DOCUMENTATION TECHNIQUE POUR DEVELOPPEUR JUNIOR SYMFONY

PARTIE 1 : UPGRADE VERS SYMFONY 3.4

Étape 1 :Créer le dossier

Commencez par créer un dossier puis à y cloner le repository de l'application en tapant dans le CLI de votre IDE : git clone https://github.com/saro0h/projet8-TodoList.git

Étape 2 : Installer les composants nécessaires

Puis, placez vous à la racine du projet.
Tapez : composer install
Tous les composants nécessaires au fonctionnement de l'application web vont s'installer.

Étape 3 : Faire fonctionner le server interne de Symfony

Ensuite tapez : symfony serve
Vous devez voir ce message apparaitre :
[OK] Web server listening
The Web server is using PHP FPM 7.3.19
http://127.0.0.1:8000

Étape 4 : Analyser quelle version de Symfony est utilisé

Rendez vous sur cette adresse http://127.0.0.1:8000
On se retrouve sur la page suivante : http://localhost:8000/login
Dans le profiler, on peut voir la version de Symfony utilisé par l'application web.
Il s'agit de la version 3.1.6.

Étape 5 : Upgrade Symfony vers la version 3.4

Commençons par upgrade cette version vers la version 3.4
A la racine du projet se trouve Composer.Json . Ouvrez le et dans la section require : Changez la version de Symfony/symfony par : 3.4.*
Puis dans le CLI, tapez : composer update

PARTIE 2 : CORRECTION DES DÉPRÉCIATIONS

Étape 6 : Analyser la nouvelle version de Symfony

Redemarrez le server et rendez vous sur http://localhost:8000/login
Dans le profiler, on peut voir que la version de symfony utilisé est désormais la 3.4.42
Notre but est d'arriver à la version 4 de Symfony.
Mais pour cela, nous devons corrigez les dépreciations.

Étape 7 : Allez dans la liste des dépréciations

Dans la barre du profiler de Symfony, cliquez sur le bouton pour voir la liste des dépréciations. Il y en a 11.

Étape 8: Correction de dépréciation

Dépréciation : User Deprecated: Symfony\Component\HttpKernel\Kernel::loadClassCache() is deprecated since Symfony 3.3, to be removed in 4.0.

On peut voir en cliquant sur Trace que la fonction qui est dépréciée se trouve dans web/app_dev.php .
Ouvrez le fichier et supprimer la ligne 26 :

```
$kernel->loadClassCache();
```

Ouvrez aussi web/app.php et supprimez la ligne 10 :

```
$kernel->loadClassCache();
```

Si l'on refresh la page, on peut voir qu'il n'y a plus que 9 dépréciations.

Étape 9 : Correction de dépréciation

Dépréciation : Using the unquoted scalar value "**!event**" is deprecated since Symfony 3.3 and will be considered as a tagged value in 4.0. You must quote it in "**/Users/jvjlondon/Desktop/SoutenanceP8/projet8-TodoList/app/config/config_dev.yml**" on line 20.

Pour corriger celle ci, rendez vous dans app/config/config_dev.yml et ajoutez des « » autour des valeurs des lignes 20 et 23 comme ceci :

```
    channels: ["!event"]
console:
    type:   console
    channels: ["!event", "!doctrine"]
```

Étape 10 : Correction de dépréciation

Dépréciations : The "**framework.trusted_proxies**" configuration key has been deprecated in Symfony 3.3. Use the Request::setTrustedProxies() method in your front controller instead.

Pour corriger, celle ci rendez vous dans config/config.yml et supprimez la ligne 26 :

```
trusted_proxies: ~
```

Étape 11 : Correction de dépréciation

Dépréciations :

Not setting "**logout_on_user_change**" to true on firewall "**main**" is deprecated as of 3.4, it will always be true in 4.0.

Pour corriger celle ci, allez dans app/config/security.yml et ajoutez cette ligne la dans la section main :

```
logout_on_user_change: true
```

Cela devrait être la ligne 17.

Étape 12 : Update des versions de certains composants

Il reste 5 dépréciations

Pour corriger une partie d'entre elles, nous allons nous rendre sur cette adresse et comparer notre composer.json à celui de la version standard de Symfony 3.4

https://github.com/symfony/symfony-standard/blob/3.4/composer.json

Donc, nous allons changer ces bundles suivants :

```
"symfony/swiftmailer-bundle": "^2.6.4"
"symfony/monolog-bundle": "^3.1.0",
"sensio/framework-extra-bundle": "^5.0.0",
"sensio/distribution-bundle": "^5.0.19",
```

Ensuite à la racine du projet, dans le CLI, tapez : composer update

Étape 13 : Correction de dépréciation

Plusieurs nouvelles dépréciations sont apparues, mais en contrepartie nous sommes à jour dans la version de nos bundles.

