

1	GUIDE D'INSTALLATION - SYMFONY 7 & ANGULAR 19 AVEC AUTHENTIFICATION JWT	4
1.1	Objectif.....	4
1.2	Prérequis	4
1.3	CRÉATION DU PROJET	4
1.3.1	Création de la structure du projet	4
1.3.2	Créer un nouveau projet Symfony	4
1.3.3	Installer les dépendances nécessaires à Symfony	4
1.3.4	3. Installation du frontend Angular	5
1.3.5	installer Angular Cli uniquement dans le projet	5
1.3.6	Créer un nouveau projet Angular	5
1.3.7	Vérifier la version d'Angular	5
1.3.8	Si besoin, faire un upgrade de la version d'Angular	5
1.4	CONFIGURATION DU BACKEND (SYMFONY)	6
1.4.1	Configuration de la base de données.....	6
1.4.2	Configuration CORS	6
1.4.2.1	Exemple de cors	6
1.4.3	Configuration JWT	7
1.4.3.1	Exemple dans .env.....	7
1.4.4	Créer/modifier config/packages/lexik_jwt_authentication.yaml :....	7
1.4.4.1	Exemple lexik_jwt_authentication.	8
1.4.5	Générer les clés JWT :	8
1.4.6	COMPLEMENT - :INTERACTIONS ENTRE LE FICHER .ENV ET LEXIK_JWT_AUTHENTICATION.YAML	8
1.4.6.1	Résumé	8
1.4.6.2	Résumé - simplifié.....	8
1.4.6.3	Résumé – plus détaillé.....	9
1.4.6.4	Flux.....	9
1.4.6.5	Schéma.....	9
1.4.6.6	Détails	10
1.4.6.6.1	Fichier .env.....	10

1.4.6.6.2	Fichier lexik_jwt_authentication.yaml	10
1.4.6.6.3	Interaction entre .env et lexik_jwt_authentication.yaml.....	11
1.4.6.6.4	Résolution des chemins	11
1.4.7	Configuration de la sécurité / fichier config/packages/security.yaml.....	12
1.4.8	Création de l'entité User	13
1.4.9	Adapter l'entité User à API Platform	13
1.4.10	Vérifier que Doctrine a détecté les changements et faire la migration	17
1.4.11	Création des fixtures	17
1.4.12	Charger les fixtures dans la base de données.....	18
1.4.13	Création du contrôleur d'authentification.....	18
1.4.13.1	Exemple du SecurityController	19
1.4.13.2	Explications du SecurityController.....	19
1.5	RESOUDRE LE PROBLEME D’AFFICHAGE DE API PLATFORM DANS WINDOWS..	22
1.5.1	Aller dans config/packages/asset_mapper.yaml	22
1.5.2	fichier original asset_mapper.yaml	23
1.5.3	Modifier ass_mapper.yaml comme suit	24
1.5.4	Vérifier l’affichage avec symphony serve	24
1.5.4.1	Affichage de l'accueil de Symfony à l'adresse http://127.0.0.1:8000/	24
1.5.4.2	Affichage de l'accueil de Api Plaform à l'adresse http://127.0.0.1:8000/api	25
1.5.4.3	Etapas pour afficher les users sur Api Plaform à l'adresse http://127.0.0.1:8000/api#/User/api_users_get_collection.....	25
1.5.4.3.1	Sur la page http://127.0.0.1:8000/api cliquer sur GET /api/ users.....	25
1.5.4.3.2	Cliquer sur « try it out »	26
1.5.4.3.3	Cliquer sur execute	26
1.5.4.3.4	On obtient http://127.0.0.1:8000/api#/User/api_users_get_collection	27
1.6	CONFIGURATION DU FRONTEND (ANGULAR)	28
1.6.1	Installation des dépendances	28
1.6.2	Configuration des modèles.....	28
1.6.2.1	Utilité des fichiers « models »	29
1.6.2.2	Fichier src/app/models/user.model.ts à utiliser	29
1.6.2.3	Fichier src/app/models/auth.model.ts à utiliser	30
1.6.3	Configuration des services.....	30
1.6.3.1	Utilité des fichiers « services »	31

1.6.3.2	Utilité de src/app/services/user.service.ts.....	31
1.6.3.3	Fichier src/app/services/user.service.ts à utiliser	31
1.6.3.4	Utilité de src/app/services/auth.service.ts	32
1.6.3.5	Fichier src/app/services/auth.service.ts à utiliser	33
1.6.4	Création des composants	34
1.6.4.1	Fichier src/app/login/login.component.ts à utiliser.....	35
1.6.4.2	Fichier src/app/welcome/welcome.component.ts à utiliser	37
1.6.4.3	Fichier src/app/user-list/user-list.component.ts à utiliser	39
1.6.4.4	Fichier src/app/users/users.component.ts à utiliser	41
1.6.5	Créer le dossier « guards » et configurer auth.guards.ts	42
1.6.5.1	Fichier src/app/guards/auth.guard.ts à utiliser	43
1.6.6	Configuration du routing.....	44
1.6.6.1	Fichier src/app/app.routes.ts à utiliser :	44
1.6.7	Configuration du composant principal	45
1.6.7.1	Fichier src/app/app.component.ts à utiliser.....	45
1.6.8	Modifier le fichier app.component.html comme suit	45
1.6.9	Modifier app.config.ts comme suit :	46
1.7	Test de l'application	46
1.7.1	1. Démarrer le backend.....	46
1.7.2	2. Démarrer le frontend	46
1.7.3	3. Tester l'application	47
2	INTERACTIONS ENTRE SYMFONY ET ANGULAR	47
2.1	Flux d'authentification	47
2.1.1	Étape 1 : Tentative de connexion.....	47
2.2	Flux de récupération des utilisateurs.....	48
2.2.1	Étape 1 : Chargement de la liste	48
2.3	Points d'interaction clés.....	49
2.3.1	API Platform (Symfony).....	49
2.3.2	Intercepteur HTTP (Angular)	49
2.4	Sécurité.....	49
2.4.1	Protection des routes.....	49
2.5	Cycle de vie d'une requête	50
2.6	État de l'application.....	50
2.6.1	Côté Angular	50

