

**ToDo&Co**

**Documentation technique: l‘implementation de l’authentification (Symfony 6)**

**Auteur:** ChingYi, Pelgrims Chen

**Dernière mise à jour:** 14/03/2022

**Version:**1.0

**Sommaire**

[Contexte 1](#_Toc13095)

[à quoi sert ce document ? 1](#_Toc30531)

[Introduction 1](#_Toc6901)

[Qu'est-ce que l'authentification ? 1](#_Toc31357)

[Security.yaml : 1](#_Toc27651)

[le fichier de configuration de la sécurité 1](#_Toc22341)

[Quel est le processus de connexion ? 3](#_Toc17775)

[User.php : création de l’utilisateur 3](#_Toc23758)

[UserService.php : la service pour hacher le mote de passe 6](#_Toc23561)

[LoginController.php : Gérer la formulaire de connexion 6](#_Toc3462)

[login.html.twig : le template du formulaire de connexion 7](#_Toc11765)

[Comment l'application sait-elle si l'utilisateur est connecté et quel est son rôle ? 8](#_Toc9894)

[Controller : 8](#_Toc1164)

[Service : 8](#_Toc20764)

[Template: 9](#_Toc15891)

[Contrôle d'accès ? L'utilisateur avec son rôle spécifique est-il autorisé à faire une telle demande ? 10](#_Toc12005)

[1. Security.yaml 10](#_Toc26169)

[2. Controller 10](#_Toc20497)

[3. Voter 11](#_Toc7039)

[Logout 11](#_Toc13572)

# 

# Contexte

## **à quoi sert ce document ?**

Ce document explique comment l'authentification fonctionne dans le framework Symfony ( Symfony version : 6) et comment elle est appliquée dans notre application: ToDo&Co. L'objectif de ce guide est d'aider tous ceux qui doivent modifier l'application en termes d'authentification à l'avenir. Il est également utile à ceux qui souhaitent mieux comprendre l'authentification dans le framework Symfony avec un exemple concret.

Avec ce guide, vous pourriez :

* comprendre quel(s) fichier(s) il faut modifier et pourquoi ;
* comment s’opère l’authentification ;
* et où sont stockés les utilisateurs.

# Introduction

## Qu'est-ce que l'authentification ?

Elle permet de poser la question "Qui êtes-vous ?" de plusieurs manières. En autrement dit, lorsqu'un utilisateur fait une certaine demande sur l’application, l'application va vérifier s'il est autorisé . La demande n'est-elle accessible qu'à l'utilisateur connecté ? Si l'utilisateur est connecté, quel rôle a-t-il ? La demande n'est-elle accessible qu'à certains rôles ?

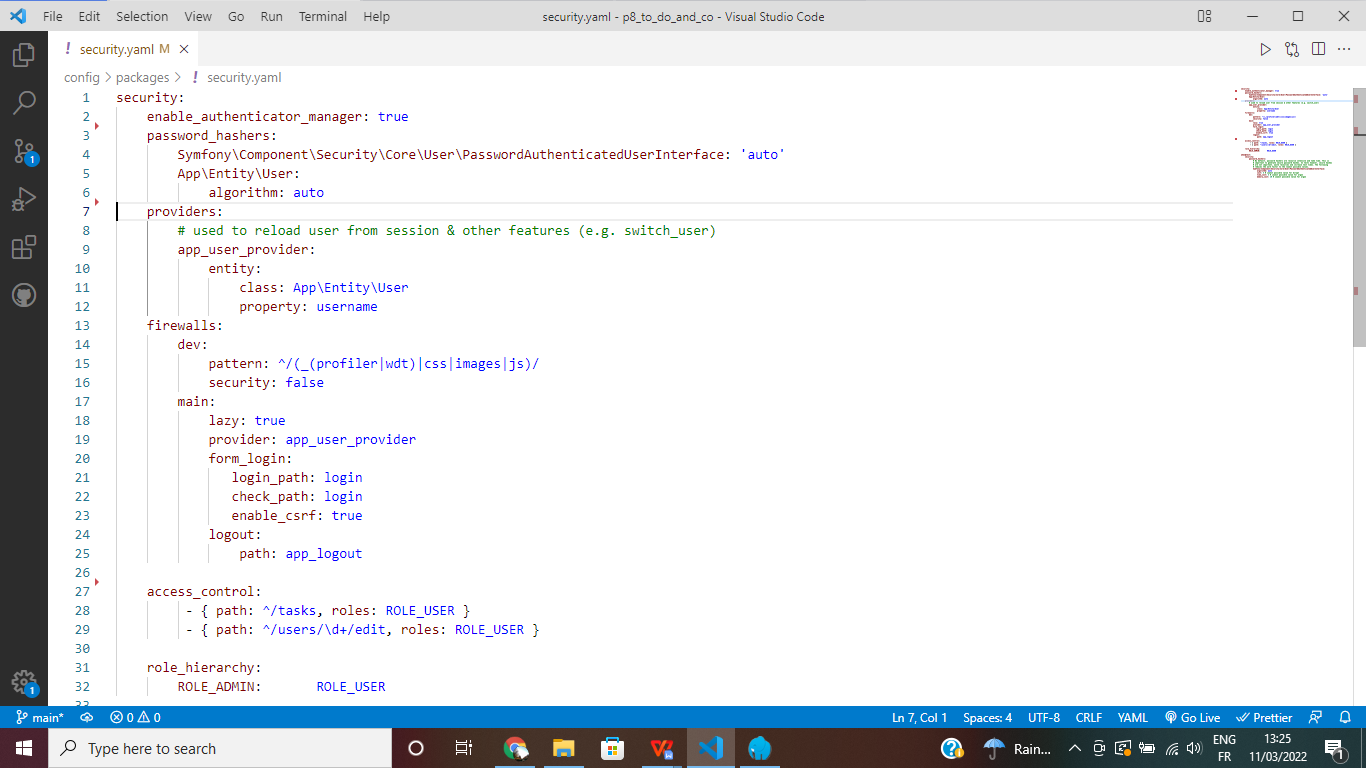
le document l'expliquera sous 3 aspects :

* le processus de connexion
* recherche de l'utilisateur
* autorisation

# Security.yaml :

# le fichier de configuration de la sécurité

Security.yaml est le fichier de configuration de la sécurité de l'application. Dans certains cas, la configuration est générée automatiquement ( vous le comprendrez mieux dans les chapitres suivants), cependant il est important de la comprendre afin de pouvoir la modifier si nécessaire.



**Security (ligne 1):**

enable\_authenticator\_manager : true

Il doit être mis à ‘true’ afin d'autoriser la fonction d'authentification dans l'application.

password\_hashers

Il permet de hacher le mot de passe avant de l'enregistrer dans la base de données et il permet à l'application de "traduire" le mot de passe rempli pendant le processus de connexion et de vérifier s'il correspond au mot de passe haché.

**Providers (ligne7):**

class: App/Entity/User

Le ‘class’ est utilisé pour charger les utilisateurs de la base de données sur la base d'un "user identifier".

property: username

La configuration ci-dessus utilise Doctrine pour charger l'entité User en utilisant la propriété 'username' comme "user identifier". Dans notre cas, l'utilisateur de l'application ‘ToDo&Co’ utilise son nom d'utilisateur (pas son adresse e-mail) et son mot de passe pour se connecter.

**Main (ligne 17):**

Provider: app\_user\_provider

les données fournies par “app\_user\_provider"(voir ligne 9) sont utilisées pour vérifier si l'utilisateur est authentifié.

form\_login

Il existe plusieurs façons pour le système d'essayer de trouver un utilisateur correspondant au visiteur de la page web. ‘ToDo&Co' utilise l'authentificateur de formulaire de connexion (form\_login), c'est-à-dire un moyen traditionnel de fournir un formulaire de connexion où les utilisateurs s'authentifient en utilisant un identifiant (nom d'utilisateur dans notre cas) et un mot de passe.

login\_path:login

check\_path: login

La route nommée 'login' ([voir ici](#_LoginController.php : Gérer la formulaire de connexion)) est utilisée pour la connexion et vérifie si l'utilisateur est connecté avec succès.

enable\_csrf: true

il est défini comme "true", de sorte que nous pouvons ajouter des jetons CSRF cachés dans le formulaire de connexion afin d'empêcher les attaques CSRF de connexion.([voir ici](#_login.html.twig : le template du formulaire de connexion))

logout:

path: app\_logout

La route nommée 'app\_logout' est utilisée pour la déconnexion.[(voir ici](#_Logout))

**Access\_control (ligne 27):**

Nous le configurons pour contrôler l'accès à certaines URL et le limiter au le rôle utilisateur (ROLE\_USER).