Voyons une autre dépréciations :

User Deprecated: The "**Sensio\Bundle\FrameworkExtraBundle\Configuration\Route**" annotation is deprecated since version 5.2. Use "**Symfony\Component\Routing\Annotation\Route**" instead.

Pour corriger celle-ci, rendez vous dans src/AppBundle/Controller et remplacez dans tous les controller :

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
```

par :

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

Étape 14 : Correction de dépréciation

Voyons une autre dépréciations :

The "**framework.trusted_proxies**" configuration key has been deprecated in Symfony 3.3. Use the Request::setTrustedProxies() method in your front controller instead.

Pour la corriger, rendez vous dans un premier temps dans app/config/config.yml et supprimez la ligne 26 :

```
trusted_proxies: ~
```

Puis allez dans web/app.php et ajoutez la ligne suivante :

Request::setTrustedProxies()

en dessous de la ligne 9 :

```
$kernel = new AppKernel('prod', false);
Request::setTrustedProxies()
```

Étape 15 : Correction de dépréciation

Voyons une autre dépréciations :

User Deprecated: Creating Doctrine\ORM\Mapping\UnderscoreNamingStrategy without making it number aware is deprecated and will be removed in Doctrine ORM 3.0.

Pour corriger celle-ci, rendez vous dans app/config/config.yml et changez la ligne 60 par :

```
naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware
```

Étape 16 : Correction de dépréciation

Voyons la dernière dépréciation :

> Enabling the "**sensio_framework_extra.router.annotations**" configuration is deprecated since version 5.2. Set it to false and use the "**Symfony\Component\Routing\Annotation\Route**" annotation from Symfony itself.

Pour la corriger rendez vous dans app/config/config.yml et ajoutez ces lignes de codes à la fin :

```
sensio_framework_extra:
   router:
      annotations: false
```

Voila il n'y a plus aucune dépréciations.

Nous allons pouvoir upgrade la version de Symfony à la version 4.

Étape 17 : Installation de symfony 4

Rendez vous dans composer.json et changez dans require :

```
"symfony/symfony": "^4.0",
```

Et aussi dans require-dev :

```
"symfony/phpunit-bridge": "^4.0"
```

Puis supprimer dans require :

```
"incenteev/composer-parameter-handler": "^2.0",
"symfony/swiftmailer-bundle": "",
```

Et supprimer aussi :

```
"sensio/distribution-bundle": "^5.0.19",
```

Et dans require-dev :

Supprimer :

```
"sensio/generator-bundle": "^3.0",
```

Ensuite dans le CLI, depuis la racine du projet tapez : composer update

PARTIE 3 : CONFIGURATION DE L'ARCHITECTURE DE SYMFONY 4

Étape 18 : Reconfiguration avec flex du dossier vendor

Nous sommes maintenant sur Symfony 4 ! Super !
Nous allons pouvoir installer la structure Flex :
Son but ?
Automatiser l'installation et la suppression de vos dépendances en fournissant une **configuration par défaut** sans avoir à aller lire la doc pour trouver quelle configuration écrire, quelles routes charger ou autre tâche rébarbative à effectuer.
Et dès Symfony 4.0, Flex sera le moyen par défaut pour développer une application Symfony.
Commencez par vérifier que votre composer est à la derniere version possible.
Ensuite installez flex : composer require symfony/flex
Vidons désormais notre dossier vendor pour le reremplir avec et configurer sa librairie avec flex.
Dans le CLI, tapez : rm -rf vendor
Puis : composer install

Étape 19 : Suppression/Mise à jour de packages et changements dans composer.json

Rendez vous dans composer.json et dans require supprimez :
Symfony/symfony

Et ajoutez :

```
"php": "^7.2",
"symfony/console": "^4.0",
"symfony/framework-bundle": "^4.0",
"symfony/lts": "^4@dev",
"symfony/yaml": "^4.0"
```

Puis dans require-dev, ajoutez :

```
"symfony/dotenv": "^4.0",
```

Puis tout en haut de composer.json, ajoutez les lignes suivantes :

```
"config": {
    "preferred-install": {
        "*": "dist"
    },
    "sort-packages": true
},
```

Puis remplacez la section scripts et extra par celles-ci :

