[1 GUIDE D’INSTALLATION - SYMFONY 7 + ANGULAR 19 AVEC AUTHENTIFICATION JWT 3](#_Toc190010193)

[1.1 Objectif 3](#_Toc190010194)

[1.2 Prérequis 4](#_Toc190010195)

[1.3 CRÉATION DU PROJET 4](#_Toc190010196)

[1.3.1 Création de la structure du projet 4](#_Toc190010197)

[1.3.2 Créer un nouveau projet Symfony 4](#_Toc190010198)

[1.3.3 Installer les dépendances nécessaires à Symfony 4](#_Toc190010199)

[1.3.4 3. Installation du frontend Angular 4](#_Toc190010200)

[1.3.5 installer Angular Cli uniquement dans le projet 5](#_Toc190010201)

[1.3.6 Créer un nouveau projet Angular 5](#_Toc190010202)

[1.3.7 Vérifier la version d’Angular 5](#_Toc190010203)

[1.3.8 Si besoin, faire un upgrade de la version d’Angular 5](#_Toc190010204)

[1.4 CONFIGURATION DU BACKEND (SYMFONY) 5](#_Toc190010205)

[1.4.1 Configuration de la base de données 5](#_Toc190010206)

[1.4.2 Configuration CORS 6](#_Toc190010207)

[1.4.2.1 Exemple de cors 6](#_Toc190010208)

[1.4.3 Configuration JWT 6](#_Toc190010209)

[1.4.3.1 Exemple dans .env 7](#_Toc190010210)

[1.4.4 Créer/modifier config/packages/lexik\_jwt\_authentication.yaml : 7](#_Toc190010211)

[1.4.4.1 Exemple lexik\_jwt\_authentication. 7](#_Toc190010212)

[1.4.5 Générer les clés JWT : 8](#_Toc190010213)

[1.4.6 COMPLEMENT - :INTERACTIONS ENTRE LE FICHIER .ENV ET LEXIK\_JWT\_AUTHENTICATION.YAML 8](#_Toc190010214)

[1.4.6.1 Résumé 8](#_Toc190010215)

[1.4.6.2 Résumé - simplifié 8](#_Toc190010216)

[1.4.6.3 Résumé – plus détaillé 8](#_Toc190010217)

[1.4.6.4 Flux 9](#_Toc190010218)

[1.4.6.5 Schéma 9](#_Toc190010219)

[1.4.6.6 Détails 10](#_Toc190010220)

[1.4.6.6.1 Fichier .env 10](#_Toc190010221)

[1.4.6.6.2 Fichier lexik\_jwt\_authentication.yaml 10](#_Toc190010222)

[1.4.6.6.3 Interaction entre .env et lexik\_jwt\_authentication.yaml 11](#_Toc190010223)

[1.4.6.6.4 Résolution des chemins 11](#_Toc190010224)

[1.4.7 Configuration de la sécurité / fichier config/packages/security.yaml 11](#_Toc190010225)

[1.4.8 Création de l’entité User 12](#_Toc190010226)

[1.4.9 Adapter l’entity User à API Platform 13](#_Toc190010227)

[1.4.10 Vérifier que Doctrine a détecté les changements et faire la migration 16](#_Toc190010228)

[1.4.11 Création des fixtures 16](#_Toc190010229)

[1.4.12 Charger les fixtures dans la base de données 18](#_Toc190010230)

[1.4.13 Création du contrôleur d’authentification 18](#_Toc190010231)

[1.4.13.1 Exemple du SecurityController 18](#_Toc190010232)

[1.4.13.2 Explications du SecurityController 19](#_Toc190010233)

[1.5 RESOUDRE LE PROBLEME D’AFFICHAGE DE API PLATFORM DANS WINDOWS 22](#_Toc190010234)

[1.5.1 PROBLEME AFFICHAGE API 22](#_Toc190010235)

[1.6 fichier original asset\_mapper.yaml 22](#_Toc190010236)

[1.6.1 Modifier ass\_mapper.yaml comme suit 23](#_Toc190010237)

[1.6.2 Vérifier l’affichage avec symphony serve 24](#_Toc190010238)

[1.6.2.1 Affichage de l’accueil de Symfony à l’adresse http://127.0.0.1:8000/ 24](#_Toc190010239)

[1.6.2.2 Affichage de l’accueil de Api Plaform à l’adresse http://127.0.0.1:8000/api 25](#_Toc190010240)

[1.6.2.3 Etapes pour afficher les users sur Api Plaform à l’adresse http://127.0.0.1:8000/api#/User/api\_users\_get\_collection 25](#_Toc190010241)

[1.6.2.3.1 Sur la page http://127.0.0.1:8000/api cliquer sur GET /api/ users 25](#_Toc190010242)

[1.6.2.3.2 Cliquer sur « try it out 26](#_Toc190010243)

[1.6.2.3.3 Cliquer sur execute 26](#_Toc190010244)

[1.6.2.3.4 On obtient  http://127.0.0.1:8000/api#/User/api\_users\_get\_collection 27](#_Toc190010245)

[1.7 CONFIGURATION DU FRONTEND (ANGULAR) 28](#_Toc190010246)

[1.7.1 1. Installation des dépendances 28](#_Toc190010247)

[1.7.2 2. Configuration des modèles 28](#_Toc190010248)

[1.7.3 3. Configuration des services 28](#_Toc190010249)

[1.7.4 4. Création des composants 28](#_Toc190010250)

[1.7.5 5. Configuration du routing 28](#_Toc190010251)

[1.7.6 6. Configuration du composant principal 28](#_Toc190010252)

[1.7.7 7. Configuration de l’intercepteur HTTP 28](#_Toc190010253)

[1.8 Test de l’application 29](#_Toc190010254)

[1.8.1 1. Démarrer le backend 29](#_Toc190010255)

[1.8.2 2. Démarrer le frontend 29](#_Toc190010256)

[1.8.3 3. Tester l’application 29](#_Toc190010257)

[1.9 Dépannage courant 29](#_Toc190010258)

[2 INTERACTIONS ENTRE SYMFONY ET ANGULAR 29](#_Toc190010259)

[2.1.1 1. Flux d’authentification 29](#_Toc190010260)

[2.1.1.1 Étape 1 : Tentative de connexion 29](#_Toc190010261)

[2.1.2 2. Flux de récupération des utilisateurs 31](#_Toc190010262)

[2.1.2.1 Étape 1 : Chargement de la liste 31](#_Toc190010263)

[2.1.3 3. Points d’interaction clés 32](#_Toc190010264)

[2.1.3.1 API Platform (Symfony) 32](#_Toc190010265)

[2.1.3.2 Intercepteur HTTP (Angular) 32](#_Toc190010266)

[2.1.4 4. Gestion des erreurs 32](#_Toc190010267)

[2.1.4.1 Cas 1 : Échec d’authentification 32](#_Toc190010268)

[2.1.4.2 Cas 2 : Token expiré 32](#_Toc190010269)

[2.1.4.3 Cas 3 : Erreur CORS 32](#_Toc190010270)

[2.1.5 5. Sécurité 32](#_Toc190010271)

[2.1.5.1 Protection des routes 32](#_Toc190010272)

[2.1.6 6. Cycle de vie d’une requête 32](#_Toc190010273)

[2.1.7 7. État de l’application 33](#_Toc190010274)

[2.1.7.1 Côté Angular 33](#_Toc190010275)

[2.1.7.2 Côté Symfony 33](#_Toc190010276)

[2.1.8 8. Bonnes pratiques 33](#_Toc190010277)

# GUIDE D’INSTALLATION - SYMFONY 7 + ANGULAR 19 AVEC AUTHENTIFICATION JWT

## Objectif

Créer un projet associant Symfony 7 (dans un sous-dossier nommé “back”) et Angular 19 (dans un sous-dossier nommé “front”)