2.6.2	Côté Symfony	50
-------	--------------------	----

1 GUIDE D'INSTALLATION - SYMFONY 7 & ANGULAR 19 AVEC AUTHENTIFICATION JWT

1.1 Objectif

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

1.2 Prérequis

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

1.3 CRÉATION DU PROJET

1.3.1 Création de la structure du projet

Créer le dossier principal

```
mkdir symang1
```

```
cd symang1
```

1.3.2 Créer un nouveau projet Symfony

```
symfony new project_name --webapp
```

Ici :

```
Symfony new back --webapp
```

1.3.3 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
```

1.3.4 3. Installation du frontend Angular

```
mkdir front
```

se placer dans ce dossier

```
cd .. front
```

1.3.5 installer Angular Cli uniquement dans le projet

Se placer dans le dossier racine du projet symang1

Exécuter :

```
npm install @angular/cli --save-dev
```

1.3.6 Créer un nouveau projet Angular

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

1.3.7 Vérifier la version d'Angular

```
npx ng version
```

ou dans package.json

```
{
  "devDependencies": {
    "@angular/cli": "^19.1.6"
  }
}
```

1.3.8 Si besoin, faire un upgrade de la version d'Angular

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

1.4 CONFIGURATION DU BACKEND (SYMFONY)

1.4.1 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/symang1?serverVersion=8.0.32&charsets=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
```

1.4.2 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
```

1.4.2.1 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
```

1.4.3 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.

1.4.3.1 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 ###
```

1.4.4 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
```

1.4.4.1 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
```

1.4.5 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.

1.4.6 COMPLEMENT - :INTERACTIONS ENTRE LE FICHIER `.ENV` ET `LEXIK_JWT_AUTHENTICATION.YAML`

1.4.6.1 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.

1.4.6.2 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.

1.4.6.3 *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.

1.4.6.4 *Flux*

1. Point de départ : Fichier .env

- Contient les variables d'environnement brutes
- Définit JWT_SECRET_KEY et JWT_PUBLIC_KEY

2. Fichier de configuration : lexik_jwt_authentication.yaml

- Utilise les variables via des placeholders
- Format : %env(resolve:JWT_SECRET_KEY)%

3. 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

4. Stockage final

- private.pem : stocke la clé privée
- public.pem : stocke la clé publique

1.4.6.5 *Schéma*

```
[ Fichier .env ]
↓
(Contient JWT_SECRET_KEY, JWT_PUBLIC_KEY)
↓
[ lexik_jwt_authentication.yaml ]
↓
(Utilise %env(resolve:JWT_SECRET_KEY)%)
↓
[ Processus de résolution Symfony ]
↓
(Lit les variables, Convertit les chemins)
↓
Chemins Absolus
↓
[ Stockage final ]
↓           ↓
[ private.pem ] [ public.pem ]
(Clé Privée)    (Clé Publique)
```

1.4.6.6 Détails

1.4.6.6.1 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.

1.4.6.6.2 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).

1.4.6.6.3 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`

1.4.6.6.4 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.

1.4.7 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
```

1.4.8 Création de l'entité User

php bin/console make:entity

ici

php bin/console make:user

1.4.9 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;
}
}

```


1.4.10 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

1.4.11 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();
}
```

1.4.12 Charger les fixtures dans la base de données

`php bin/console doctrine:fixtures:load`

S'assurer que tout est en ordre :