```
"scripts": {
    "auto-scripts": {
        "cache:clear": "symfony-cmd",
        "assets:install --symlink --relative %PUBLIC_DIR%": "symfony-cmd"
    },
    "post-install-cmd": [
        "@auto-scripts"
    ],
    "post-update-cmd": [
        "@auto-scripts"
    ]
},
"conflict": {
    "symfony/symfony": "*"
},
"extra": {
    "symfony": {
        "allow-contrib": true
    }
}
```

Remplacez les lignes suivantes aussi , pour autoload et autoload-dev

```
"autoload": {
    "psr-4": {
        "App\\": "src/"
    }
},
"autoload-dev": {
    "psr-4": {
        "App\\Tests\\": "tests/"
    }
},
```

Puis tapez dans le CLI, composer update .

Étape 20 : Mise à jour de bin/console

Allez à l'adresse suivante :
https://github.com/symfony/recipes/blob/master/symfony/console/4.4/bin/console

Puis copiez le contenu et coller le dans notre projet dans bin/console .

Étape 21 :Installation de package validator et form

Nous allons migrer tous le contenu de app/ vers la nouvelle strucuture symfony4 petit à petit.
On peut s'occuper de nouveaux composants pour commencer avec les formulaires et validator
Tapez dans le CLI :
composer require form
et
composer require validator
Grâce à flex tout est déjà configurer pour nous.
Passons à la suite.
Nous allons installer le composant de sécurité pour crsf_protection
Tapez dans le CLI :
composer require security_csrf

Étape 22 : Installation de package twig

Ensuite occupons nous de twig :
Composer require twig
Un nouveau dossier template a été créer.
Migrons toutes nos anciennes vues dans ce dossier.
Pour cela, allez dans app/Resources/view, copiez tous les dossiers et collez les dans template et overwrite le base.html.twig .
Vous pouvez ensuite supprimer le dossier Resources.

Étape 23 : Installation de asset

Dans config/framework.yaml
Changez la ligne 14 par celle-ci :

```
fragments: ~
```

Puis voici le tour des asset, tapez dans le CLI :
Composer require asset

Étape 24 : migrations de dossiers dans src et suppression de src/AppBundle

Allez dans src/AppBundle et copiez le contenu des dossiers Controller et Entity pour le copier dans src/Controller et src/Entity.
Bougez le dossier Form aussi dans src/. Puis supprimez src/AppBundle.

Étape 25 : Changement des namespaces et correction d'une dépréciation

Changez tous les namespaces et les use contenant AppBundle par App dans les controllers et les entités ainsi que dans le dossier Form
Puis dans les controllers changer la classe extends par AbstractController ainsi que le use par

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

et supprimer celui qui n'est plus utilisé.

Étape 26 : Installation de package security

Migrons ensuite la sécurité en installant le bundle suivant :
Dans le CLI, tapez : composer require security
Copiez le contenu de app/config/security.yml et collez le dans config/security.yaml
En changeant par la suite la ligne 8 :

```
class: App:User
```

Étape 27 : installation du web profiler

On va installer le nouveau web profiler avec la commande suivante, dans le CLI : composer require profiler

Étape 28 : Mise en place de routes.yaml

Dans config/routes.yaml, décommentez les 3 lignes :

```
index:
    path: /
    controller: App\Controller\DefaultController::index
```

Étape 29 : On renseigne la database dans .ENV

Avec Flex, parameters.yaml n'existe pas, on référence plutôt des variables d'environnement.

On passe les variables de dev dans .env .
Renseignez la base de donnée dans le .env :

```
DATABASE_URL=mysql://root:root@127.0.0.1:8889/projet8Todolist?serverVersion=5.7
```

Ici, j'utilise mysql , le user est root, le password root et le host : l27.0.0.1 avec le port 8889 et le nom de la base de donnée todolist.

Étape 30 : Création de la database

Nous allons créer la database, en tapant la commande suivante dans le CLI,
./bin/console doctrine:database:create
Vous devez avoir la réponse suivante qui s'affiche :
Created database `projet8Todolist` for connection named default

Étape 31 : Suppression du dossier app/ , web/ et installation de apache-pack

Supprimez maintenant le dossier app/ .
Dans le dossier var, supprimez tout sauf le dossier cache et log
Ensuite Dans le dossier web/ : prenez css/ , img/, js/ et robots.txt et placez les dans le dossier public/
Supprimez le dossier web/
Puis tapez dans le CLI (si vous utilisez Apache) :
Composer require symfony/apache-pack
Ceci va créer un .htaccess dans le dossier /public

Étape 32 : Vérification des derniers AppBundle et Modification

Vérifions qu'il n'y a plus de AppBundle dans nos fichiers en tapant la commande suivante dans le CLI :