## Prérequis

* PHP 8.2 ou supérieur
* Composer
* Node.js 18.x ou supérieur
* npm
* Symfony CLI
* Git

## CRÉATION DU PROJET

### Création de la structure du projet

Créer le dossier principal

mkdir symang1

cd symang1

### Créer un nouveau projet Symfony

symfony new project\_name –webapp

Ici :

Symfony new back --webapp

### Installer les dépendances nécessaires à Symfony

composer require api

composer require symfony/security-bundle

composer require lexik/jwt-authentication-bundle

composer require doctrine/doctrine-fixtures-bundle –dev

composer require nelmio/cors-bundle

### 3. Installation du frontend Angular

mkdir front

se placer dans ce dossier

cd .. front

### installer Angular Cli uniquement dans le projet

Se placer dans le dossier racine du projet symang1

Exécuter :

npm install @angular/cli --save-dev

### Créer un nouveau projet Angular

npx ng new front --standalone --routing --style=css

### Vérifier la version d’Angular

npx ng version

ou dans package.json

{

  "devDependencies": {

    "@angular/cli": "^19.1.6"

  }

}

### Si besoin, faire un upgrade de la version d’Angular

<https://angular.dev/update-guide?v=17.0-18.0&l=1>

## CONFIGURATION DU BACKEND (SYMFONY)

### Configuration de la base de données

Copier le fichier .env et le nommer .env.local

Le modifier comme suit :

DATABASE\_URL="mysql://**root**:@127.0.0.1:3306/**symang**1?serverVersion=8.0.32&charset=utf8mb4"

Créer la base de données :

php bin/console doctrine:database:create

Vérifier la connexion à la base de données :

php bin/console doctrine:schema:validate

### Configuration CORS

Créer/modifier config/packages/nelmio\_cors.yaml :

nelmio\_cors:  
 defaults:  
 origin\_regex: true  
 allow\_origin: ['http://localhost:4200']  
 allow\_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']  
 allow\_headers: ['Content-Type', 'Authorization']  
 expose\_headers: ['Link']  
 max\_age: 3600

#### Exemple de cors

nelmio\_cors:

    defaults:

        origin\_regex: true

        allow\_origin: ['http://localhost:4200']

        allow\_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']

        allow\_headers: ['Content-Type', 'Authorization']

        expose\_headers: ['Link']

        max\_age: 3600

# Ancienne configuration commentée :

#nelmio\_cors:

#    defaults:

#        origin\_regex: true

#        allow\_origin: ['%env(CORS\_ALLOW\_ORIGIN)%']

#        allow\_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']

#        allow\_headers: ['Content-Type', 'Authorization']

#        expose\_headers: ['Link']

#        max\_age: 3600

#    paths:

#        '^/': null

### Configuration JWT

**Dans le fichier .env on doit avoir :**

###> lexik/jwt-authentication-bundle ###  
JWT\_SECRET\_KEY=%kernel.project\_dir%/config/jwt/private.pem  
JWT\_PUBLIC\_KEY=%kernel.project\_dir%/config/jwt/public.pem  
JWT\_PASSPHRASE=votre\_phrase\_secrete\_complexe  
###< lexik/jwt-authentication-bundle ###

Explications :

* **JWT\_SECRET\_KEY** : Chemin vers la clé privée (private.pem) utilisée pour signer les tokens JWT.
* **JWT\_PUBLIC\_KEY** : Chemin vers la clé publique (public.pem) utilisée pour vérifier la signature des tokens JWT.
* **JWT\_PASSPHRASE** : Phrase secrète utilisée pour protéger la clé privée.

Les balises ###> ... ### et ###< ... ### sont des délimitations automatiques ajoutées par Symfony Flex pour identifier clairement les configurations liées à un bundle spécifique.

#### Exemple dans .env

###> lexik/jwt-authentication-bundle ###

JWT\_SECRET\_KEY=%kernel.project\_dir%/config/jwt/private.pem

JWT\_PUBLIC\_KEY=%kernel.project\_dir%/config/jwt/public.pem

JWT\_PASSPHRASE=af95901d997b321129b21eb4548b9e1a4dea1a18c5723d6d3ea0ad39a79bbb5d

###< lexik/jwt-authentication-bundle ###

### Créer/modifier config/packages/lexik\_jwt\_authentication.yaml :

lexik\_jwt\_authentication:  
 secret\_key: '%env(resolve:JWT\_SECRET\_KEY)%'  
 public\_key: '%env(resolve:JWT\_PUBLIC\_KEY)%'  
 pass\_phrase: '%env(JWT\_PASSPHRASE)%'  
 token\_ttl: 3600

#### Exemple lexik\_jwt\_authentication.

lexik\_jwt\_authentication:

    secret\_key: '%env(resolve:JWT\_SECRET\_KEY)%'

    public\_key: '%env(resolve:JWT\_PUBLIC\_KEY)%'

    pass\_phrase: '%env(JWT\_PASSPHRASE)%'

    token\_ttl: 3600

### Générer les clés JWT :

php bin/console lexik:jwt:generate-keypair

Cette commande va créer un sous-dossier jwt dans config/ avec les fichiers private.pem et public.pem,

prêts à être utilisés pour la gestion des tokens JWT.

* private.pem : La clé privée utilisée pour signer les tokens JWT.
* public.pem : La clé publique utilisée pour vérifier les tokens JWT.

### COMPLEMENT - :INTERACTIONS ENTRE LE FICHIER .ENV ET LEXIK\_JWT\_AUTHENTICATION.YAML

#### Résumé

* Le fichier .env contient les valeurs brutes des variables d'environnement.
* Le fichier lexik\_jwt\_authentication.yaml utilise ces variables pour configurer le bundle JWT.
* Symfony résout les variables d'environnement au moment de charger la configuration, en remplaçant les placeholders comme %env(JWT\_SECRET\_KEY)% par les valeurs réelles.

#### Résumé - simplifié

* Symfony lit les variables d'environnement brutes dans .env.
* Ces variables sont utilisées dans lexik\_jwt\_authentication.yaml via des placeholders comme

%env(resolve:JWT\_SECRET\_KEY)%.

* Symfony résout ces placeholders en remplaçant les variables par leurs valeurs brutes, qui

pointent vers les fichiers private.pem et public.pem contenant les clés cryptographiques.

#### Résumé – plus détaillé

* Symfony lit les valeurs brutes des variables d'environnement à partir du fichier .env.
* Ces valeurs sont ensuite référencées dans le fichier lexik\_jwt\_authentication.yaml à l'aide de placeholders, par exemple : %env(resolve:JWT\_SECRET\_KEY)%.
* Symfony résout ces placeholders en remplaçant les variables par leurs valeurs effectives.

Par exemple, %kernel.project\_dir%/config/jwt/private.pem sera converti en un chemin absolu.

* Les clés cryptographiques (privée et publique) sont stockées dans les fichiers private.pem et

public.pem, dont les chemins sont spécifiés dans le fichier .env.