`php bin/console doctrine:schema:validate`

1.4.13 Création du contrôleur d'authentification

Exécuter

`php bin/console make:controller SecurityController`

Ce contrôleur 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

1.4.13.1 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()
        ]);
    }
}
```

1.4.13.2 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']`.

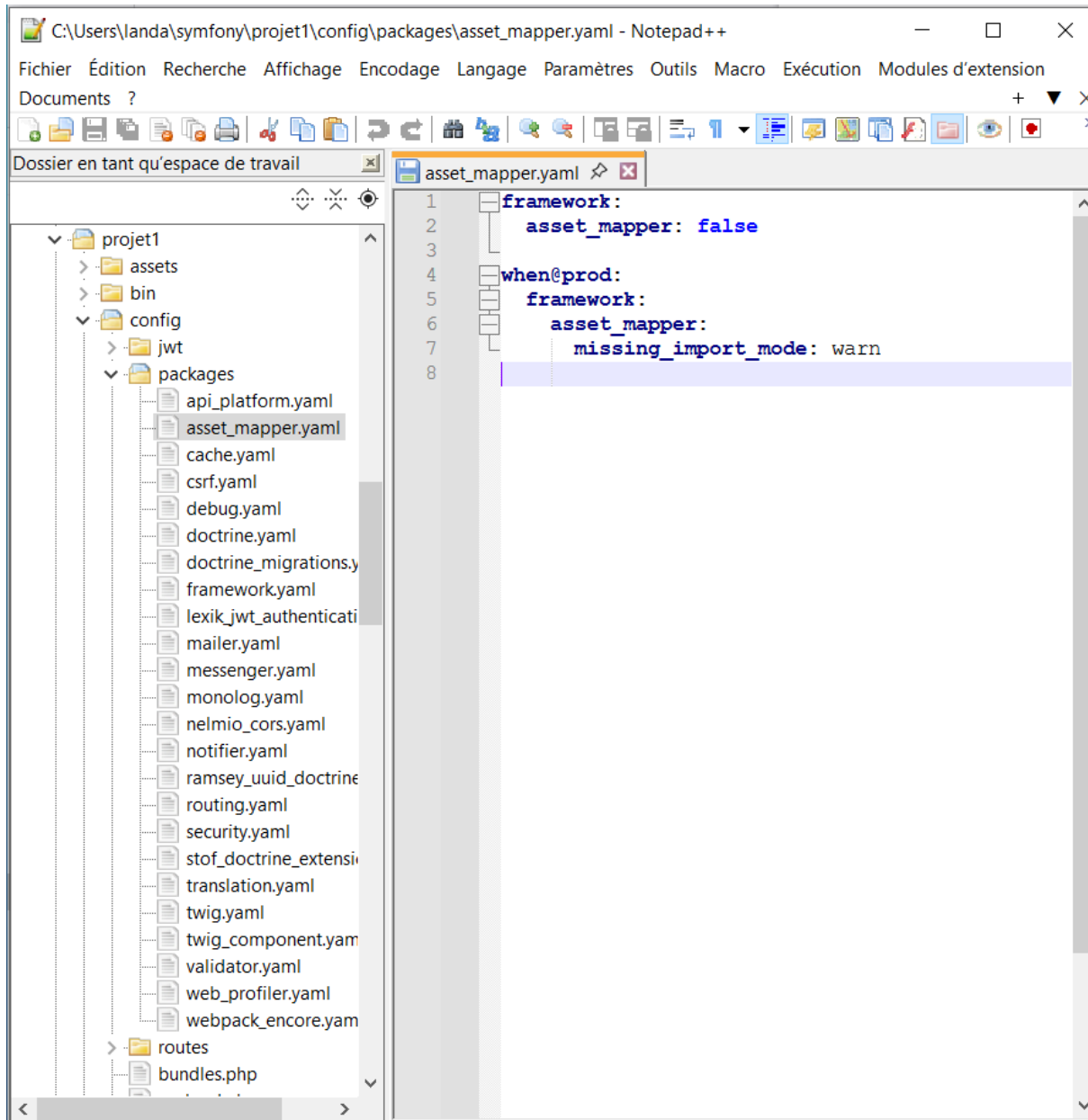
```
]); Fermeture du tableau contenant les données à renvoyer dans la réponse JSON.
```

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

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

Si API Platform ne s’affiche pas correctement,

1.5.1 Aller dans `config/packages/asset_mapper.yaml`



1.5.2 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
```