```
git grep AppBundle
```

Dans TaskController et UserController , remplacez le AppBundle par App.
Puis dans le dossiers tests, bouger le dossier Controller à la racine de tests et supprimer le dossier AppBundle.
Le but du dossier tests est de refléter comme un miroir le dossier src/.
Puis dans test/Controller/DefaultControllerTest.php, remplacer AppBundle par App dans le namespace.

Étape 33 : Création des Repository

Rendez vous dans le dossier src/Repository et créons nos deux repository :
UserRepository.php

```
<?php

namespace App\Repository;
```

```php
use App\Entity\User;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @method User|null find($id, $lockMode = null, $lockVersion = null)
 * @method User|null findOneBy(array $criteria, array $orderBy = null)
 * @method User[]    findAll()
 * @method User[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class UserRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, User::class);
    }
}
```

Et TaskRepository.php

```php
<?php

namespace App\Repository;

use App\Entity\Task;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @method Task|null find($id, $lockMode = null, $lockVersion = null)
 * @method Task|null findOneBy(array $criteria, array $orderBy = null)
 * @method Task[]    findAll()
 * @method Task[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class TaskRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Task::class);
    }
}
```

PARTIE 4 : IMPLÉMENTATION DE NOUVELLES FONCTIONNALITÉS

Étape 34 : Installation du système d'authentification et configuration de fichiers

Nous allons maintenant implémenter le nouveau Maker, tapez dans le CLI :
Composer require maker –dev
Nous allons nous servir de ce service pour créer notre système d'authentification, tapez dans le CLI :
Composer require make :auth
Choisissez Empty authenticator, puis appelez l'authenticator AppAuthenticator.
Un dossier et fichier src/Security/appAuthenticator.php vient d'être créer et config/packages/security.yaml vient d'être mis à jour.
Cependant, remplacer le code par celui-ci :

```yaml
security:
    encoders:
        App\Entity\User:
            algorithm: bcrypt

    providers:
        in_memory: { memory: null }
        in_database:
            entity:
                class: App\Entity\User
                property: username

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: lazy
            provider: in_database

            form_login:
                login_path: login
                check_path: login_check
                always_use_default_target_path:  true
                default_target_path:  /tasks

            logout:
                path: /logout
                target: /
                invalidate_session: true
            guard:
                authenticators:
                    - App\Security\AppAuthenticator

    access_control:
        # - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        #- { path: ^/users, roles: ROLE_ADMIN }
        #- { path: ^/tasks, roles: IS_AUTHENTICATED_FULLY }
```

Étape 35 : Modification de la function loginAction

Dans le dossier src/Controller/SecurityController.php
Remplacez totalement la function loginAction par celle-ci :

```
/**
 * @Route("/login", name="login")
 */
public function login(AuthenticationUtils $authenticationUtils): Response
{
    // if ($this->getUser()) {
    //     return $this->redirectToRoute('target_path');
    // }

    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();
    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render('security/login.html.twig', ['last_username' =>
$lastUsername, 'error' => $error]);
}
```

En effet, avec Symfony 4, nous injectons directement la classe :

```
use
Symfony\Component\Security\Http\Authentication\AuthenticationUtils;
```

Étape 36 : Installation du package migration

Pour remplir notre base de données, nous avons besoin de faire des migrations.
Cependant le package n'est pas encore présent.
Tapez dans le CLI, composer require migrations
Puis après l'installation, tapez :
./bin/console make:migration
Puis
php bin/console doctrine:migrations:migrate
Super cela fonctionne

Étape 37 : Installation d'une première nouvelle fonctionnalité : Choix du Rôle du User, modification de l'Entité User

Nous allons faire évoluer l'entité User.
Implémentons une nouvelle fonctionnalité : nous devons être capable, lors de la création d'un utilisateur, de choisir son rôle : Administrateur ou Utilisateur.
Commençons par éditer l'entité User :
Tapez dans le CLI ,
Php bin/console make :entity
Class name of the entity to create or update (e.g. FiercePuppy):
 > User
Ajoutons la propriétée roles :
New property name (press <return> to stop adding fields):
 > roles
Renseignez le field type : array
Field type (enter ? to see all types) [string]:
 > array
Et repondez que le field peut être null par : yes
Can this field be null in the database (nullable) (yes/no) [no]:
 >yes
Puis Migrer ceci dans la base de donnée avec les commandes :
php bin/console make:migration
et
php bin/console doctrine:migrations:migrate

Remplacez dans src/Entity/User, le get et set de Roles par cela :

