilisateur depuis le lien envoyé dans le mail d'activation. Ajouter le code ci-dessous :

# TP Sécurité Symfony

```
#[Route('/register/activate/{id}', name: 'app_register_activate')]
    public function activateUser($id, EntityManagerInterface $em,
UserRepository $repo):Response{
        //récupérer le compte utilisateur
        $user = $repo->find($id);
        //tester si le compte existe
        if($user){
            $user->setActivated(true);
            $em->persist($user);
            $em->flush();
            //redirection vers la connexion
            return $this->redirectToRoute('app_login');
        }
        //test sinon le compte n'existe pas
        else{
            //redirection vers l'inscription
            return $this->redirectToRoute('app_register');
        }
    }
```

Cette méthode va s'exécuter quand l'utilisateur va cliquer sur le lien du mail.

## 1.18 Ajout de la méthode sendMailActivate :

Nous allons ajouter une nouvelle méthode qui va renvoyer le mail d'activation si le compte n'est pas activé, au moment de la connexion. Nous allons utiliser le code ci-dessous :

```
//fonction qui envoi le mail d'activation
    #[Route('/sendMail/activate/{id}', name:'app_send_activate')]
    public function sendMailActivate(Utils $utils,
    Messagerie $messagerie, UserRepository $repo,$id):Response{
        //nettoyage de l'id
        $id = $utils->cleanInput($id);
        //récupération des identifiant de messagerie
        $login = $this->getParameter('login');
        $mdp = $this->getParameter('mdp');
        //variable qui récupère l'utilisateur
        $user = $repo->find($id);
        if($user){
            $objet = 'activation du compte';
            $content = '<p>Pour activer votre compte veuillez cliquer ci-
dessous :</p><a
href="localhost:8000/register/activate/'.$id.'">Activer</a>';
            //on stocke la fonction dans une variable
```

# TP Sécurité Symfony

```
            $status = $messagerie->sendEmail($login, $mdp, $objet,
$content, $user->getEmail());
            //rediriger vers la fonction login et affichage de l'erreur le
compte n'est pas activé
            return $this->redirectToRoute('app_login', ['error'=>1]);
        }
        else{
            return new Response('Le compte n\'existe pas', 200, []);
        }
    }
```

## 1.19 Modification de la méthode onAuthenticationSuccess :

Nous allons modifier la classe Security en ajoutant un attribut, modifiant le constructeur et la méthode onAuthenticationSuccess.

1 Nous allons ajouter le **UserRepository** en dépendance :

```
use App\Repository\UserRepository;
```

2 ajouter un attribut repository :

```
private $repo;
```

3 Modification du constructeur de la classe :

```
public function __construct(private UrlGeneratorInterface $urlGenerator,
UserRepository $UserRepo)
    {
        $this->repo = $UserRepo;
    }
```

4 Modification de la méthode onAuthenticationSuccess :

```
public function onAuthenticationSuccess(Request $request, TokenInterface
$token, string $firewallName): ?Response
    {
        if ($targetPath = $this->getTargetPath($request->getSession(),
$firewallName)) {
            return new RedirectResponse($targetPath);
        }
        //récupérer le mail
        $email = $request->request->get('email', '');
        //récupérer le compte
        $recup = $this->repo->findOneBy(['email'=>$email]);
        //tester si l'utilisateur est activé
        if($recup->isActivated()){
```

# TP Sécurité Symfony

```
            return new
RedirectResponse($this->urlGenerator->generate('app_home'));
        }
        //envoyer vers la méthode envoi un mail d'activation
        else{
            return new
RedirectResponse($this->urlGenerator->generate('app_send_activate',
['id'=>$recup->getId()]));
        }
        // For exemple:

        // return new
RedirectResponse($this->urlGenerator->generate('some_route'));
        //throw new \Exception('TODO: provide a valid redirect inside
'.__FILE__);
    }
```