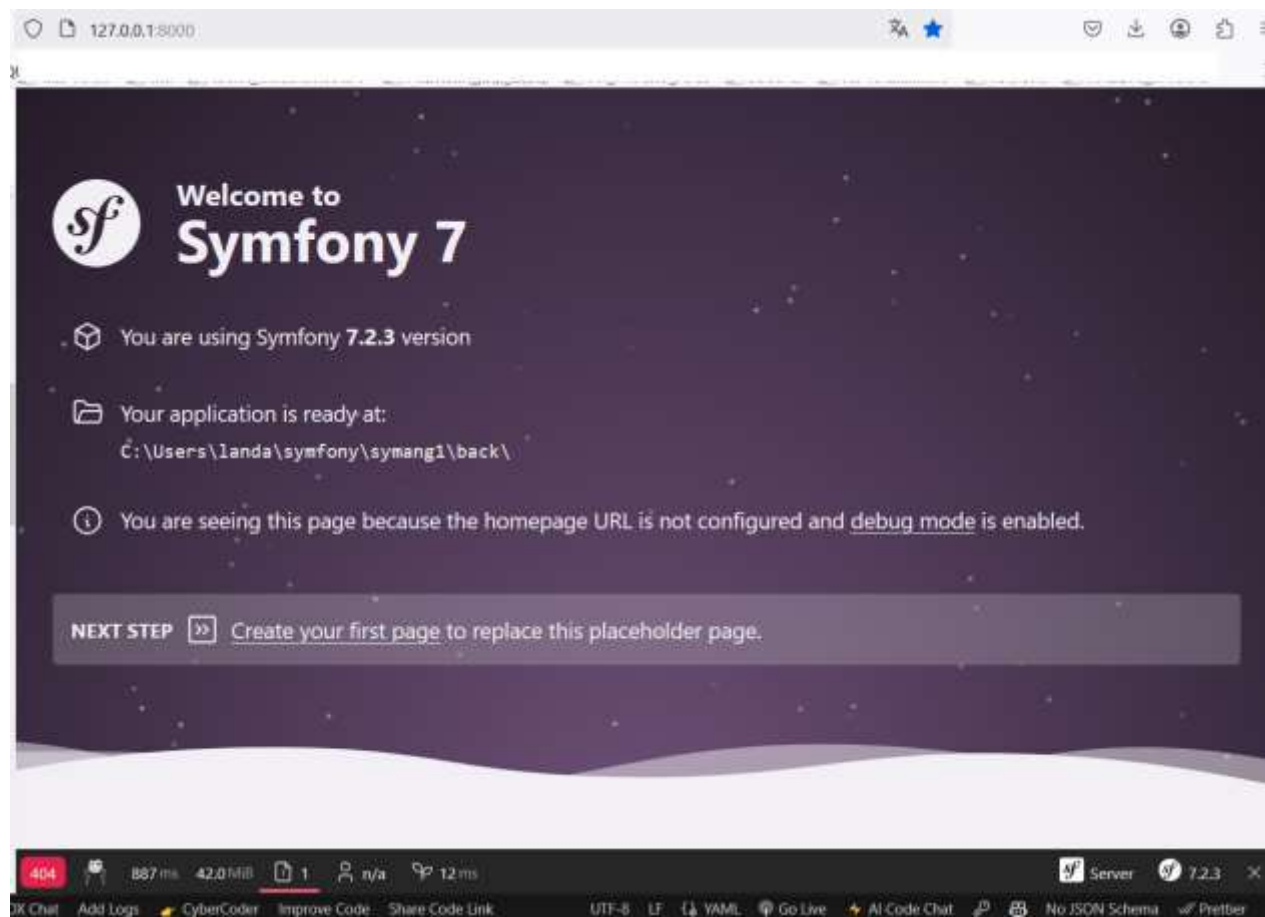
1.5.3 Modifier `ass_mapper.yaml` comme suit

```
framework:
  asset_mapper: false

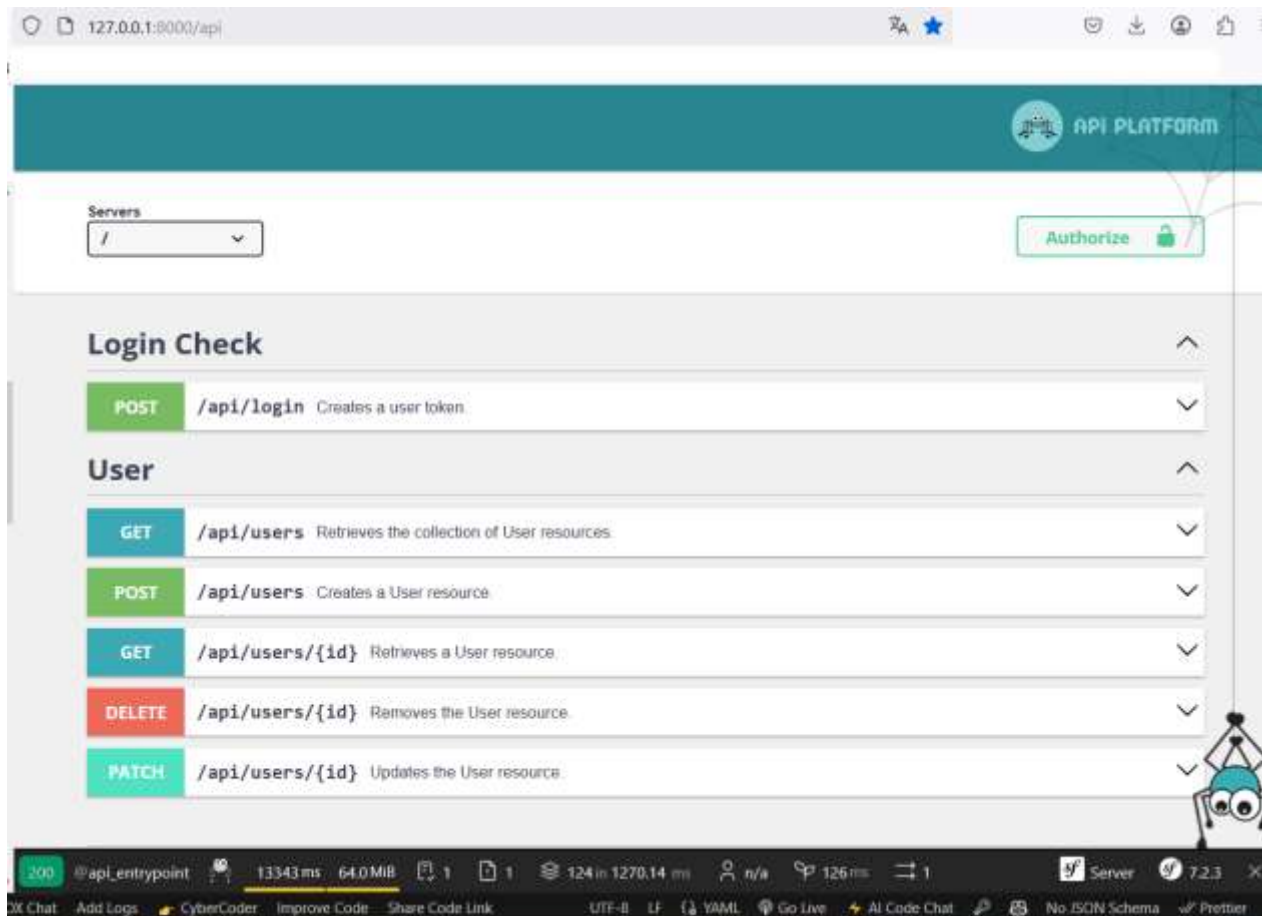
when@prod:
  framework:
    asset_mapper:
      missing_import_mode: warn
```