Pour des cas plus complexes, par exemple, dans notre cas, le contenu de la page de 'consultation des utilisateurs' est affiché différemment en fonction du rôle de l'utilisateur, il est configuré dans le dossier ‘Controller’. (voir ici)

Voter est également utilisé dans certaines URL (dans notre cas, pour la modification et la suppression de tâches) lorsque le rôle de l'utilisateur n'est pas le seul critère permettant de déterminer si l'utilisateur est autorisé à y accéder. ([voir ici](#_Voter))

**Role\_hierarchy (ligne 31):**

ROLE\_AMIN: ROLE\_USER

Ce paramètre spécifie dans notre cas que le rôle administrateur possède aussi le rôle utilisateur.

# Quel est le processus de connexion ?

## User.php : création de l’utilisateur

Les permissions dans Symfony sont toujours liées à un objet utilisateur, donc une classe utilisateur doit être créée pour 'ToDo&Co'. Il s'agit d'une classe qui implémente ‘UserInterface’. Dans notre cas, elle est aussi une entité Doctrine, ,qui nous permet de communiquer avec la base de données en ce qui concerne un utilisateur.

**Comment le créer?**

Grâce au Maker Bundle, il suffit de lancer les commandes :

php bin/console make:user

Les réponses suivantes ont été données lors de la création de l'"Entity: 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

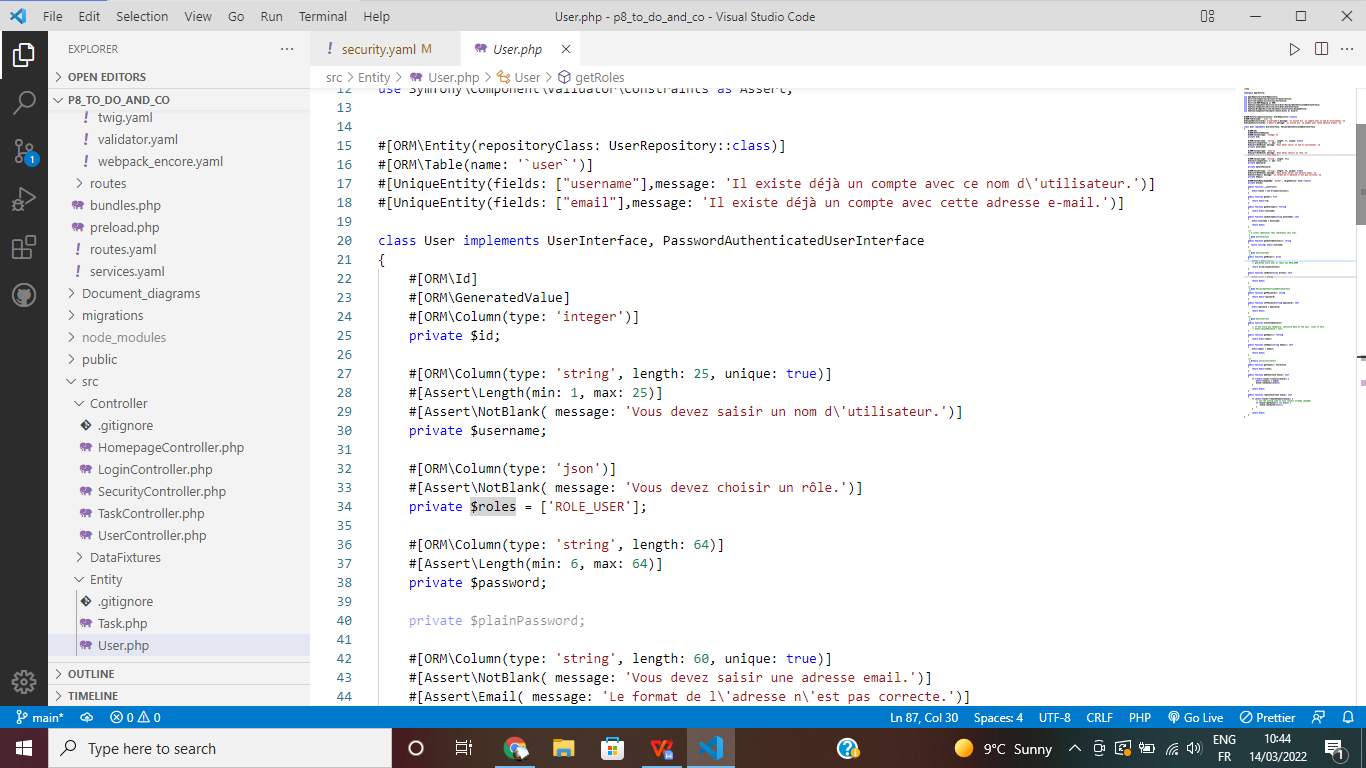
Does this app need to hash/check user passwords? (yes/no) [yes]:

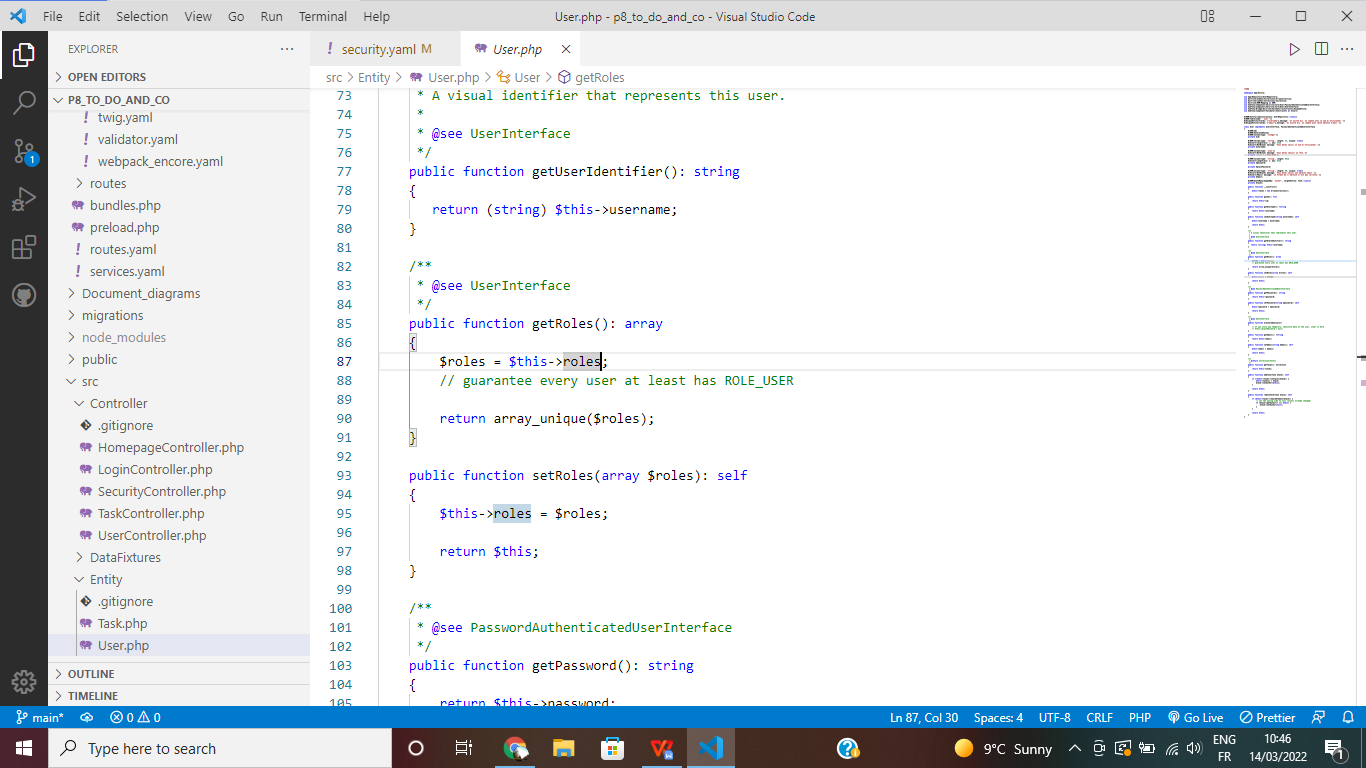
> yes

l'opération nous permet de modifier le security.yaml (e.g. providers, password\_hasher ) et de créer Entity/User.php avec des propriétés et méthodes et correspondant automatiquement à la réponse ci-dessus.