```php
public function getRoles()
{
    $roles = $this->roles;
    $roles[] = 'ROLE_USER';
    return array_unique($roles);
}
```

et

```php
public function setRoles(array $roles): self
{
    $this->roles = $roles;

    return $this;
}
```

Cela sert à attribuez le rôle User par défaut à un utilisateur, si il n'a pas de rôle déjà attribué.

Étape 38 : Suite de l'installation de la première nouvelle fonctionnalité, modification du formulaire

Allons éditer maintenant le formulaire de création de User et implémentons la nouvelle fonctionnalité :

```php
<?php

namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;
use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\CollectionType;

class UserType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('username', TextType::class, ['label' => "Nom d'utilisateur"])
            ->add('password', RepeatedType::class, [
                'type' => PasswordType::class,
                'invalid_message' => 'Les deux mots de passe doivent correspondre.',
                'required' => true,
                'first_options'  => ['label' => 'Mot de passe'],
                'second_options' => ['label' => 'Tapez le mot de passe à nouveau'],
            ])
            ->add('email', EmailType::class, ['label' => 'Adresse email'])
            ->add('roles',  CollectionType::class, [
                                'label' => false,
                                'entry_type'   => ChoiceType::class,
                                'entry_options' => [
                                    'label' => 'Définir le role',
                                    'choices' =>  [
                                        'Administrateur' => 'ROLE_ADMIN',
                                        'Utilisateur' => 'ROLE_USER',
                                    ],
                                ],
            ]);
    }
}
```

Remplacez le code par celui-ci, On peut voir que nous utilisons le CollectionType dans lequel nous implémentons un ChoiceType.

Allons tester ce code et la création d'un utilisateur.

Étape 39 : Suite de l'installation de la première nouvelle fonctionnalité, Correction d'une dépréciation

Nous allons commencer par corriger une dépréciation, rendez vous dans src/Controller/UserController et remplacez le code suivant à la ligne 40 et 56 :

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

Puis faites la méme chose dans src/Controller/TaskController et remplacez le code à la ligne 31 et 54

En effet, avec Symfony 4, nous devons vérifiez que le formulaire a bien été submit.

Étape 40 : Suite de l'installation de la première nouvelle fonctionnalité, injection de UserPasswordEncoderInterface

Rendez vous dans src/Controller/UserController.

Nous devons injecter l'encoder dans le code et ne plus utiliser le service :

```
security.password_encoder
```

Remplacez la ligne 33 par celle-ci :

```
$password = $encoder->encodePassword($user, $user->getPassword());
```

Et ajoutez ceci dans les paramètres de la fonction :

```
UserPasswordEncoderInterface $encoder
```

Ce qui doit donner ceci :

```
public function createAction(Request
$request,UserPasswordEncoderInterface $encoder)
```

ajoutez ensuite :

```
use
Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterf
ace;
```

Étape 41 : Suite de l'installation de la première nouvelle fonctionnalité, amélioration du formulaire

Tapez dans le CLI : symfony serve

Et rendez vous ici : http://localhost:8000/users/create

Nous allons transformer le formulaire pour le rendre plus joli.

Rendez vous dans config/packages/twig.yaml et ajoutez cette ligne :

```
form_themes: ['bootstrap_4_layout.html.twig']
```

Puis rendez vous dans templates/user/create.html.twig et ajoutez cette ligne juste après la première ligne :

```
{% form_theme form 'bootstrap_4_layout.html.twig' %}
```

Refresh la page et admirez le travail.
Ce formulaire est bien plus joli. Super. Testons le et créons un administrateur.Tout fonctionne : **Superbe !** L'utilisateur a bien été ajouté.

Étape 42 : Suite de l'installation de la première nouvelle fonctionnalité, mise en place de la modification dans edit

Nous allons maintenant customiser la fonction create et y incorporer la possibilité de modifier un utilisateur existant, remplacer le code de la fonction par celui-ci :

```
/**
 * @Route("/users/create", name="user_create")
 * @Route("/users/{id}/edit", name="user_edit")
 */
public function createAction(User $user = null, Request
$request,UserPasswordEncoderInterface $encoder)
{
    if (!$user) {
        $user = new User();
    }

    $form = $this->createForm(UserType::class, $user);
    $form->handleRequest($request);