#### Flux

1. **Point de départ : Fichier .env**

→ Contient les variables d'environnement brutes

→ Définit JWT\_SECRET\_KEY et JWT\_PUBLIC\_KEY

1. **Fichier de configuration : lexik\_jwt\_authentication.yaml**

→ Utilise les variables via des placeholders

→ Format : %env(resolve:JWT\_SECRET\_KEY)%

1. **Processus de résolution Symfony**

→ Lit les variables du .env

→ Convertit les chemins relatifs en chemins absolus

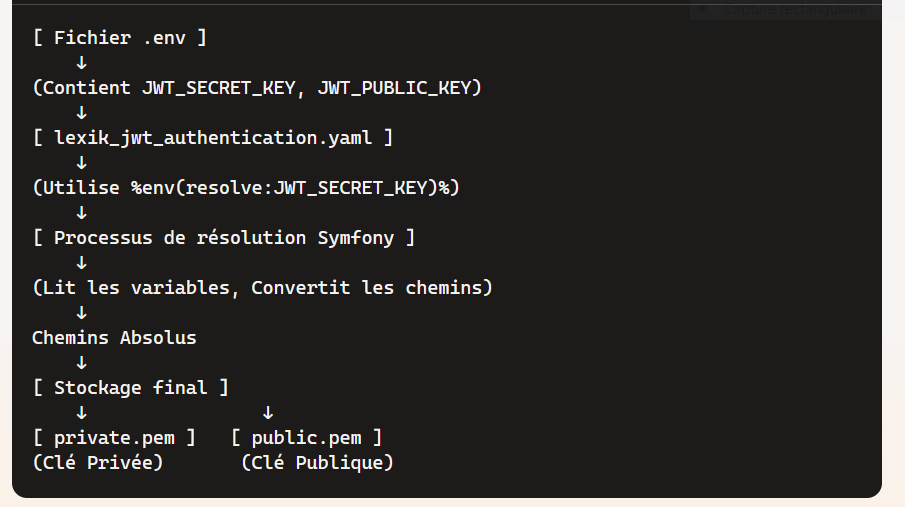
→ Exemple : %kernel.project\_dir%/config/jwt/private.pem devient le chemin complet

1. **Stockage final**

→ private.pem : stocke la clé privée

→ public.pem : stocke la clé publique

#### Schéma



#### Détails

##### Fichier .env

Le fichier .env est utilisé pour stocker des variables d'environnement spécifiques à votre application.

Ces variables sont souvent utilisées pour configurer des paramètres sensibles ou spécifiques à l'environnement (comme les clés JWT, les mots de passe, les URLs de base de données, etc.).

###> lexik/jwt-authentication-bundle ###

JWT\_SECRET\_KEY=%kernel.project\_dir%/config/jwt/private.pem

JWT\_PUBLIC\_KEY=%kernel.project\_dir%/config/jwt/public.pem

JWT\_PASSPHRASE=af95901d997b321129b21eb4548b9e1a4dea1a18c5723d6d3ea0ad39a79bbb5d

###< lexik/jwt-authentication-bundle ###

* JWT\_SECRET\_KEY : Chemin vers la clé privée utilisée pour signer les tokens JWT.
* JWT\_PUBLIC\_KEY : Chemin vers la clé publique utilisée pour vérifier les tokens JWT.
* JWT\_PASSPHRASE : Une phrase secrète utilisée pour chiffrer/déchiffrer les clés.

##### Fichier lexik\_jwt\_authentication.yaml

Ce fichier est un fichier de configuration spécifique au bundle lexik/jwt-authentication-bundle.

Il permet de configurer le comportement du bundle, comme les chemins des clés, la phrase secrète, et la durée de vie des tokens.

lexik\_jwt\_authentication:

    secret\_key: '%env(resolve:JWT\_SECRET\_KEY)%'

    public\_key: '%env(resolve:JWT\_PUBLIC\_KEY)%'

    pass\_phrase: '%env(JWT\_PASSPHRASE)%'

    token\_ttl: 3600

* secret\_key : Cette ligne utilise la variable d'environnement JWT\_SECRET\_KEY définie dans le fichier .env. La fonction %env(resolve:JWT\_SECRET\_KEY)% indique à Symfony de résoudre la valeur de cette variable d'environnement.
* public\_key : De même, cette ligne utilise la variable d'environnement JWT\_PUBLIC\_KEY.
* pass\_phrase : Cette ligne utilise la variable d'environnement JWT\_PASSPHRASE.
* token\_ttl : Cette option définit la durée de vie des tokens JWT en secondes (ici, 3600 secondes, soit 1 heure).

##### Interaction entre .env et lexik\_jwt\_authentication.yaml

Lorsque Symfony charge la configuration, il va d'abord lire les variables d'environnement définies dans le fichier .env.

Ensuite, dans le fichier lexik\_jwt\_authentication.yaml, les valeurs des clés secret\_key, public\_key, et pass\_phrase sont résolues en utilisant les variables d'environnement définies dans .env.

 %env(resolve:JWT\_SECRET\_KEY)% sera remplacé par la valeur de JWT\_SECRET\_KEY définie dans .env, c'est-à-dire %kernel.project\_dir%/config/jwt/private.pem.

 %env(JWT\_PASSPHRASE)% sera remplacé par la valeur de JWT\_PASSPHRASE définie dans .env, c'est-à-dire af95901d997b321129b21eb4548b9e1a4dea1a18c5723d6d3ea0ad39a79bbb5d

##### Résolution des chemins

La notation %kernel.project\_dir% est une notation spécifique à Symfony qui fait référence au répertoire racine du projet.

Ainsi, %kernel.project\_dir%/config/jwt/private.pem sera résolu en un chemin absolu vers le fichier

private.pem dans le répertoire config/jwt du projet.

### Configuration de la sécurité / fichier config/packages/security.yaml

Modifier config/packages/security.yaml :

security:

    password\_hashers:

        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'

    providers:

        app\_user\_provider:

            entity:

                class: App\Entity\User

                property: email

    firewalls:

        dev:

            pattern: ^/(\_(profiler|wdt)|css|images|js)/

            security: false

        login:

            pattern: ^/api/login

            stateless: true

            json\_login:

                check\_path: /api/login

                success\_handler: lexik\_jwt\_authentication.handler.authentication\_success

                failure\_handler: lexik\_jwt\_authentication.handler.authentication\_failure

        api:

            pattern: ^/api

            stateless: true

            jwt: ~

        main:

            lazy: true

            provider: app\_user\_provider

    access\_control:

        - { path: ^/api/login, roles: PUBLIC\_ACCESS }

        # - { path: ^/api, roles: IS\_AUTHENTICATED\_FULLY }

when@test:

    security:

        password\_hashers:

            Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:

                algorithm: auto

                cost: 4

                time\_cost: 3

                memory\_cost: 1024

### Création de l’entité User

php bin/console make:entity

ici

php bin/console make:user

### Adapter l’entity User à API Platform

#[ApiResource]

#[ORM\Table(name: 'user')]

Import de ApiResource

**Ajouter dans l’entity :**

use  ApiPlatform\Metadata\ApiResource;

Pour exposer l'entité comme ressource API.

**Ajouter dans l’entity :**

#[ApiResource]

* #[ApiResource] est une annotation d'API Platform.
* Elle permet d'exposer automatiquement l'entité User comme une ressource API RESTful.
* Cela génère des endpoints (comme GET /users, POST /users, etc.) sans que l’on ait besoin d'écrire de contrôleur.

**Ajouter dans l’entity :**

#[ORM\Table(name: '`user`')]

* Cela définit explicitement le nom de la table dans la base de données.
* L'utilisation des backticks `user` est utile si le nom "user" est un mot réservé dans certains SGBD (comme MySQL).
* Sans cette annotation, Doctrine utilise par défaut le nom de la classe en minuscules (user).
* L'ajout de #[ORM\Table(name: 'user')] est une bonne pratique pour éviter des problèmes avec des SGBD sensibles aux mots réservés.