Dans le fichier Entity/User.php, on peut trouver:

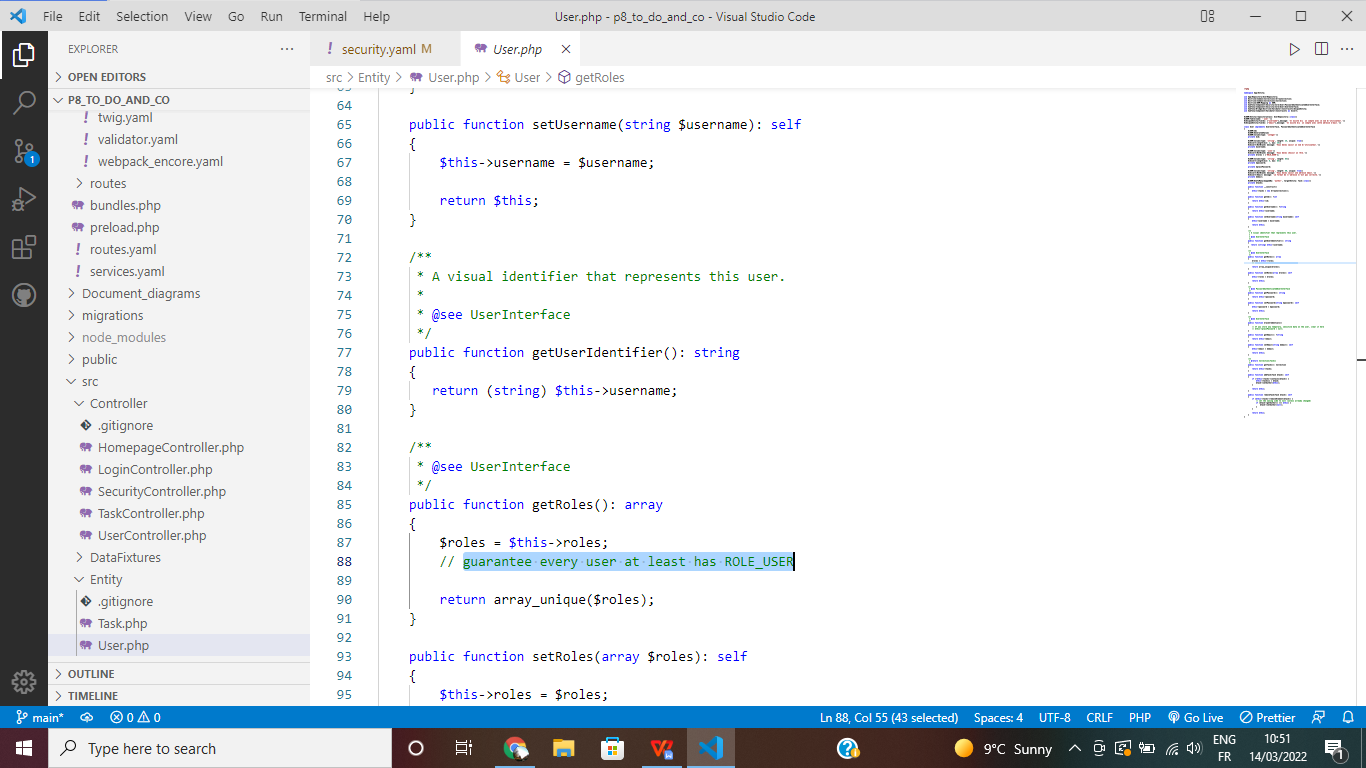
Private $roles= [‘ ROLE\_USER’]





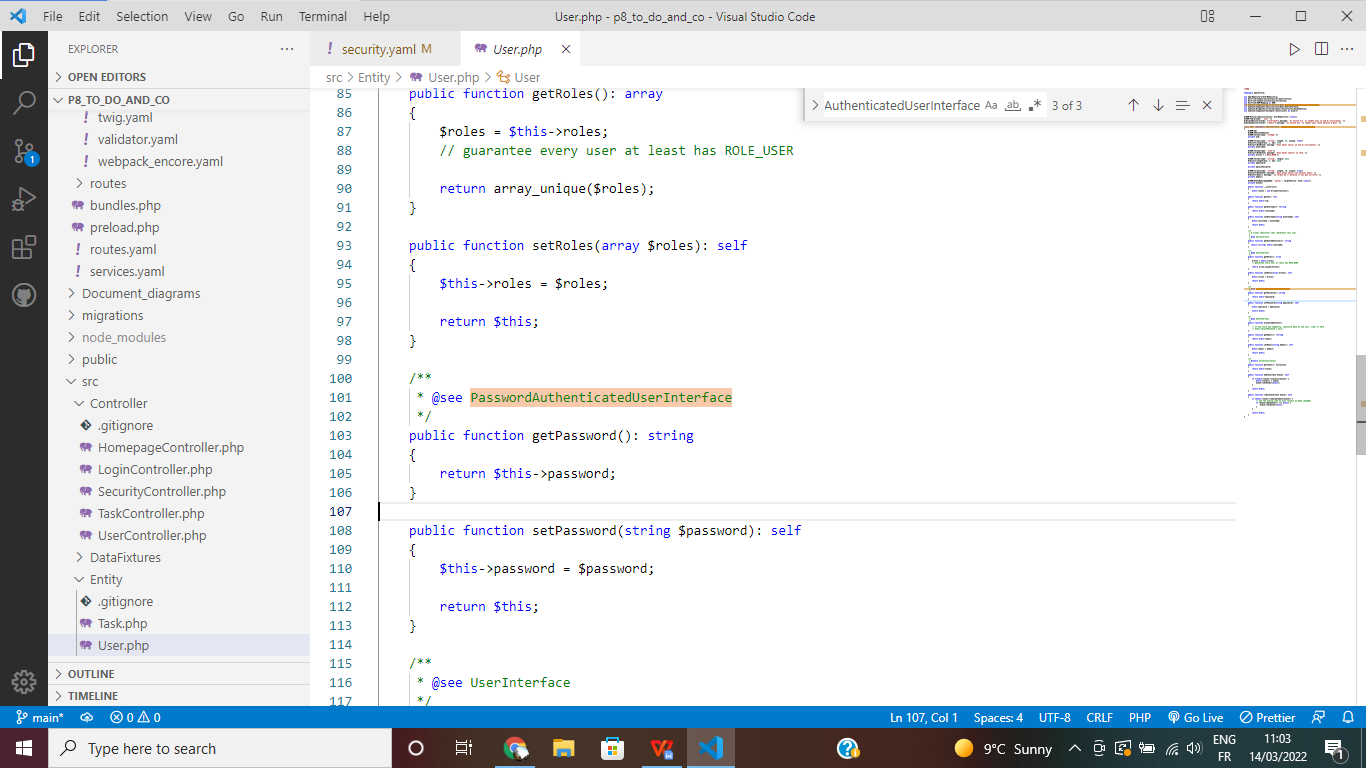
Il est nécessaire de garantir que chaque utilisateur possède au moins ROLE\_USER. Dans notre cas, chaque utilisateur est soit ROLE\_USER soit ROLE\_ADMIN .