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

        $em = $this->getDoctrine()->getManager();
        $password = $encoder->encodePassword($user, $user->getPassword());
        $user->setPassword($password);
        $em->persist($user);
        $em->flush();

        if (!$user->getId()) {
            $this->addFlash('success', "L'utilisateur a bien été ajouté.");
        } else {
            $this->addFlash('success', "L'utilisateur a bien été modifié.");
        }

        return $this->redirectToRoute('user_list');
    }

    return $this->render('user/create.html.twig', [
        'form' => $form->createView(),
        'editMode' => $user->getId() !== null
    ]);
}
```

Vous pouvez supprimer du coup, la fonction editAction qui n'est plus d'actualité ainsi que la vue dans templates/user/edit.html.twig

Étape 43 : Suite de l'installation de la première nouvelle fonctionnalité,Mise en place de changement dans le rendu du formulaire dans le fichier twig

Nous allons devoir éditer la vue templates/user/create.html.twig, afin qu'elle puisse répondre aux deux éventualités : création ou modification d'utilisateur.

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

{% form_theme form 'bootstrap_4_layout.html.twig' %}

{% block header_title %}
   {%  if editMode %}
      <h1>Modifier un utilisateur</h1>
   {%  else  %}
      <h1>Créer un utilisateur</h1>
   {%  endif  %}
{% endblock %}
{% block header_img %}{% endblock %}

{% block body %}
   <div class="row">

      {{ form_start(form) }}

      {{ form_row(form.username) }}
      {{ form_row(form.email) }}
      {{ form_row(form.password) }}
      {{ form_row(form.roles.0) }}
      {{ form_row(form._token) }}

      <button type="submit" class="btn btn-success">
         {%  if editMode %}
            Enregistrer les modifications
         {%  else  %}
            Ajouter
         {%  endif  %}
      </button>
      {{  form_end(form, {'render_rest': false}) }}


   </div>
{% endblock %}
```

Étape 44 : Installation de la seconde nouvelle fonctionnalité : Création d'un back office avec gestion des utilisateurs, réservé aux ROLE_ADMIN.

Nous allons ensuite ajouter l'interdiction de se rendre dans ce formulaire, ainsi que vers la liste des utilisateurs si l'utilisateur n'a pas le rôle Admin.

On se rend dans src/Controller/UserController et ajoutons dans les use :

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
```

Puis dans nos annotations :

```
/**
 * @Route("/users", name="user_list")
 * @IsGranted("ROLE_ADMIN")
 */
```

Et

```
/**
 * @Route("/users/create", name="user_create")
 * @Route("/users/{id}/edit", name="user_edit")
 * @IsGranted("ROLE_ADMIN")
 */
```

On remarque que l'on a ajouté :

```
@IsGranted("ROLE_ADMIN")
```

Essayer désormais de vous rendre sur la page de création, modification ou sur la liste des utilisateurs sans être Administrateur et vous serez tout de suite redirigez vers le formulaire de login.

Étape 45 : suite de l'installation de la seconde nouvelle fonctionnalité, création de boutons

Créons un lien pour se rendre sur la page des utilisateur et cachons le bouton de création si l'utilisateur n'est pas admin.
Dans templates/base.html.twig, ajoutons une condition à la ligne 42 :

```
{% if app.user and app.user.roles.0 == 'ROLE_ADMIN' %}
<a href="{{ path('user_create') }}" class="btn btn-primary">Créer un utilisateur</a>
{% endif %}
```

Modifions ici la route et le nom de ce bouton qui n'est pas vraiment adapté.
Remplaçons le par BackOffice Administrateur

Ce qui va donner ceci :

```
{% if app.user and app.user.roles.0 == 'ROLE_ADMIN' %}
<a href="{{ path('users') }}" class="btn btn-primary">BackOffice
Administrateur</a>
{% endif %}
```

Puis allez dans template/user/list.html.twig

Et ajoutez au début du block body, la ligne suivante :

```
<a href="{{ path('user_create') }}" class="btn btn-primary">Créer un
utilisateur</a>
```

Étape 46 : suite de l'installation de la seconde nouvelle fonctionnalité : on vérifie dans la base de donnée

Les utilisateurs sont stockés dans la base de donnée, on peut observer qu'il y a maintenant seulement un utilisateur et qu'il a le rôle admin :

id	username	password	email	roles (DC2Type:array)
1	Administrateur	$2y$13$m6O7MPkN2JIan.Owar6ju.DefNpvaIGNAzlMo74ckSc...	admin@admin.test	a:1:{i:0;s:10:"ROLE_ADMIN";}

Étape 47 : suite de l'installation de la seconde nouvelle fonctionnalité , création d'un bouton

Ajoutons un bouton dans le backoffice pour se rendre vers la liste des tâches. Aller dans templates/user/list.html.twig et ajouter à la suite de la ligne 8, le code suivant :