**Entity User complète :**

<?php

namespace App\Entity;

use  ApiPlatform\Metadata\ApiResource;

use App\Repository\UserRepository;

use Doctrine\ORM\Mapping as ORM;

use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;

use Symfony\Component\Security\Core\User\UserInterface;

#[ORM\Entity(repositoryClass: UserRepository::class)]

#[ApiResource]

#[ORM\Table(name: '`user`')]

#[ORM\UniqueConstraint(name: 'UNIQ\_IDENTIFIER\_EMAIL', fields: ['email'])]

class User implements UserInterface, PasswordAuthenticatedUserInterface

{

    #[ORM\Id]

    #[ORM\GeneratedValue]

    #[ORM\Column]

    private ?int $id = null;

    #[ORM\Column(length: 180)]

    private ?string $email = null;

    /\*\*

     \* @var list<string> The user roles

     \*/

    #[ORM\Column]

    private array $roles = [];

    /\*\*

     \* @var string The hashed password

     \*/

    #[ORM\Column]

    private ?string $password = null;

    public function getId(): ?int

    {

        return $this->id;

    }

    public function getEmail(): ?string

    {

        return $this->email;

    }

    public function setEmail(string $email): static

    {

        $this->email = $email;

        return $this;

    }

    /\*\*

     \* A visual identifier that represents this user.

     \*

     \* @see UserInterface

     \*/

    public function getUserIdentifier(): string

    {

        return (string) $this->email;

    }

    /\*\*

     \* @see UserInterface

     \*

     \* @return list<string>

     \*/

    public function getRoles(): array

    {

        $roles = $this->roles;

        // guarantee every user at least has ROLE\_USER

        $roles[] = 'ROLE\_USER';

        return array\_unique($roles);

    }

    /\*\*

     \* @param list<string> $roles

     \*/

    public function setRoles(array $roles): static

    {

        $this->roles = $roles;

        return $this;

    }

    /\*\*

     \* @see PasswordAuthenticatedUserInterface

     \*/

    public function getPassword(): ?string

    {

        return $this->password;

    }

    public function setPassword(string $password): static

    {

        $this->password = $password;

        return $this;

    }

    /\*\*

     \* @see UserInterface

     \*/

    public function eraseCredentials(): void

    {

        // If you store any temporary, sensitive data on the user, clear it here

        // $this->plainPassword = null;

    }

}

### Vérifier que Doctrine a détecté les changements et faire la migration

php bin/console doctrine:schema:validate

php bin/console make:migration

php bin/console doctrine:migrations:migrate

vérifier la cohérence :

php bin/console doctrine:schema:validate

### Création des fixtures

php bin/console make:fixture

Modifier le fichier UserFixtures.php comme suit :

<?php

namespace App\DataFixtures;

use Doctrine\Bundle\FixturesBundle\Fixture;

use Doctrine\Persistence\ObjectManager;

use App\Entity\User;

use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;

class UserFixtures extends Fixture

{

    private UserPasswordHasherInterface $passwordHasher;

    public function \_\_construct(UserPasswordHasherInterface $passwordHasher)

    {

        $this->passwordHasher = $passwordHasher;

    }

    public function load(ObjectManager $manager): void

    {

        // Liste des utilisateurs à créer

        $users = [

            ['email' => 'user1@mail.com', 'password' => 'user', 'roles' => ['ROLE\_USER']],

            ['email' => 'user2@mail.com', 'password' => 'user', 'roles' => ['ROLE\_USER']],

            ['email' => 'admin@mail.com', 'password' => 'admin', 'roles' => ['ROLE\_ADMIN']],

            ['email' => 'user3@mail.com', 'password' => 'user', 'roles' => ['ROLE\_USER']],

        ];

        foreach ($users as $userData) {

            $user = new User();

            $user->setEmail($userData['email']);

            // Hachage du mot de passe avant enregistrement

            $hashedPassword = $this->passwordHasher->hashPassword($user, $userData['password']);

            $user->setPassword($hashedPassword);

            $user->setRoles($userData['roles']);

            $manager->persist($user);

        }

        $manager->flush();

    }

}

### Charger les fixtures dans la base de données

php bin/console doctrine:fixtures:load

S’asurer que tout est en ordre :

php bin/console doctrine:schema:validate

### Création du contrôleur d’authentification

Exécuter

php bin/console make:controller SecurityController

Ce controller gère l'authentification des utilisateurs via la route /api/login.

Lorsque l'utilisateur est authentifié, il reçoit son identifiant et ses rôles en réponse.

Sinon, un message d'erreur est renvoyé pour indiquer que les identifiants sont invalide

#### Exemple du SecurityController

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\JsonResponse;

use Symfony\Component\Routing\Annotation\Route;

use Symfony\Component\Security\Http\Attribute\CurrentUser;

use App\Entity\User;

class SecurityController extends AbstractController

{

    #[Route('/api/login', name: 'api\_login', methods: ['POST'])]

    public function login(#[CurrentUser] ?User $user): JsonResponse

    {

        if (null === $user) {

            return $this->json([

                'message' => 'Identifiants invalides'

            ], 401);

        }

        return $this->json([

            'user' => $user->getUserIdentifier(),

            'roles' => $user->getRoles()

        ]);

    }

}

#### Explications du SecurityController

**namespace App\Controller;**

Cela définit le namespace pour la classe SecurityController. En Symfony, il est important de définir un namespace pour organiser les fichiers de manière cohérente. Ici, la classe appartient au namespace App\Controller, ce qui signifie qu'elle fait partie du répertoire src/Controller de l'application.

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

Cela importe la classe AbstractController de Symfony. Cette classe est la classe de base pour tous les contrôleurs Symfony, fournissant des méthodes utiles pour gérer les requêtes HTTP, rendre des vues, renvoyer des réponses JSON, etc.

**use Symfony\Component\HttpFoundation\JsonResponse;**

Cela importe la classe JsonResponse, qui est utilisée pour renvoyer des réponses HTTP au format JSON. Dans ce contrôleur, on utilise cette classe pour envoyer des réponses JSON avec des informations sur l'utilisateur authentifié ou un message d'erreur.

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

Cela importe l'annotation Route, qui est utilisée pour définir des routes dans Symfony. Elle permet de lier une URL spécifique à une méthode de contrôleur. Ici, cette annotation est utilisée pour définir une route /api/login.

**use Symfony\Component\Security\Http\Attribute\CurrentUser;**

Cela importe l'attribut CurrentUser, qui est utilisé pour injecter l'utilisateur actuel dans la méthode du contrôleur. Cet attribut permet de récupérer facilement l'utilisateur connecté dans la méthode login.

**use App\Entity\User;**

Cela importe la classe User, qui représente l'entité de l'utilisateur dans la base de données. Cette entité est généralement utilisée pour interagir avec les données des utilisateurs, comme leurs identifiants, rôles, etc.

**class SecurityController extends AbstractController**

Cela définit la classe SecurityController, qui hérite de la classe AbstractController de Symfony. La classe SecurityController contient la logique pour gérer les actions liées à l'authentification des utilisateurs.

**#[Route('/api/login', name: 'api\_login', methods: ['POST'])]**

C'est une annotation qui définit la route /api/login. Elle spécifie que cette route correspond à une méthode POST et que le nom de la route est api\_login. Cette route est utilisée pour l'authentification de l'utilisateur via une requête POST envoyée à l'URL /api/login.

**public function login(#[CurrentUser] ?User $user): JsonResponse**

Cette méthode définit l'action associée à la route /api/login. Elle prend comme paramètre un utilisateur actuel, qui est injecté automatiquement grâce à l'attribut #[CurrentUser].

Le ?User indique que l'utilisateur peut être nul (lorsque l'utilisateur n'est pas authentifié). La méthode retourne une instance de JsonResponse (une réponse HTTP au format JSON).

**if (null === $user) {**

Cette condition vérifie si la variable $user est null, ce qui signifie que l'utilisateur n'est pas authentifié. Cela se produit si la demande est effectuée sans un jeton d'authentification valide.