Public function getUserIdentifier()



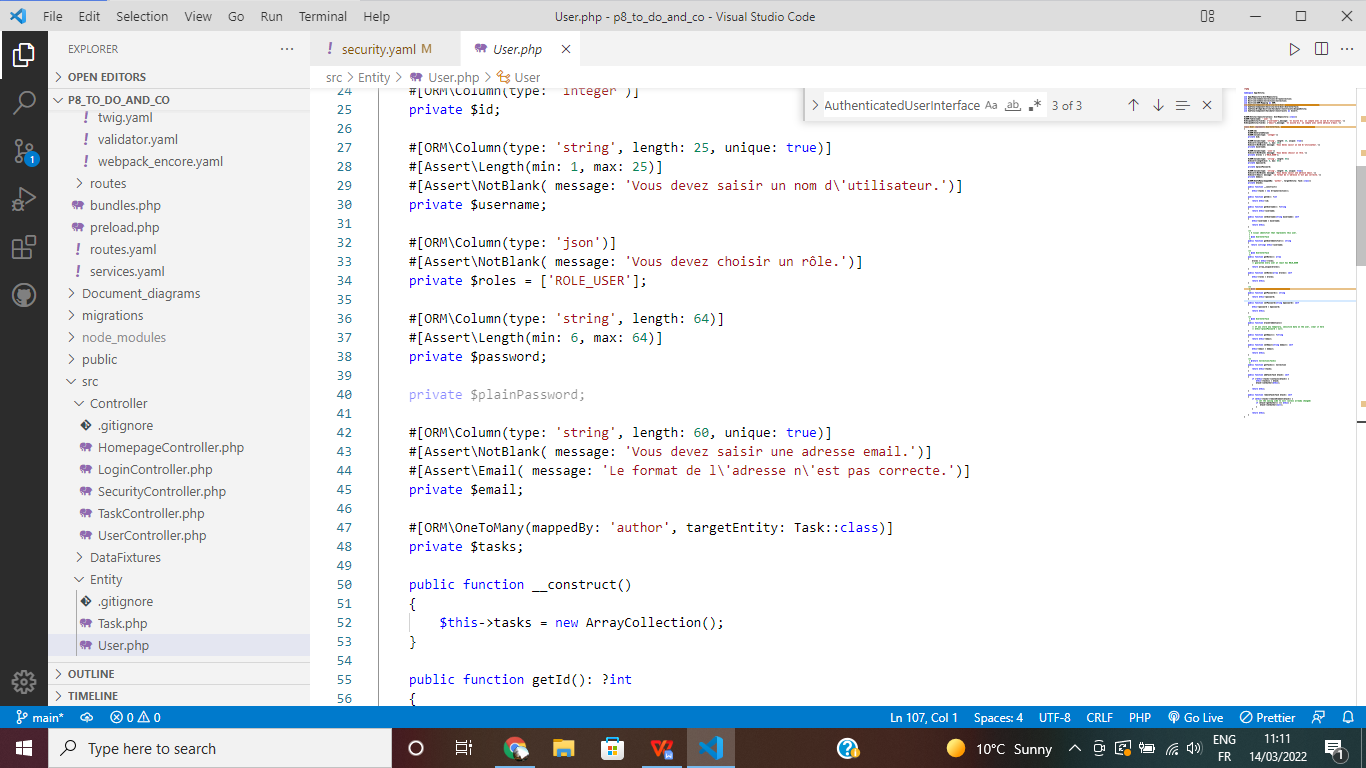
Le nom d'utilisateur est utilisé pour représenter un utilisateur grâce à cette fonction.

PasswordAuthenticaterUserInterface



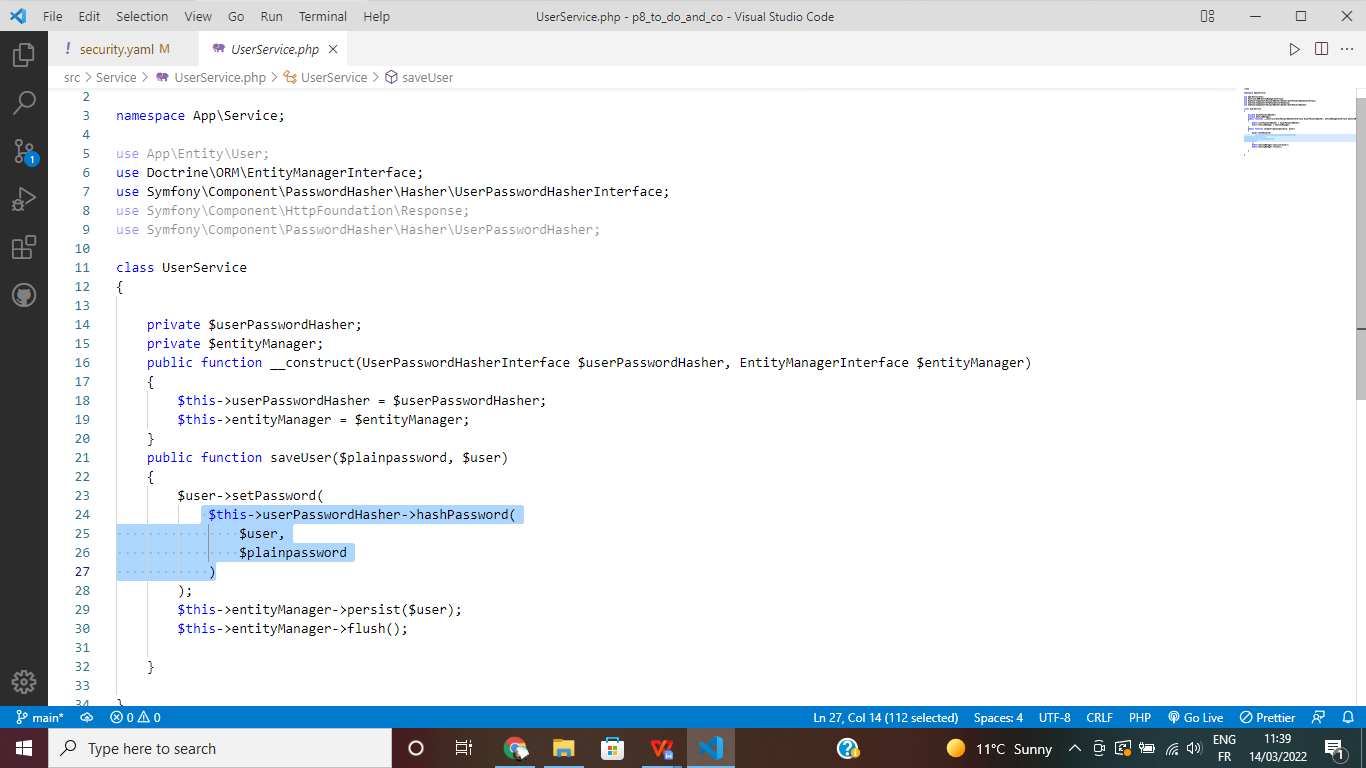
L'interface :PasswordAuthenticaterUserInterface est implémentée afin que getPassword() puisse vérifier le mot de passe de l'utilisateur pour voir si le mot de passe correspond au mot de passe haché.

Private $plainPassword



La propriété $plainPassword est crée et utilisée uniquement pour obtenir le mot de passe du formulaire d'inscription et n’est pas enregistrée dans la base de données. Pour des raisons de sécurité, la propriété $password sert à enregistrer le mot de passe haché dans la base de données. La propriété $plainPassword est utilisée pour obtenir le mot de passe du formulaire d'inscription afin que l'application puisse effectuer le processus de hachage avant de le sauvegarder.

## UserService.php : la service pour hacher le mote de passe



Nous avons créé un service et utilisé l'injection de dépendance pour injecter ‘UserPasswordHasherInterface’ afin que $plainPassword puisse être haché avant d'être enregistré dans la base de données.