1.5.4 Vérifier l'affichage avec `symphony serve`

1.5.4.1 Affichage de l'accueil de Symfony à l'adresse `http://127.0.0.1:8000/`

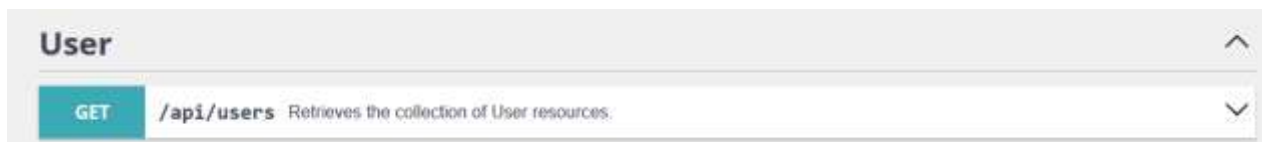


1.5.4.2 Affichage de l'accueil de Api Plaform à l'adresse <http://127.0.0.1:8000/api>

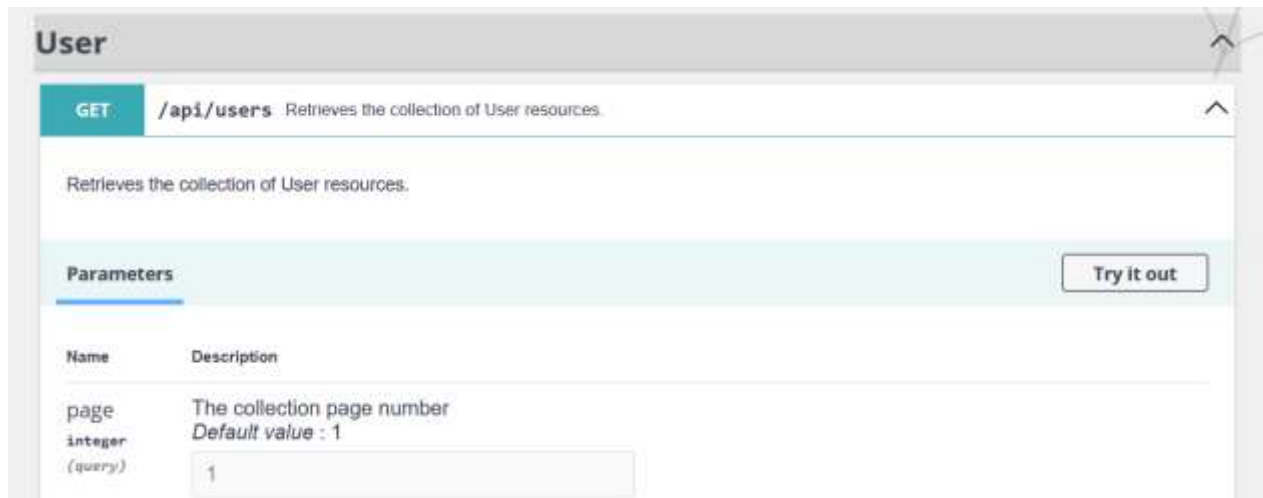


1.5.4.3 Etapes pour afficher les users sur Api Plaform à l'adresse http://127.0.0.1:8000/api#/User/api_users_get_collection

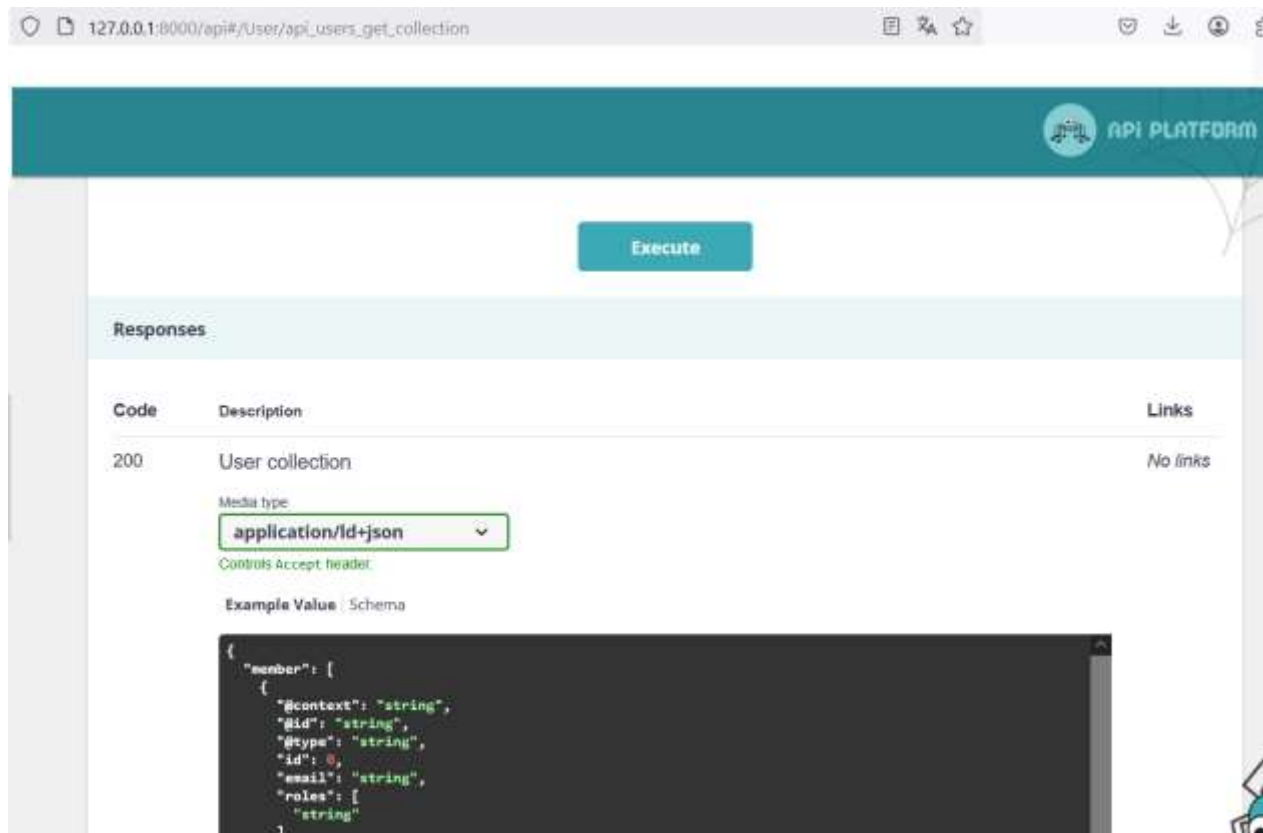
1.5.4.3.1 Sur la page <http://127.0.0.1:8000/api> cliquer sur GET /api/ users