**return $this->json([**

Si l'utilisateur n'est pas authentifié, on renvoie une réponse JSON avec un message indiquant que les identifiants sont invalides. La méthode json() est utilisée pour créer une réponse HTTP avec des données JSON.

**'message' => 'Identifiants invalides'**

Ce tableau définit la réponse JSON avec un message d'erreur. Ce message sera renvoyé au client pour l'informer que les informations d'identification sont incorrectes.

**], 401);**

Cela termine la réponse JSON et envoie un code HTTP 401 (Unauthorized), qui signifie que l'utilisateur n'est pas autorisé à accéder à cette ressource.

**} Fermeture de la condition if.**

**return $this->json([**

Si l'utilisateur est authentifié (si $user n'est pas nul), on renvoie une réponse JSON contenant des informations sur l'utilisateur. La méthode json() est à nouveau utilisée pour renvoyer des données au format JSON.

**'user' => $user->getUserIdentifier(),**

Cela inclut l'identifiant de l'utilisateur dans la réponse JSON.

La méthode getUserIdentifier() renvoie l'identifiant de l'utilisateur, comme son email ou son nom d'utilisateur.

**'roles' => $user->getRoles()**

Cela inclut les rôles de l'utilisateur dans la réponse JSON.

La méthode getRoles() renvoie un tableau des rôles de l'utilisateur, par exemple ['ROLE\_USER'].

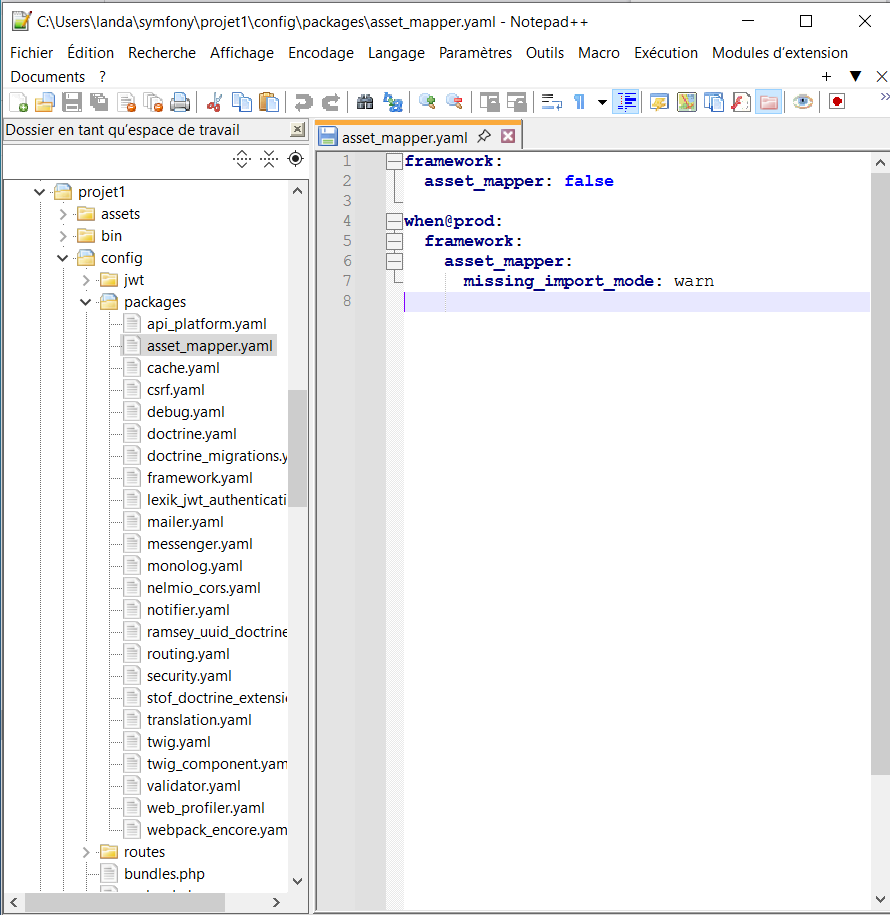
**]); Fermeture du tableau contenant les données à renvoyer dans la réponse JSON.**

**} Fermeture de la méthode login().**

## RESOUDRE LE PROBLEME D’AFFICHAGE DE API PLATFORM DANS WINDOWS

Si API Platform ne s’affiche pas correctement,

### Aller dans config/packages/asset\_mapper.yaml



### fichier original asset\_mapper.yaml

framework:

    asset\_mapper:

        # The paths to make available to the asset mapper.

        paths:

            - assets/

        missing\_import\_mode: strict

when@prod:

    framework:

        asset\_mapper:

            missing\_import\_mode: warn

### Modifier ass\_mapper.yaml comme suit

framework:

    asset\_mapper: false

when@prod:

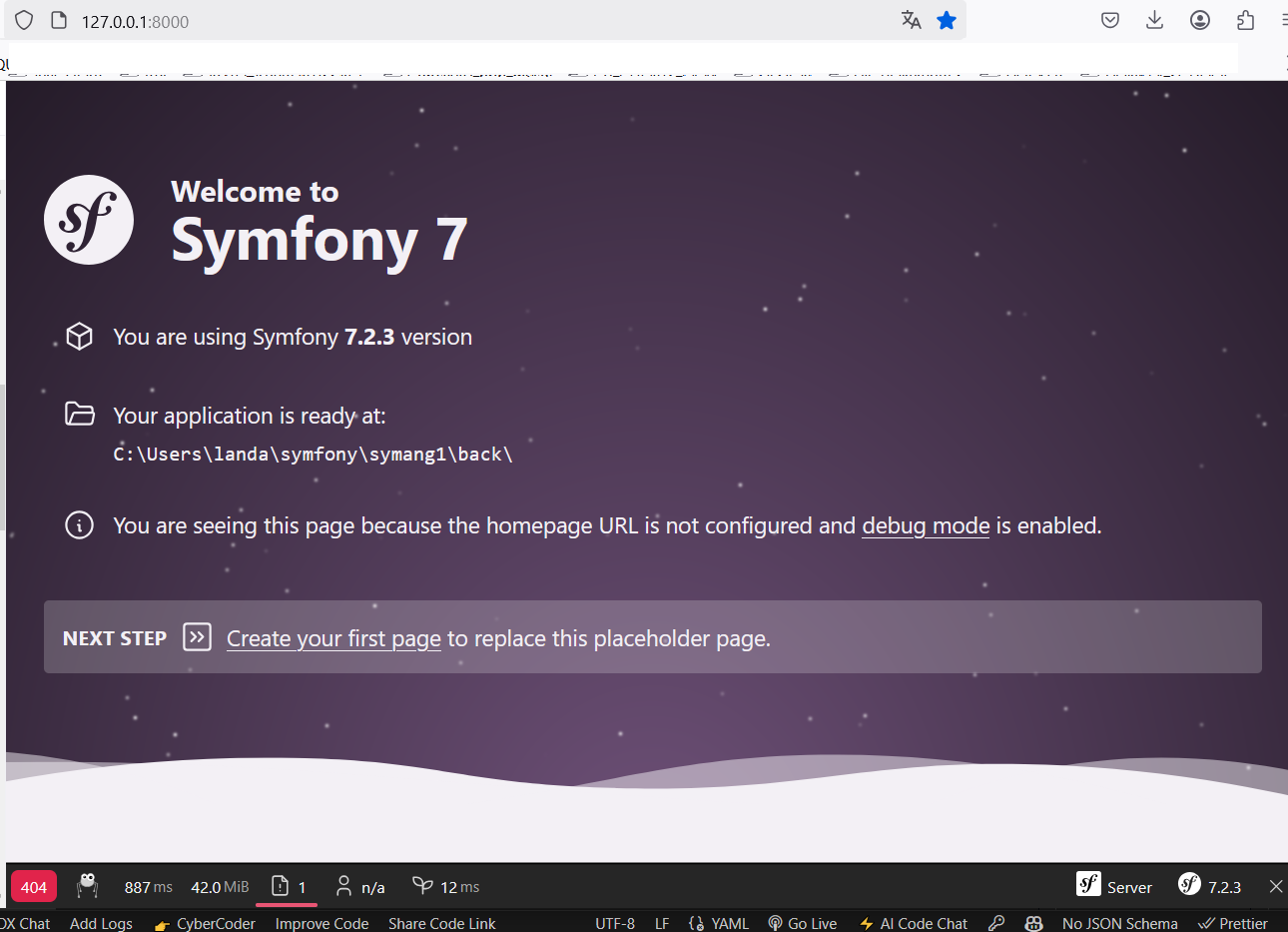
    framework:

        asset\_mapper:

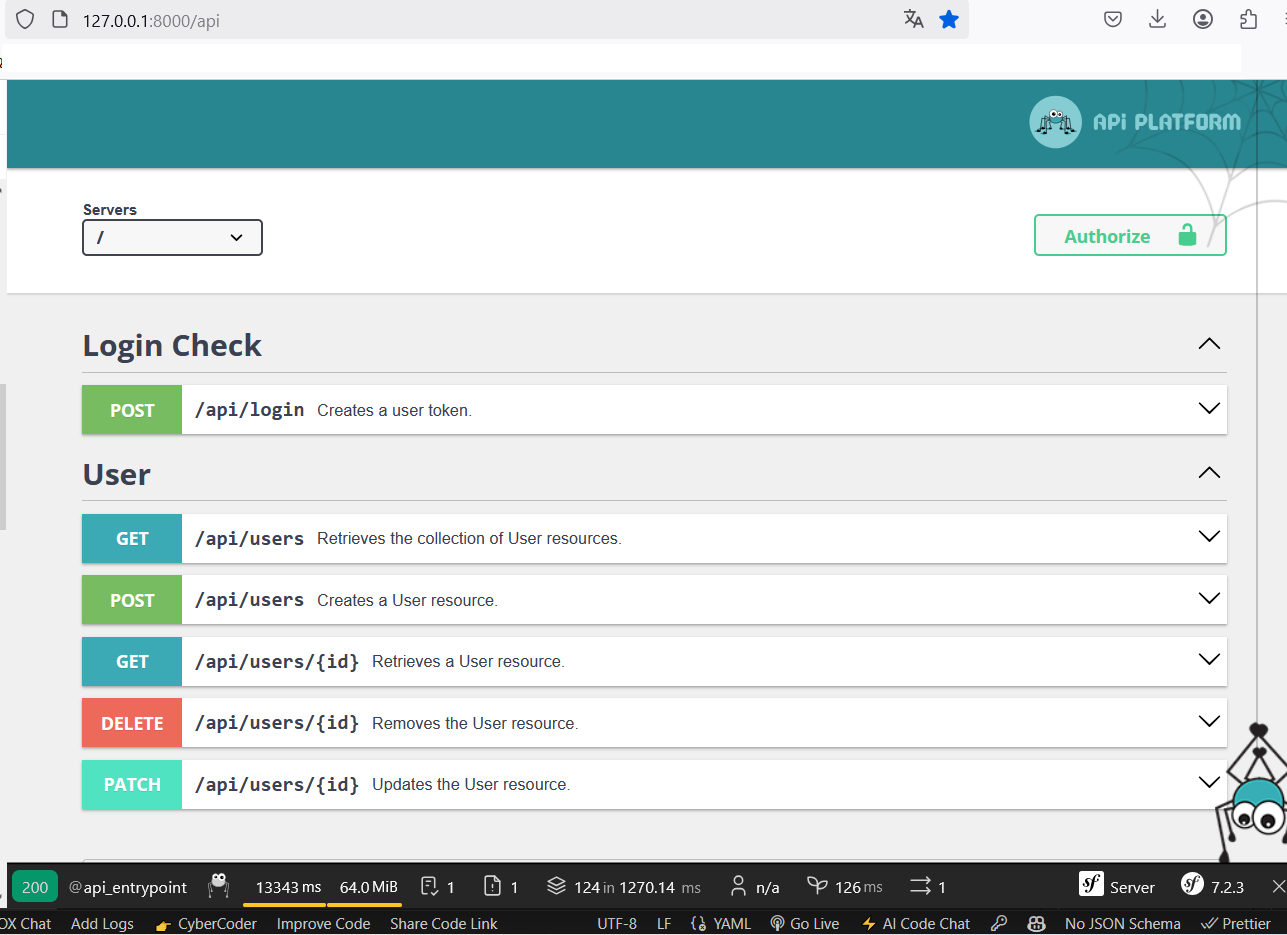
            missing\_import\_mode: warn

### Vérifier l’affichage avec symphony serve

#### Affichage de l’accueil de Symfony à l’adresse <http://127.0.0.1:8000/>

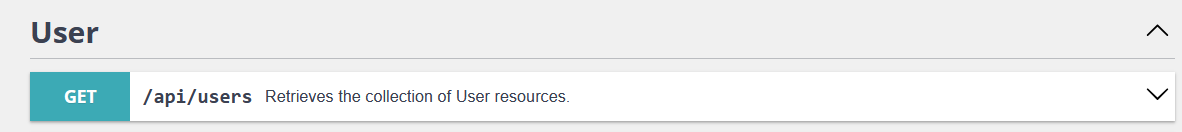


#### Affichage de l’accueil de Api Plaform à l’adresse <http://127.0.0.1:8000/api>

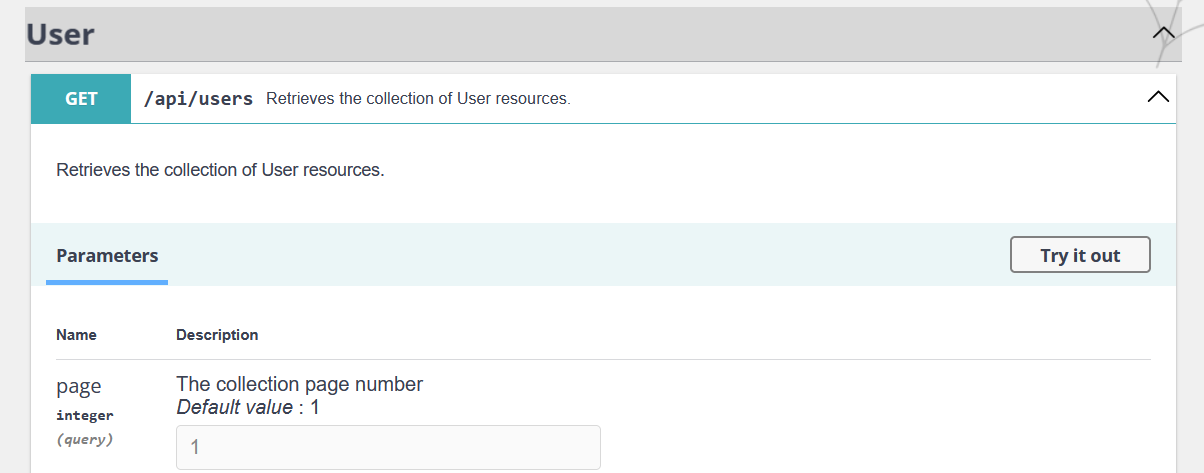


#### Etapes pour afficher les users sur Api Plaform à l’adresse http://127.0.0.1:8000/api#/User/api\_users\_get\_collection