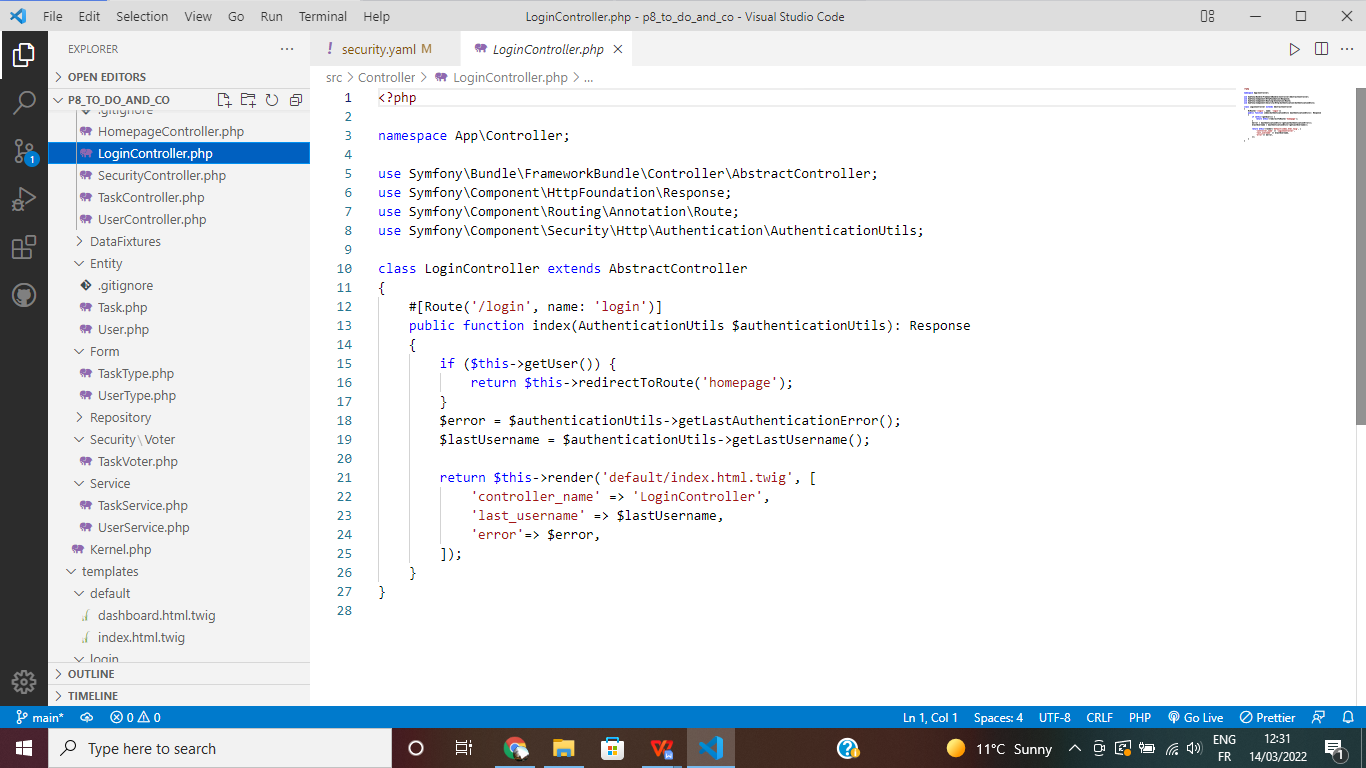
## LoginController.php : Gérer la formulaire de connexion

LoginController permet à retourne le formulaire de connexion et permet à l'authentificateur form\_login de gérer la soumission du formulaire automatiquement.

**Comment le créer?**

Grâce au Maker Bundle, il suffit de lancer les commandes :

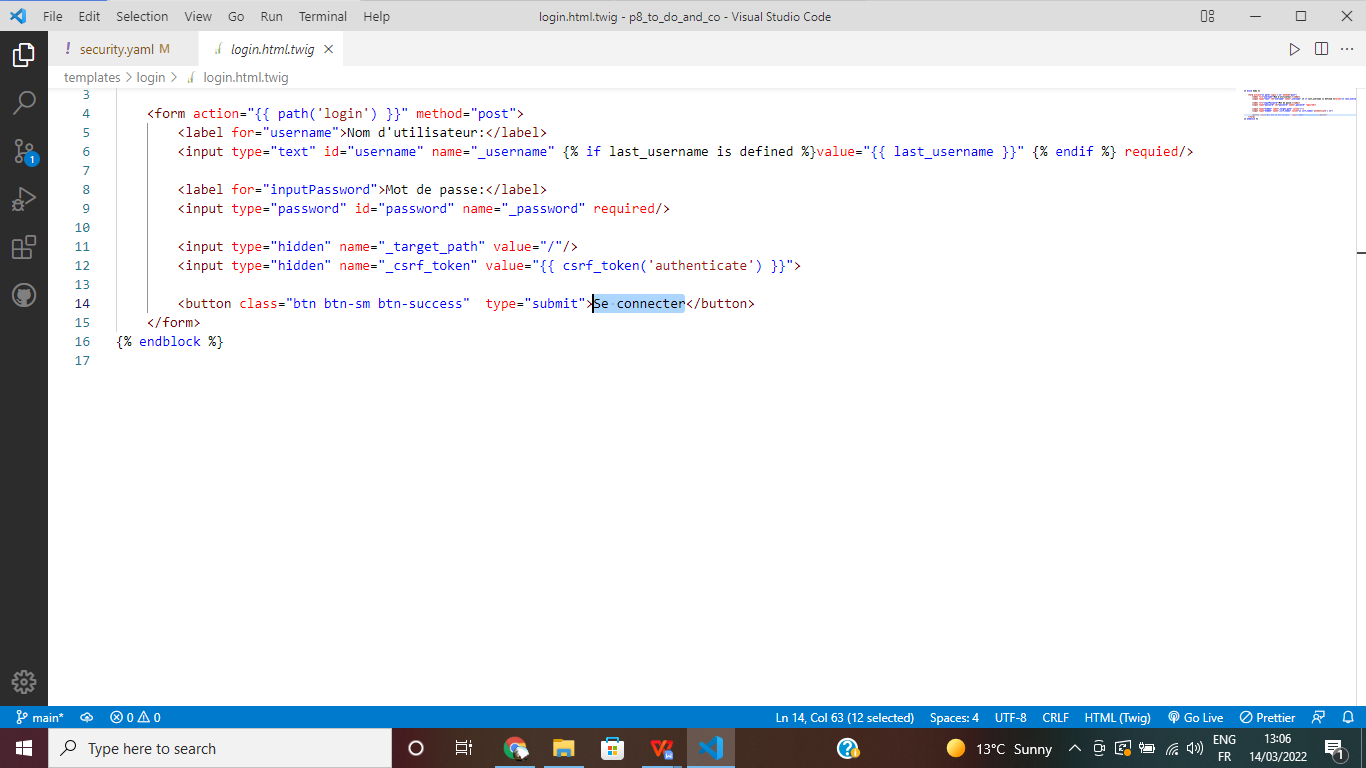
php bin/console make:controller Login



Avec le paramètre dans la route normée ‘login’, il stocke l'erreur lorsque le mot de passe/nom d'utilisateur rempli est incorrect. Dans notre cas, lorsqu'une telle erreur se produit, elle déclenche le message flash du template (login/login.html.twig qui est inclus dans default/index.html.twig). En outre, lorsqu'un utilisateur non authentifié tente d'accéder à un accès protégé, il sera redirigé vers cette route.

l'opération nous permet aussi de modifier le security.yaml (e.g. form\_login ) automatiquement.

## login.html.twig : le template du formulaire de connexion



<input type=”hidden” name=”\_target\_path” value =”/”>

il permet à l'application d'être redirigée vers l'URL "/" lorsque la connexion est réussie.

<input type=”hidden” name=”\_csrf\_token” value={{ csrf\_token (“authenticate”)}}”>

Il est ajouté afin d'empêcher les attaques CSRF de connexion.