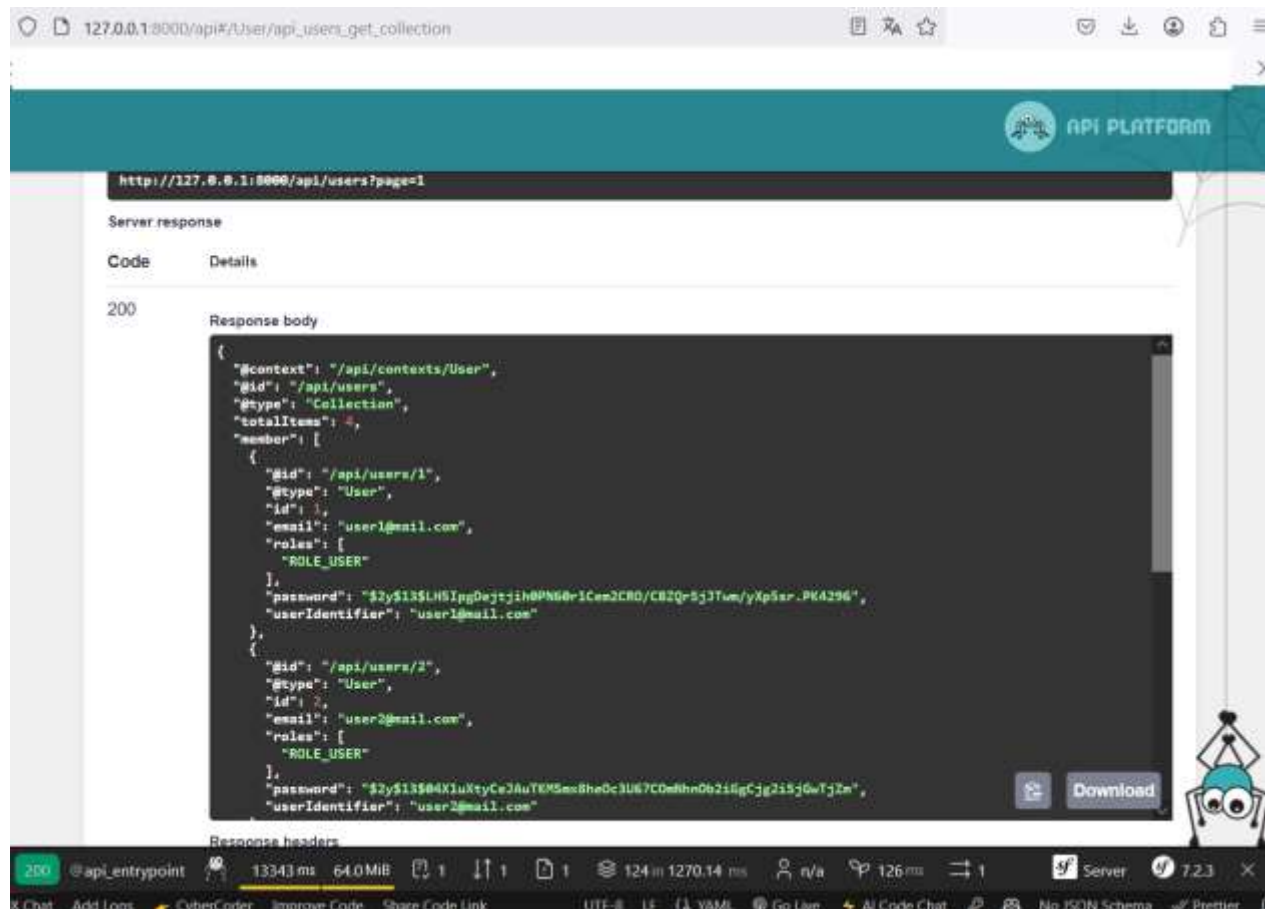
1.5.4.3.2 Cliquer sur « try it out »



1.5.4.3.3 Cliquer sur execute



1.5.4.3.4 On obtient http://127.0.0.1:8000/api#/User/api_users_get_collection



The screenshot shows an API client interface with a teal header labeled "API PLATFORM". The address bar displays the URL `http://127.0.0.1:8000/api#/User/api_users_get_collection`. Below the header, the "Server response" section shows a status code of `200`. The "Response body" is a JSON array containing two user objects. The first user has an ID of 1, email `user1@mail.com`, and a password starting with `$2y$13$`. The second user has an ID of 2, email `user2@mail.com`, and a password starting with `$2y$13$`. The interface also shows "Response headers" and a status bar at the bottom with various metrics like response time (13343 ms) and size (64.0 MIB).

```
{
  "@context": "/api/contexts/User",
  "@id": "/api/users",
  "@type": "Collection",
  "totalItems": 2,
  "member": [
    {
      "@id": "/api/users/1",
      "@type": "User",
      "id": 1,
      "email": "user1@mail.com",
      "roles": [
        "ROLE_USER"
      ],
      "password": "$2y$13$1H5IpgDejtjiH0PN60r1Cem2CR0/C82Qr-5j3Twm/yXp5er.PK4296",
      "userIdentifier": "user1@mail.com"
    },
    {
      "@id": "/api/users/2",
      "@type": "User",
      "id": 2,
      "email": "user2@mail.com",
      "roles": [
        "ROLE_USER"
      ],
      "password": "$2y$13$04XluXtyCa3AuTKMSmc8he0c3U67C0nd8hnOb2i6gCjg2i5j0wTj2e",
      "userIdentifier": "user2@mail.com"
    }
  ]
}
```

1.6 CONFIGURATION DU FRONTEND (ANGULAR)

1.6.1 Installation des dépendances

```
cd ../front
```

```
npm install @angular/common @angular/router @angular/forms
```

La commande `npm install @angular/common @angular/router @angular/forms` est utilisée pour installer trois modules spécifiques d'Angular :

- **@angular/common** : Ce module contient des services et des fonctionnalités courantes qui sont utilisés par plusieurs parties de l'application Angular.
- **@angular/router** : Ce module gère la navigation et le routage entre différentes vues ou pages de l'application.
- **@angular/forms** : Ce module fournit des outils pour créer et gérer des formulaires dans l'application.