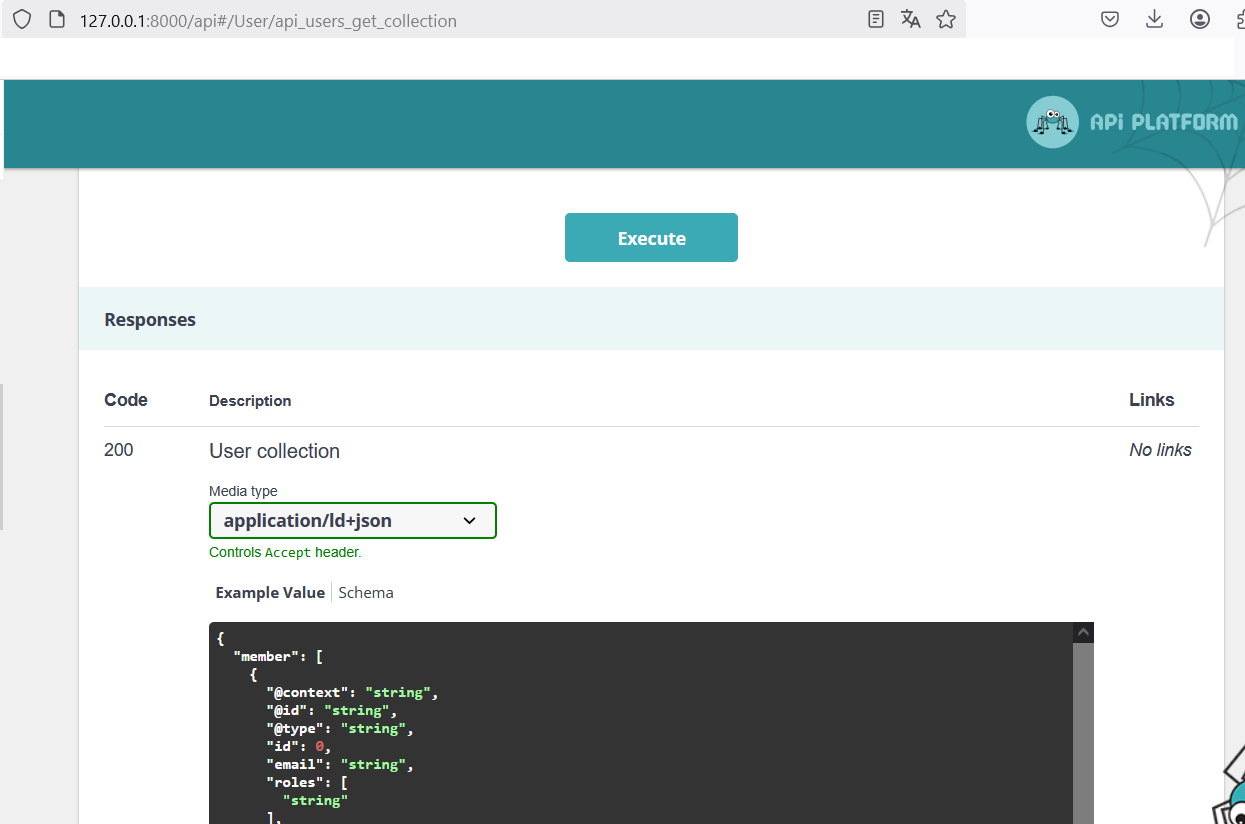
##### Sur la page <http://127.0.0.1:8000/api> cliquer sur GET /api/ users



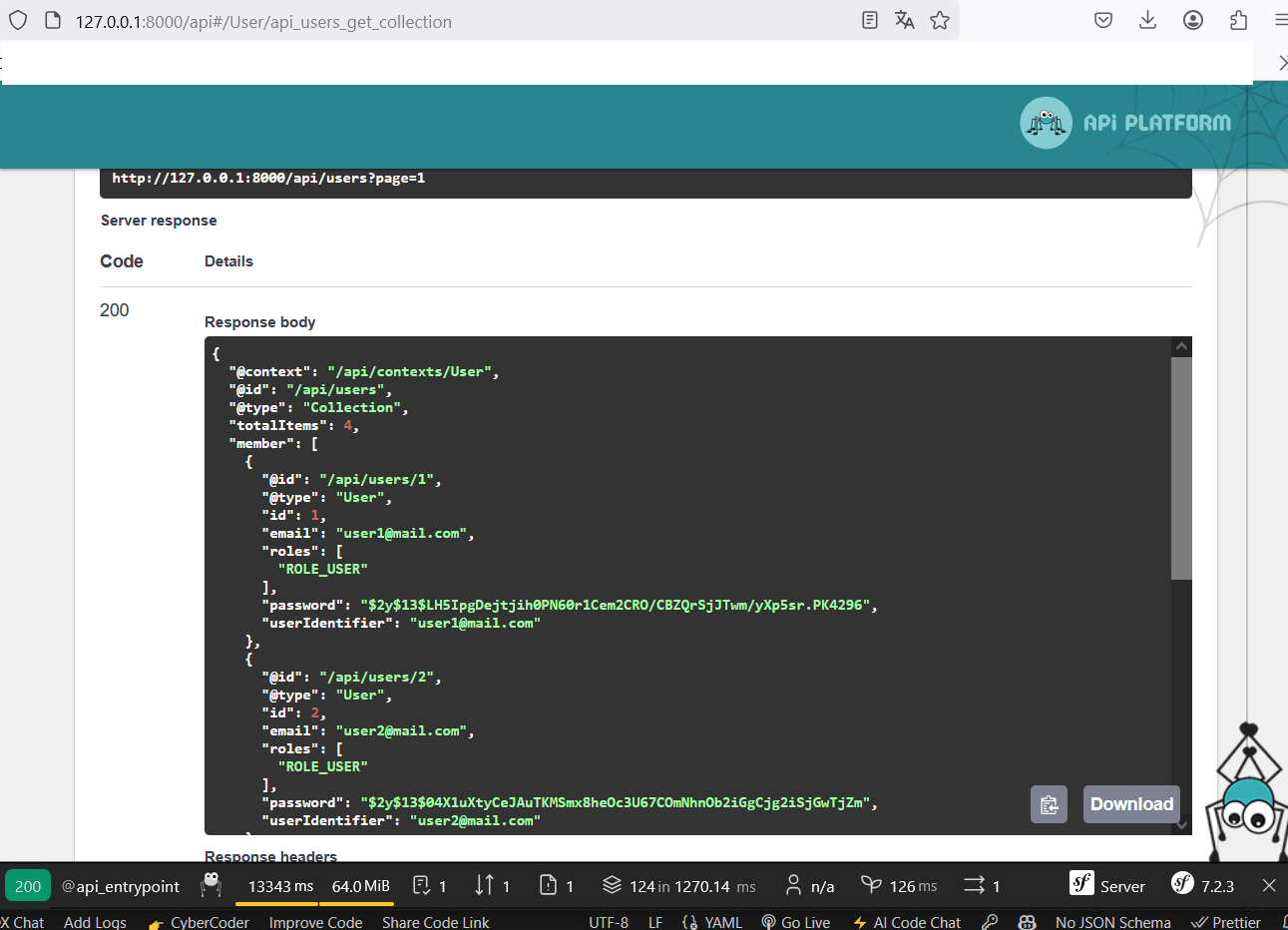
##### Cliquer sur « try it out



##### Cliquer sur execute



##### On obtient  <http://127.0.0.1:8000/api#/User/api_users_get_collection>



## CONFIGURATION DU FRONTEND (ANGULAR)

### 1. Installation des dépendances

cd ../front  
npm install @angular/common @angular/router @angular/forms

### 2. Configuration des modèles

Créer les fichiers suivants avec le contenu fourni : - src/app/models/user.model.ts - src/app/models/auth.model.ts

### 3. Configuration des services

Créer les fichiers suivants avec le contenu fourni : - src/app/services/user.service.ts - src/app/services/auth.service.ts

### 4. Création des composants

Créer les fichiers suivants avec le contenu fourni : - src/app/login/login.component.ts - src/app/welcome/welcome.component.ts - src/app/user-list/user-list.component.ts

### 5. Configuration du routing

Modifier src/app/app.routes.ts avec le contenu fourni.