# Comment l'application sait-elle si l'utilisateur est connecté et quel est son rôle ?

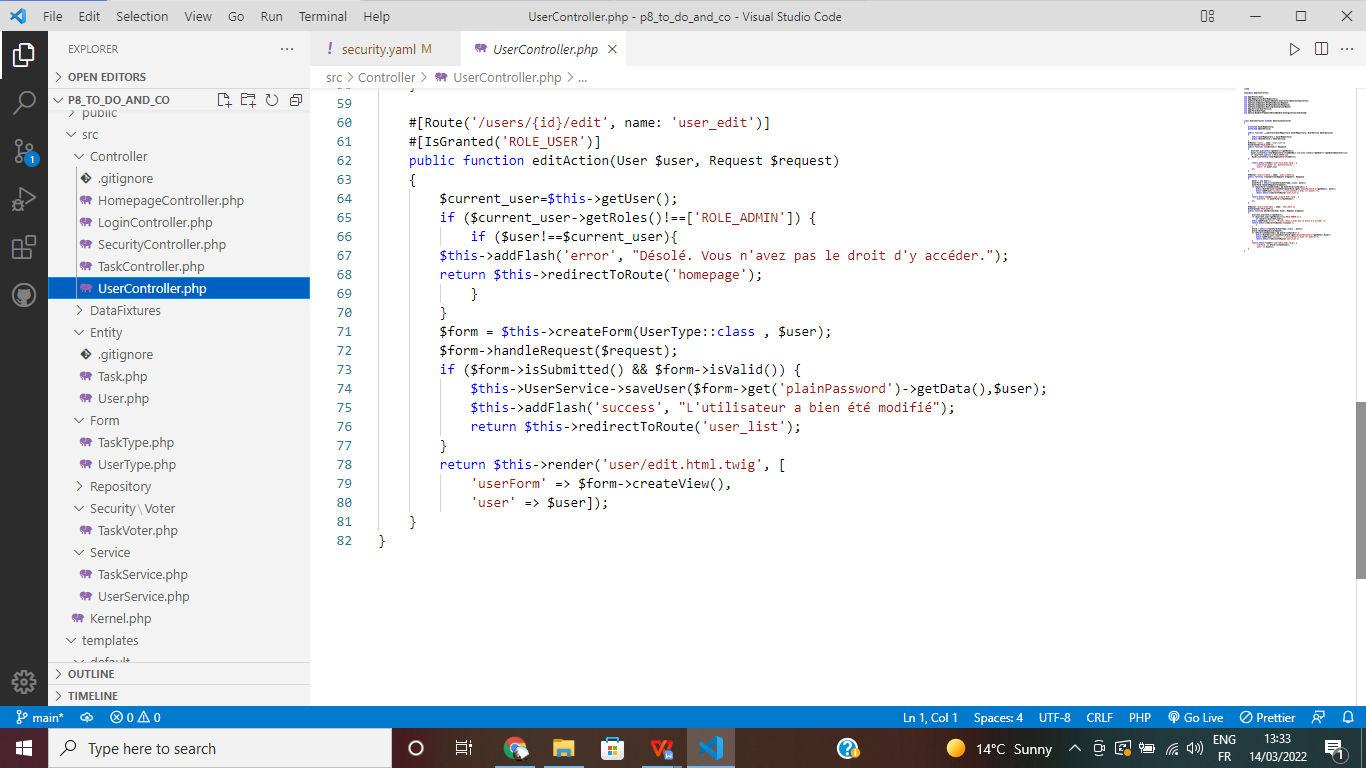
Controller : $this->getUser()

La méthode $this->getUser() ne peut être utilisée que par le contrôleur.

$this->getUser() permet à l'application de savoir si un utilisateur est connecté.

$this->getUser()->getRoles() permet à l'application de connaître le rôle de l'utilisateur connecté.

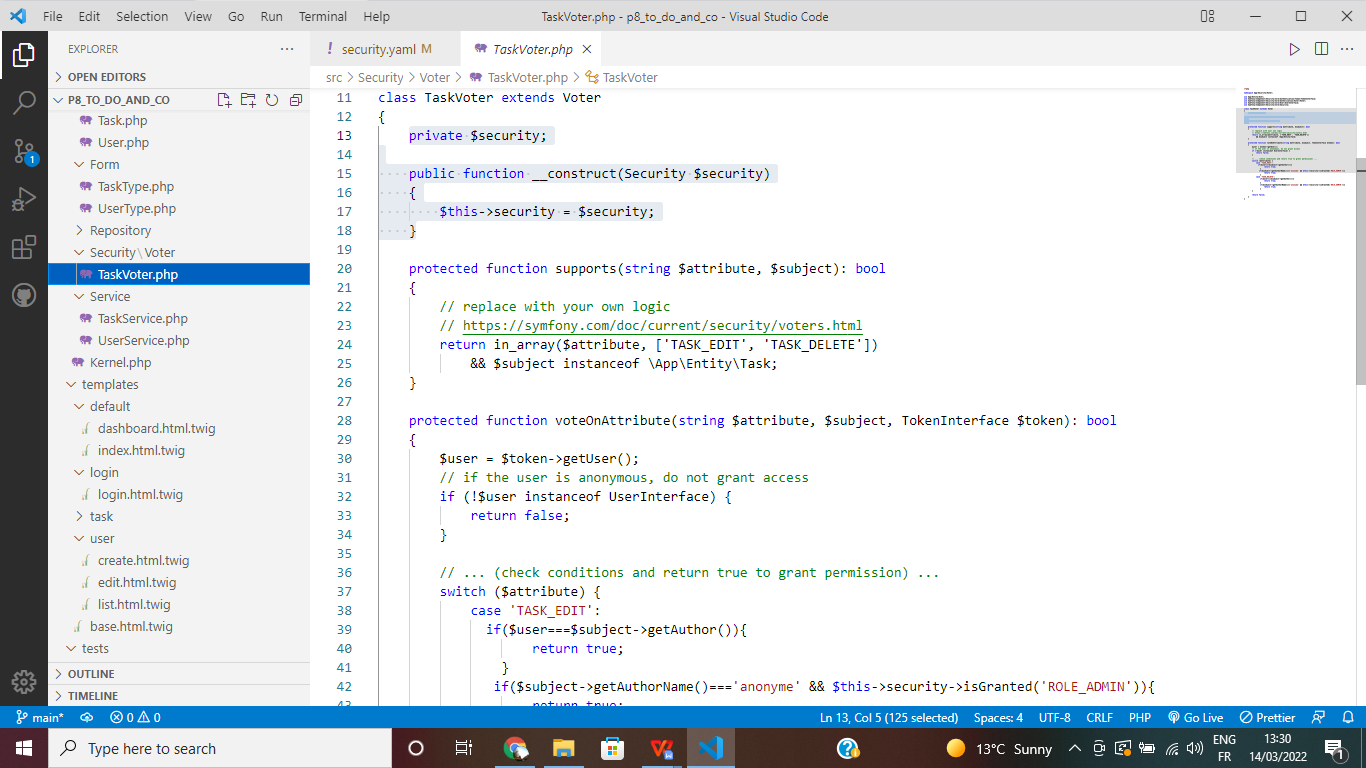
Voici un exemple tiré de notre application



Service : Security Service

Symfony\Component\Security\Core\Security peut être utilisé dans un service par injection de dépendances.

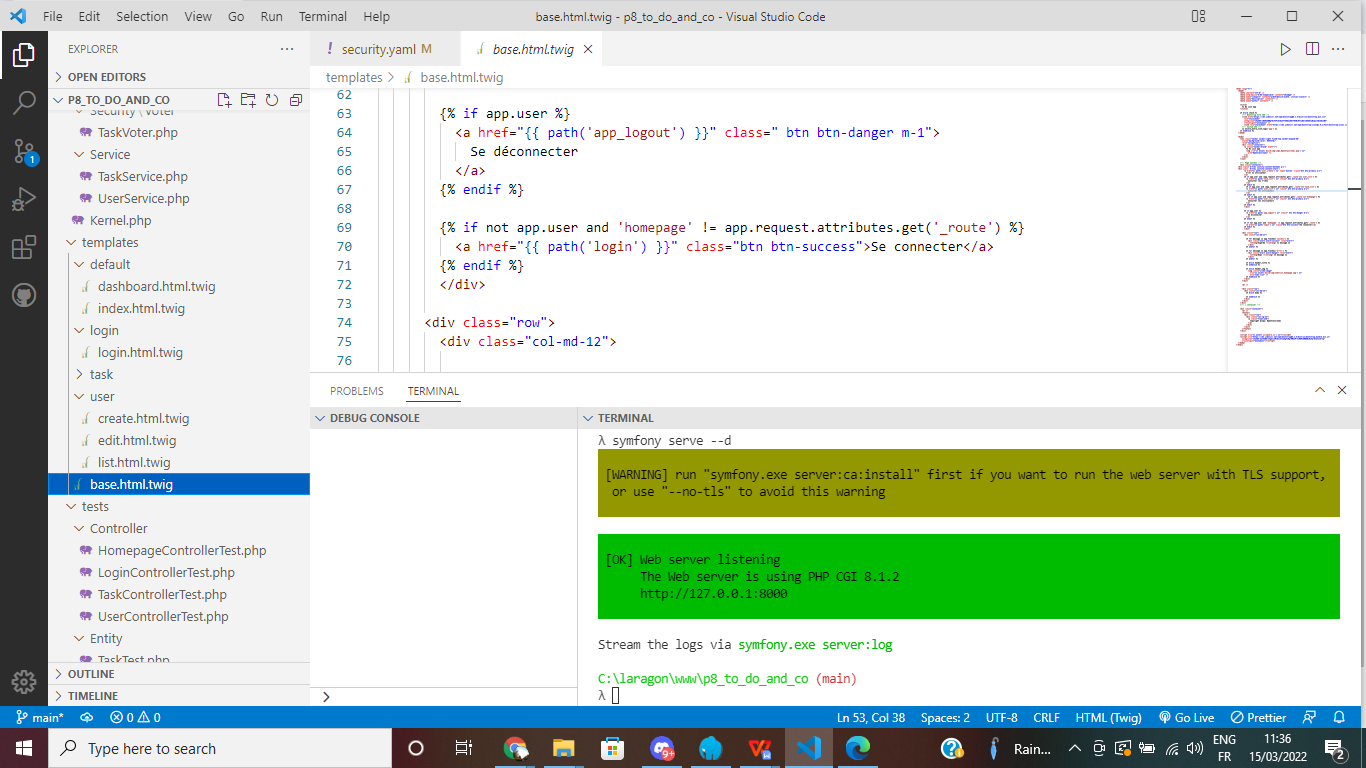
Voici un exemple tiré de notre application



Dans notre cas, $this->security->getUser() permet le service ‘Voter’ de savoir si un utilisateur est connecté. $this->security->isGranted(‘ROLE\_ADMIN’) est pour savoir si l’utilisateur a le role de ‘ROLE\_ADMIN’.