```
<a href="{{ path('task_list') }}" class="btn btn-info">Consulter la liste des
tâches à faire</a>
```

Étape 48 : Installation d'une troisième nouvelle fonctionnalité : Rattaché une tâche lors de sa création, à un utilisateur.

Ajoutons une nouvelle fonctionnalité : à la sauvegarde de la tâche, l'utilisateur authentifié doit être rattaché à la tâche nouvellement créée.

Commençons par ajouter un lien entre l'entité User et l'entité Task.
Nous allons faire cela grâce au maker de symfony et ensuite migrer cela vers la base de donnée, afin que la clef étrangère sois mise en place.

Dans le CLI, tapez ./bin/console make :entity

Class name of the entity to create or update (e.g. GrumpyElephant):
> User

New property name (press <return> to stop adding fields):
> tasks

Field type (enter ? to see all types) [string]:
> relation

What class should this entity be related to?:
> Task

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> OneToMany

New field name inside Task [user]:
>

Is the Task.user property allowed to be null (nullable)? (yes/no) [yes]:
>

Puis tapez dans le CLI :
php bin/console make:migration
et ensuite
php bin/console doctrine:migrations:migrate

Super. src/Entity/User et src/Entity/Task ont été mis à jour, ainsi que la base de donée.

Étape 49 : Installation d'une troisième nouvelle fonctionnalité : Modification de createAction()

Modifions ensuite le fichier src/Controller/TaskController, afin de renseigner lors de la création d'une tâche, l'id de l'auteur.

```
/**
 * @Route("/tasks/create", name="task_create")
 */
public function createAction(Request $request)
{
    $task = new Task();
    $form = $this->createForm(TaskType::class, $task);
    $user = $this->getUser();

    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $task->setUser($user);
        $em->persist($task);
        $em->flush();

        $this->addFlash('success', 'La tâche a été bien été ajoutée.');

        return $this->redirectToRoute('task_list');
    }

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

Testons notre code et rendez vous sur http://localhost:8000/tasks/create

Vérifions dans notre base de donnée ! Super cela fonctionne bien !

Étape 50 : Ajout de la quatrième nouvelle fonctionnalité concernant la suppression des tâches

Ajoutons une nouvelle fonctionnalité : Les tâches ne peuvent être supprimées que par les utilisateurs ayant créé les tâches en question et Les tâches rattachées à l'utilisateur "anonyme" peuvent être supprimées uniquement par les utilisateurs ayant le rôle administrateur (*ROLE_ADMIN*).

Tout d'abord, ajoutons :

```
@IsGranted("ROLE_USER")
```

Dans src/Controller/TaskController.php, à toutes les fonctions afin de bloquer l'accès à cette partie du site aux personnes qui ne sont pas authentifiés en tant qu'utilisateur.Puis modifions le fichier templates/task/list.html.twig pour implémenter les nouvelles fonctionnalités

Ajoutons des conditions au niveau du bouton supprimer :

```
{% if (task.user is not null and task.user == app.user) or (task.user is null and
app.user.roles.0 == 'ROLE_ADMIN') %}
   <form action="{{ path('task_delete', {'id' : task.id }) }}">
      <button class="btn btn-danger btn-sm pull-right">Supprimer</button>
   </form>
{% endif %}
```

Puis modifions le fichier src/Controller/TaskController et deleteTaskAction()

Remplacez le code par celui-ci :

```
/**
 * @Route("/tasks/{id}/delete", name="task_delete")
 *  @IsGranted("ROLE_USER")
 */
public function deleteTaskAction(Task $task)
{
   $user = $this->getUser();
   $arrayRoles = $user->getRoles();
   $roles0 = $arrayRoles[0];


   $em = $this->getDoctrine()->getManager();
   if ($task->getUser() == $user) {
      $em->remove($task);
      $em->flush();
      $this->addFlash('success', 'La tâche a bien été supprimée.');
   } elseif ($task->getUser() == null && $roles0 == "ROLE_ADMIN") {
      $em->remove($task);
      $em->flush();
      $this->addFlash('success', 'La tâche a bien été supprimée.');
   }else{
      $this->addFlash('error', 'Vous ne pouvez pas supprimer la tâche.');
   }

   return $this->redirectToRoute('task_list');
}
```

PARTIE BONUS : RAPPEL DES RÈGLES À RESPECTER

1)Rappel : Les grands principes du développement d'application

SOLID : c'est un acronyme reprenant 5 grands principes de la programmation orientée objet.