Certains modules sont inclus de base lors de la création d'un nouveau projet Angular. Cependant, il est courant d'installer explicitement certains modules, surtout lorsqu'ils sont essentiels pour le développement de fonctionnalités spécifiques. En ajoutant `@angular/common`, `@angular/router` et `@angular/forms` au projet via la commande `npm install`, on s'assure que l'on utilise les dernières versions de ces modules et que toutes les dépendances nécessaires sont correctement gérées.

1.6.2 Configuration des modèles

Créer manuellement le dossier « models » dans `src/app`

Puis y créer les fichiers `auth.model.ts` et `user.model.ts`.

On peut aussi générer un fichier directement dans un dossier. Si le dossier n'existe pas, Angular le créera automatiquement.

```
ng g interface models/auth --type=model
```

```
ng g interface models/user --type=model
```

Ces deux commandes vont générer le dossier “models” ainsi que les deux fichiers suivants :

```
export interface User {  
}
```

```
export interface Auth {  
}
```

1.6.2.1 Utilité des fichiers « models »

Définissent des interfaces TypeScript pour structurer les données échangées entre Angular et API Symfony. Cela permet de s'assurer que les données reçues et envoyées respectent un certain format.

Les fichiers models définissent des interfaces TypeScript qui servent à structurer et typer les données échangées entre l'application Angular et l'API Symfony.

1.6.2.2 Fichier `src/app/models/user.model.ts` à utiliser

```
//modèle pour l'utilisateur  
//src/app/models/user.model.ts  
// Déclaration de l'interface User qui représente les informations d'un  
// utilisateur  
export interface User {  
  id: number;           // Identifiant unique de l'utilisateur  
  email: string;        // Adresse e-mail de l'utilisateur  
  roles: string[];      // Liste des rôles de l'utilisateur (ex:  
  ['ROLE_USER', 'ROLE_ADMIN'])  
  userIdentifier: string; // Identifiant de l'utilisateur (peut être un nom  
  // d'utilisateur ou autre identifiant)  
}  
  
// Déclaration de l'interface ApiResponse qui représente la réponse de l'API pour  
// une liste d'utilisateurs  
export interface ApiResponse {  
  '@context': string;    // Contexte de l'API (informations de métadonnées)  
  '@id': string;         // Identifiant de l'élément API  
  '@type': string;       // Type de l'élément API  
  totalItems: number;    // Nombre total d'éléments dans la réponse  
  member: User[];        // Tableau des utilisateurs (membres) retournés  
  // dans la réponse  
}
```

1.6.2.3 Fichier `src/app/models/auth.model.ts` à utiliser

```
//modèle pour l'authentification
// src/app/models/auth.model.ts
// Déclaration de l'interface LoginRequest qui représente la requête pour une
// authentification
export interface LoginRequest {
  username: string;          // Nom d'utilisateur de l'utilisateur
  password: string;          // Mot de passe de l'utilisateur
}

// Déclaration de l'interface LoginResponse qui représente la réponse de l'API
// lors de l'authentification
export interface LoginResponse {
  token: string;             // Jeton d'authentification retourné par l'API
}
```

1.6.3 Configuration des services

Créer les services avec les commandes

```
ng g service services/user
```

```
ng g service services/auth
```

Ces commandes vont créer le dossier `services` ainsi que les deux fichiers suivants :

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  constructor() { }
}
```

```
import { Injectable } from '@angular/core';
```

```
@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor() { }
}
```

1.6.3.1 Utilité des fichiers « services »

Centralisation de la logique métier : Les services permettent de centraliser et d'organiser la logique métier et les interactions avec l'API.

Communication avec l'API : Les services Angular utilisent `HttpClient` pour effectuer des requêtes HTTP (GET, POST, etc.) vers l'API Symfony. Ils gèrent la récupération des données depuis l'API et la transmission des données vers l'API.

1.6.3.2 Utilité de `src/app/services/user.service.ts`

Ce fichier définit un service Angular appelé `UserService` qui interagit avec l'API pour obtenir des informations sur les utilisateurs.

Il centralise la logique pour effectuer des requêtes HTTP à l'API et récupérer des données sur les utilisateurs.

- `apiUrl`: L'URL de l'API où les informations sur les utilisateurs sont disponibles.
- `getUsers()`: Méthode qui envoie une requête GET à l'API et renvoie la réponse contenant les informations sur les utilisateurs sous forme d'`Observable<ApiResponse>`.

Ce service `UserService` effectue des requêtes HTTP à une API pour obtenir des informations sur les utilisateurs.

Il utilise `HttpClient` pour effectuer la requête GET et renvoie un `Observable` de type `ApiResponse` qui peut être utilisé par les composants pour souscrire à cette réponse.

1.6.3.3 Fichier `src/app/services/user.service.ts` à utiliser

```
//créer le service UserService
// src/app/services/user.service.ts
import { Injectable } from '@angular/core'; // Importe le décorateur Injectable d'Angular
import { HttpClient } from '@angular/common/http'; // Importe HttpClient pour effectuer des requêtes HTTP
import { Observable } from 'rxjs'; // Importe Observable de RxJS
import { ApiResponse } from '../models/user.model'; // Importe l'interface ApiResponse