## Template:

le template est écrit dans le langage de template Twig. app.user est utilisé dans notre template pour que le template soit modifié selon que l'utilisateur est connecté ou non.



## Contrôle d'accès ? L'utilisateur avec son rôle spécifique est-il autorisé à faire une telle demande ?

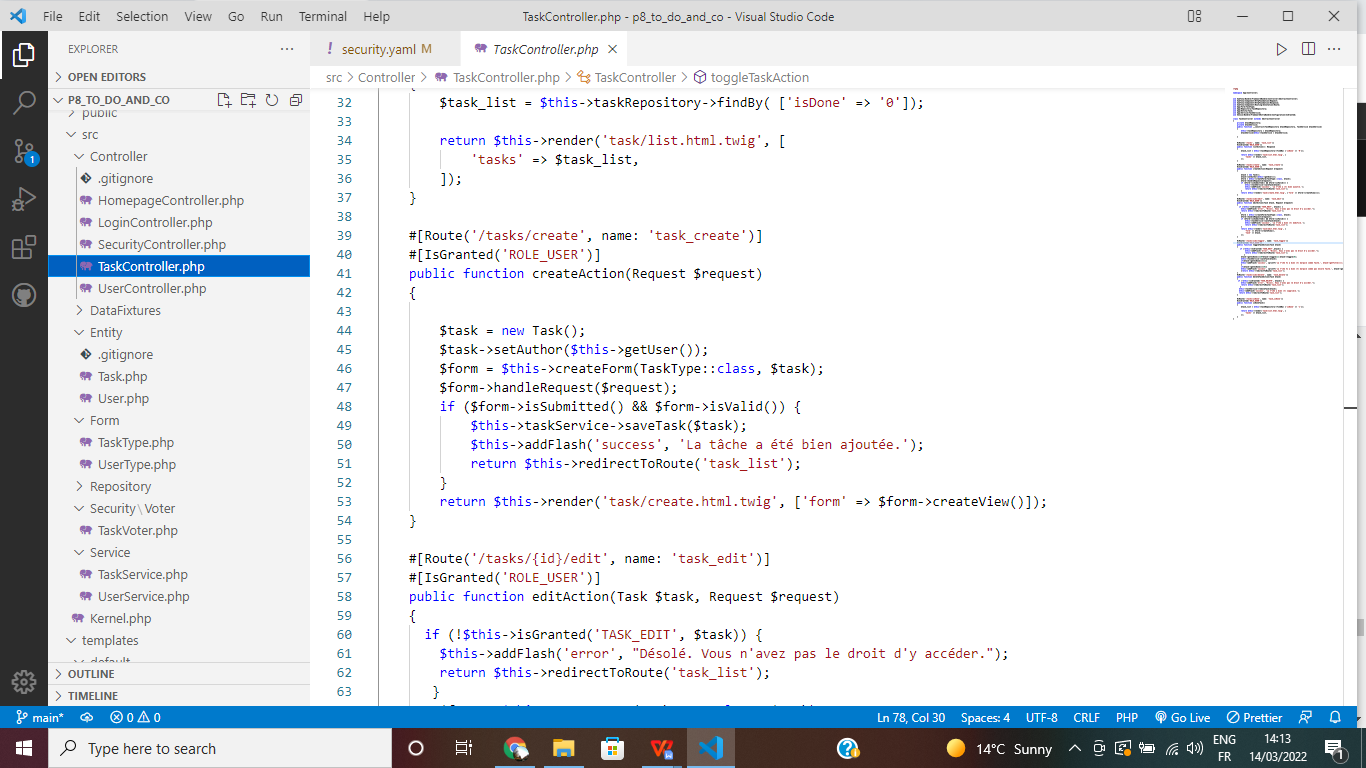
il y a plusieurs façons de configurer l'autorisation, dans notre application, ils sont situés :

1. Security.yaml (e.g. Role\_hierarchy, access\_control) ([voir ici](#_le fichier de configuration de la sécurité))

## Controller

Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted est utilisé pour que nous puissions utiliser une annotation pour limiter l'accès à certains rôles pour les routes.

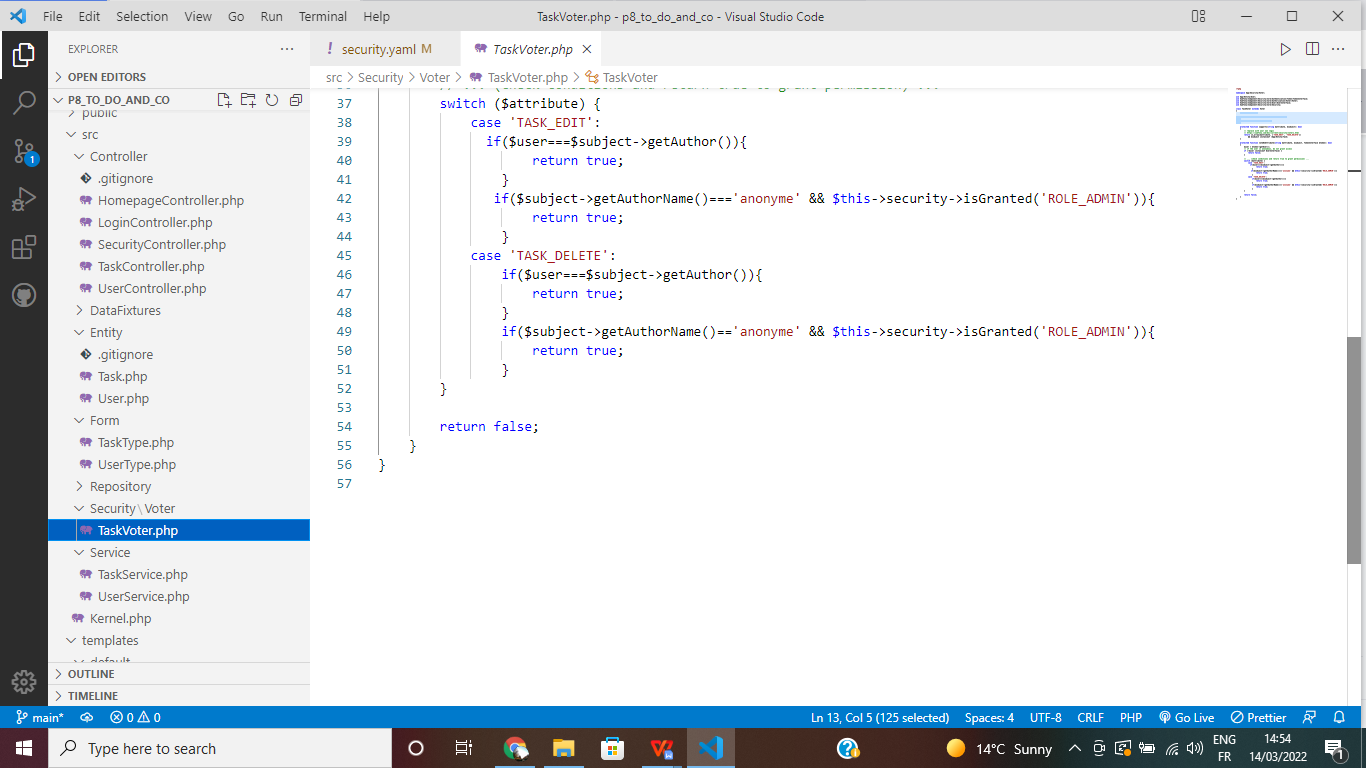
Voici un exemple tiré de notre application