### 6. Configuration du composant principal

Modifier src/app/app.component.ts avec le contenu fourni.

### 7. Configuration de l’intercepteur HTTP

Créer src/app/interceptors/auth.interceptor.ts :

import { Injectable } from '@angular/core';  
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@angular/common/http';  
import { Observable } from 'rxjs';  
  
@Injectable()  
export class AuthInterceptor implements HttpInterceptor {  
 intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  
 const token = localStorage.getItem('token');  
   
 if (token) {  
 request = request.clone({  
 setHeaders: {  
 Authorization: `Bearer ${token}`  
 }  
 });  
 }  
   
 return next.handle(request);  
 }  
}

## Test de l’application

### 1. Démarrer le backend

cd back  
symfony server:start

### 2. Démarrer le frontend

cd front  
ng serve

### 3. Tester l’application

1. Ouvrir http://localhost:4200 dans un navigateur
2. Se connecter avec les identifiants :
   * Email : admin@example.com
   * Mot de passe : admin

## Dépannage courant

1. Erreur CORS :
   * Vérifier la configuration dans nelmio\_cors.yaml
   * Vérifier que le frontend tourne sur http://localhost:4200
2. Erreur d’authentification :
   * Vérifier que les clés JWT ont été générées
   * Vérifier les identifiants dans les fixtures
   * Vérifier la configuration JWT dans Symfony
3. Erreur de base de données :
   * Vérifier la configuration dans .env
   * Vérifier que les migrations sont à jour
   * Vérifier que les fixtures sont chargées

# INTERACTIONS ENTRE SYMFONY ET ANGULAR

### 1. Flux d’authentification

#### Étape 1 : Tentative de connexion

* **Frontend (Angular)**
  + L’utilisateur saisit ses identifiants dans le composant LoginComponent
  + Le AuthService envoie une requête POST à http://127.0.0.1:8000/api/login
  + Corps de la requête : { username: "email@example.com", password: "motdepasse" }
* **Backend (Symfony)**
  + Le SecurityController reçoit la requête sur la route /api/login
  + Le système de sécurité Symfony vérifie les identifiants
  + Si valides : génère un token JWT
  + Renvoie le token dans la réponse
* **Frontend (Angular)**
  + Reçoit le token JWT
  + Le stocke dans localStorage
  + Redirige vers la page d’accueil

sequenceDiagram  
 participant U as Utilisateur  
 participant A as Angular  
 participant S as Symfony  
 U->>A: Saisit identifiants  
 A->>S: POST /api/login  
 S->>S: Vérifie identifiants  
 S-->>A: Retourne JWT token  
 A->>A: Stocke token  
 A->>U: Redirige vers accueil

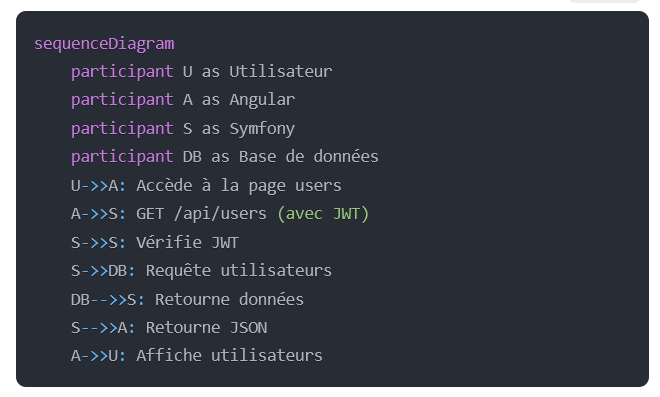


### 2. Flux de récupération des utilisateurs

#### Étape 1 : Chargement de la liste

* **Frontend (Angular)**
  + Le UserListComponent s’initialise
  + Le UserService envoie une requête GET à http://127.0.0.1:8000/api/users
  + L’AuthInterceptor ajoute automatiquement le token JWT dans l’en-tête
* **Backend (Symfony)**
  + Vérifie la validité du token JWT
  + L’ApiResource de l’entité User traite la requête
  + Retourne la collection d’utilisateurs
* **Frontend (Angular)**
  + Reçoit les données et met à jour l’affichage

sequenceDiagram  
 participant U as Utilisateur  
 participant A as Angular  
 participant S as Symfony  
 participant DB as Base de données  
 U->>A: Accède à la page users  
 A->>S: GET /api/users (avec JWT)  
 S->>S: Vérifie JWT  
 S->>DB: Requête utilisateurs  
 DB-->>S: Retourne données  
 S-->>A: Retourne JSON  
 A->>U: Affiche utilisateurs



### 3. Points d’interaction clés

#### API Platform (Symfony)

# Points d'entrée API générés automatiquement  
GET /api/users # Liste des utilisateurs  
GET /api/users/{id} # Détails d'un utilisateur  
POST /api/login # Authentification

#### Intercepteur HTTP (Angular)

// Ajout automatique du token JWT à chaque requête  
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1...

### 4. Gestion des erreurs

#### Cas 1 : Échec d’authentification

* **Frontend** : Affiche le message d’erreur “Identifiants non reconnus”
* **Backend** : Renvoie une erreur 401 Unauthorized

#### Cas 2 : Token expiré

* **Frontend** : Redirige vers la page de login
* **Backend** : Renvoie une erreur 401 Unauthorized

#### Cas 3 : Erreur CORS

* **Frontend** : Console affiche une erreur de CORS
* **Backend** : Les en-têtes CORS ne correspondent pas à la configuration

### 5. Sécurité

#### Protection des routes

* **Symfony** : Utilise les annotations de sécurité et le firewall JWT

access\_control:  
 - { path: ^/api/login, roles: PUBLIC\_ACCESS }  
 - { path: ^/api, roles: IS\_AUTHENTICATED\_FULLY }

* **Angular** : Utilise le authGuard pour protéger les routes

{  
 path: 'welcome',  
 component: WelcomeComponent,  
 canActivate: [authGuard]  
}

### 6. Cycle de vie d’une requête

1. **Préparation de la requête (Angular)**
   * Construction de la requête HTTP
   * Ajout du token par l’intercepteur
   * Ajout des en-têtes CORS
2. **Traitement par Symfony**
   * Vérification des en-têtes CORS
   * Validation du token JWT
   * Exécution de la logique métier
   * Préparation de la réponse
3. **Traitement de la réponse (Angular)**
   * Gestion des erreurs éventuelles
   * Mise à jour du state de l’application
   * Rafraîchissement de l’interface

### 7. État de l’application

#### Côté Angular

* Token JWT stocké dans localStorage
* État d’authentification géré par BehaviorSubject
* Liste des utilisateurs dans le component

#### Côté Symfony

* Session stateless (pas de session serveur)
* Validation du token à chaque requête
* Gestion des permissions basée sur les rôles

### 8. Bonnes pratiques

1. **Sécurité**
   * Ne jamais stocker de données sensibles côté client
   * Toujours valider les données côté serveur
   * Utiliser HTTPS en production
2. **Performance**
   * Minimiser les requêtes API
   * Mettre en cache les réponses quand possible
   * Pagination pour les grandes listes
3. **Maintenance**
   * Centraliser la configuration des URLs
   * Utiliser des interfaces pour les modèles