S : **S**ingle responsibility principle ou responsabilité unique : une classe (fonction ou méthode) ne doit avoir **qu'un et unique rôle**.

O : **O**pen/closed principle ou ouvert/fermé : une classe doit être **ouvert à l'extension mais pas à la modification**. En somme, on peut faire évoluer une classe en ajoutant de nouvelles méthodes/propriétés. Mais on ne modifie pas la structure existante, au risque de provoquer des erreurs dans l'utilisation de cette même classe.

L : **L**iskov substitution principle ou substituion de Liskov : imaginons une classe *Véhicule* et une sous-classe *Voiture,* alors tout objet *Véhicule* peut être remplacé par un objet de type *Voiture*. C'est le principe même de l'héritage.
Lorsque nous traitons avec un **objet parent**, nous devons être capable de pouvoir le remplacer avec un **objet enfant** sans en altérer le comportement.

I : **I**nterface segregation principle ou ségrégation des interfaces : ce principe peut sensiblement ressembler au principe de responsabilité unique. On sait qu'une classe doit avoir une responsabilité unique. Le principe de ségrégation des interfaces permet quant à lui de connaitre **les liens de dépendances qui doivent se faire.**
Une classe ne doit pas dépendre d'interface dont elle n'a pas l'utilité, **mais de limiter au maximum l'utilisation d'interface à son strict minimum**.

D : **D**ependency inversion principle ou inversion de dépendances : ce principe est certainement mon préféré car ça parle d'un sujet que j'adore, **l'abstraction**. Et pour cela, j'adore parler de généricité. Le principe est simple mais parfois difficile à mettre en oeuvre, l'idée est qu'une classe ne doit pas dépendre d'une autre classe, mais plutôt dépendre des abstractions. Encore une fois, nous aurons tout le temps de voir ça dans un cas concret plus tard.

2)Rappel : Les bonnes pratiques de Symfony 4

CONFIGURATION

1)Respecter les standards php, c'est-à-dire les PSR.

2)Respecter les variables d'environnements présentes dans le .env

Exemple : Dans Symfony 3, les paramètres d'environnements sont dans parameters.yml. Tandis que dans Symfony 4, on travaille avec des variables d'environnement.

Pour gérer les variables d'environnements, un nouveau composant Dotenv a été créé. Consulter sa documentation pour approfondir les connaissances dessus.

3)WebFrontController unifié.

Avec Symfony4, on utilise seulement un webfrontcontroller => public/index.php

Contrairement à Symfony3, où il y avait 2 WebFrontController.

4)Utiliser les variables secrets pour les informations sensibles

5)Utiliser des paramètres pour la configuration d'application
Définissez ces options en tant que paramètres dans le fichier config / services.yaml. Vous pouvez remplacer ces options par environnement dans les fichiers config / services_dev.yaml et config / services_prod.yaml.

6) Utiliser des constantes pour définir des options qui changent rarement

LOGIQUE MÉTIER

7) Ne créez aucun bundle pour organiser votre logique d'application

8) Utiliser l'auto-writing pour automatiser la configuration des services d'application.
Le l'auto-writing des services est une fonctionnalité qui lit les indications de type sur votre constructeur (ou d'autres méthodes) et transmet automatiquement les services appropriés à chaque méthode, ce qui rend inutile la configuration explicite des services et simplifie la maintenance de l'application.

9) Utiliser le format YAML pour configurer vos propres services. YAML est le format recommandé pour configurer les services car il est convivial pour les nouveaux arrivants et concis, mais Symfony prend également en charge la configuration XML et PHP.

10) Utiliser des annotations pour définir le mappage d'entité de doctrine

CONTROLLER

11) Faites étendre vos controller par : AbstractController

12) Utiliser des annotations pour configurer Routing,Caching et Security

13) Utiliser l'injection de dépendances pour utiliser des services

FORMULAIRE

14) Définir les formulaires comme des classes PHP

15) Ajouter les boutons de soumission de formulaires dans les templates

SECURITY

17) Définir un seul firewall

18) Utiliser le auto Password Hasher

19) Utiliser les Voters pour implémenter des sécurités de restrictions et utiliser l'annotation @Security