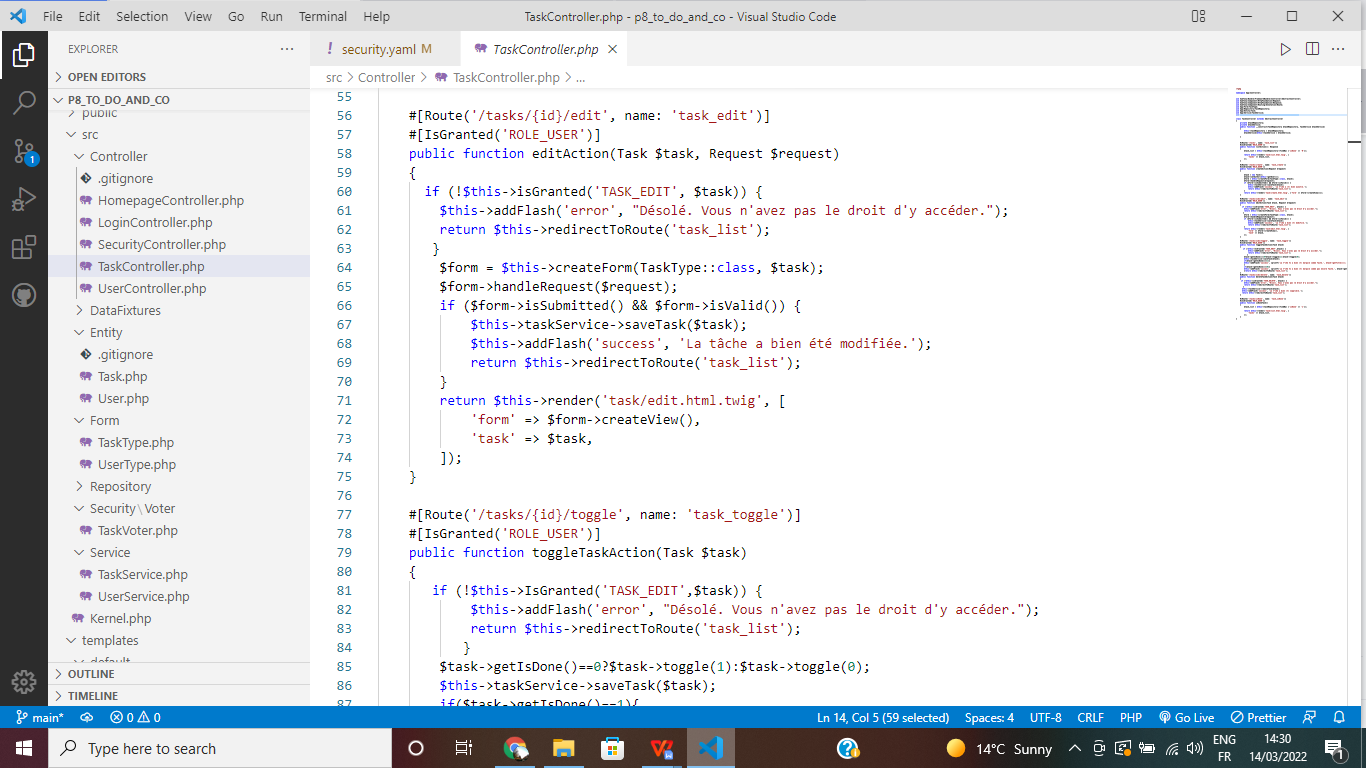
## Voter

Le système de voter est également mis en œuvre dans certaines routes pour centraliser la logique d'autorisation. Dans notre cas, elle est mise en œuvre lorsque le rôle de l'utilisateur n'est pas le seul critère.

Dans notre application, "TASK\_EDIT" est créé dans le TaskVoter, qui implémente l’interface VoterInterface .



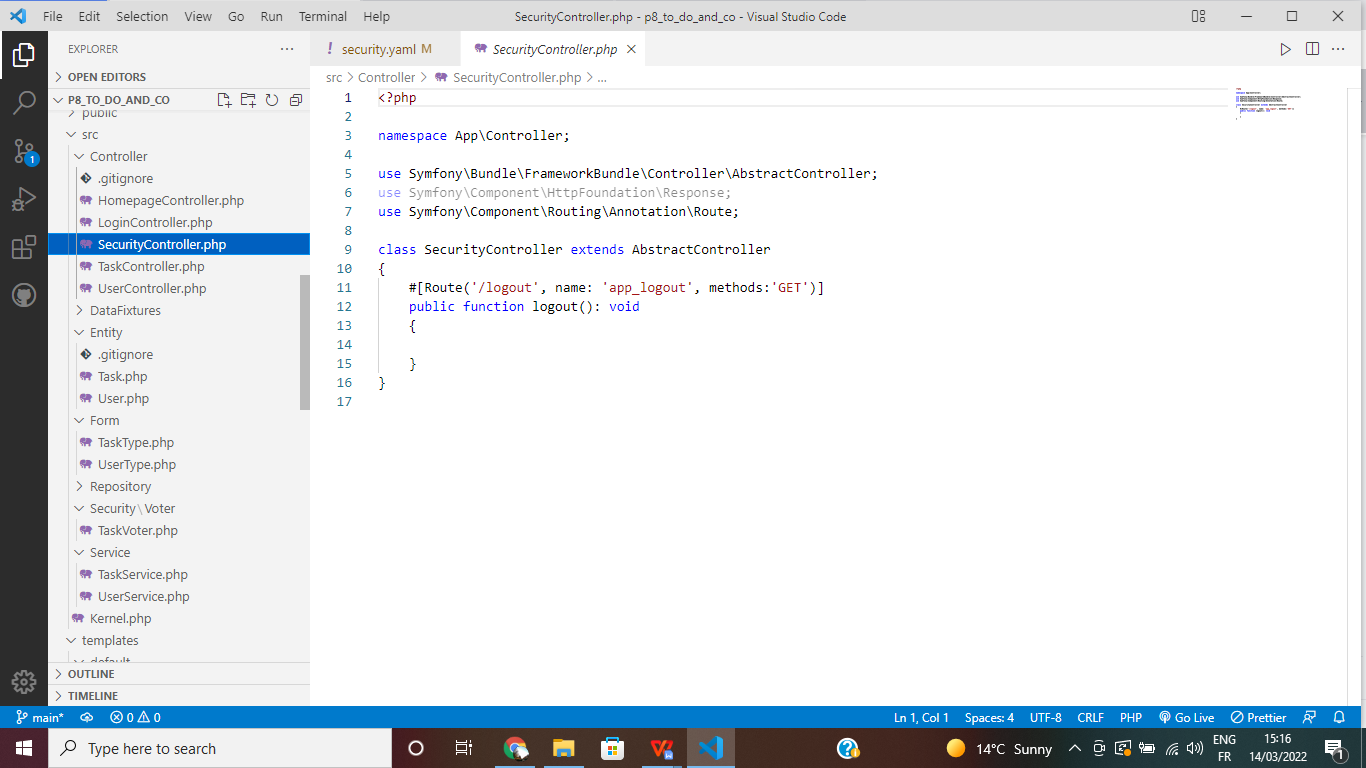
"Task\_edit" vérifiera si l'utilisateur est soit le propriétaire de la tâche, soit si l'utilisateur est ROLE\_ADMIN et que la tâche appartient à 'anonyme'.



l'exemple ci-dessus a utilisé ‘TASK\_EDIT’ pour la route nommé ‘task\_edit’, ce qui signifie que l'utilisateur devra répondre aux critères de TASK\_EDIT pour y’accéder.

# Logout

Afin de se déconnecter, il est nécessaire de l'avoir configuré dans le security.yaml ([voir ici](#_le fichier de configuration de la sécurité)) et de créer une route correspondant au nom qui est configuré sur security.yaml (‘app\_logout’ dans notre cas)



Le document explique principalement l'authentification de Symfony qui est liée à notre application. Pour en savoir plus:

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