@Injectable({ providedIn: 'root' }) // Déclare le service comme étant injectable partout dans l'application
export class UserService {
  private apiUrl = 'http://127.0.0.1:8000/api/users'; // URL de l'API utilisateur

  constructor(private http: HttpClient) {} // Injection du service HttpClient dans le constructeur

  getUsers(): Observable<ApiResponse> { // Déclaration de la méthode getUsers qui renvoie un Observable d'ApiResponse
    return this.http.get<ApiResponse>(this.apiUrl); // Effectue une requête GET à l'API et renvoie la réponse
  }
}
```

1.6.3.4 Utilité de `src/app/services/auth.service.ts`

Ce fichier définit un service Angular appelé `AuthService` qui gère l'authentification des utilisateurs.

Il centralise la logique pour se connecter, se déconnecter et suivre l'état d'authentification de l'utilisateur.

- `apiUrl`: L'URL de base de l'API d'authentification.
- `isAuthenticatedSubject`: Un `BehaviorSubject` qui maintient l'état d'authentification de l'utilisateur.
- `login(credentials: LoginRequest)`: Méthode qui envoie une requête POST à l'API pour l'authentification, stocke le token reçu dans le `localStorage` et met à jour l'état d'authentification.
- `logout()`: Méthode qui supprime le token du `localStorage` et met à jour l'état d'authentification pour indiquer que l'utilisateur n'est plus authentifié.
- `isAuthenticated()`: Méthode qui renvoie l'état actuel d'authentification de l'utilisateur.

En résumé, `UserService` gère la récupération des données utilisateur depuis l'API, tandis que `AuthService` gère l'authentification des utilisateurs et maintient l'état d'authentification au sein de l'application Angular.

Ce service `AuthService` gère l'authentification des utilisateurs.

Il permet de se connecter (`login`), de se déconnecter (`logout`) et de vérifier l'état d'authentification (`isAuthenticated`).

Il utilise `HttpClient` pour envoyer des requêtes POST pour la connexion et utilise `BehaviorSubject` pour suivre l'état d'authentification de l'utilisateur en temps réel.

1.6.3.5 Fichier `src/app/services/auth.service.ts` à utiliser

```
// Créer un service d'authentification
// src/app/services/auth.service.ts
import { Injectable } from '@angular/core'; // Importe le décorateur Injectable
d'Angular
import { HttpClient } from '@angular/common/http'; // Importe HttpClient pour
effectuer des requêtes HTTP
import { BehaviorSubject, Observable, tap } from 'rxjs'; // Importe
BehaviorSubject, Observable et tap de RxJS
import { LoginRequest, LoginResponse } from '../models/auth.model'; // Importe
les interfaces LoginRequest et LoginResponse

@Injectable({providedIn: 'root'}) // Déclare le service comme étant injectable
partout dans l'application
export class AuthService {
  private apiUrl = 'http://127.0.0.1:8000/api'; // URL de base de l'API
d'authentification
  private isAuthenticatedSubject = new BehaviorSubject<boolean>(false); // Crée
un BehaviorSubject pour l'état d'authentification
  isAuthenticated$ = this.isAuthenticatedSubject.asObservable(); // Expose le
BehaviorSubject comme un Observable

  constructor(private http: HttpClient) {
    // Vérifie si un token existe au démarrage de l'application
    const token = localStorage.getItem('token');
    this.isAuthenticatedSubject.next(!!token);
  }
}
```

```
login(credentials: LoginRequest): Observable<LoginResponse> { // Déclare la
méthode login qui renvoie un Observable de type LoginResponse
  return this.http.post<LoginResponse>(`${this.apiUrl}/login`, credentials) //
Effectue une requête POST à l'API de login
  .pipe(tap((response) => { // Utilise l'opérateur tap pour effectuer des actions
supplémentaires
    if(response.token){ // Si la réponse contient un token
      localStorage.setItem('token', response.token); // Stocke le token dans le
localStorage
      this.isAuthenticatedSubject.next(true); // Met à jour le BehaviorSubject pour
indiquer que l'utilisateur est authentifié
    }
  }));
}

logout(): void { // Déclare la méthode logout
  localStorage.removeItem('token'); // Supprime le token du localStorage
  this.isAuthenticatedSubject.next(false); // Met à jour le BehaviorSubject pour
indiquer que l'utilisateur n'est plus authentifié
}

isAuthenticated(): boolean { // Déclare la méthode isAuthenticated
  return this.isAuthenticatedSubject.value; // Renvoie la valeur actuelle du
BehaviorSubject
}
}
```

1.6.4 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

Créer les composants suivants :

- Login.component.ts – composant de connexion
- Users.component.ts – composant d’affichage des utilisateurs
- user-list.component.ts - composant détaillé de la liste des utilisateurs
- welcome.component.ts – composant de bienvenue

Utiliser les commandes :

ng generate component login

ng generate component welcome

ng generate component user-list

ng generate component users

1.6.4.1 Fichier `src/app/login/login.component.ts` à utiliser

```
//login.component.ts (Composant de connexion)
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '../services/auth.service';

@Component({
  selector: 'app-login', // Sélecteur pour utiliser ce composant dans l'HTML
  standalone: true,      // Déclare que ce composant est autonome (pas besoin
d'être déclaré dans un module)
  imports: [CommonModule, FormsModule], // Modules nécessaires pour les
directives Angular et les formulaires
  template: `
    <div class="login-container">
      <h2>Connexion</h2>

      <!-- Affiche un message d'erreur si l'authentification échoue -->
      @if (error) {
        <div class="alert alert-danger">
          {{ error }}
        </div>
      }

      <!-- Formulaire de connexion -->
      <form (ngSubmit)="onSubmit()" #loginForm="ngForm">
        <div class="form-group">
          <label for="email">Email:</label>
          <input
            type="email"
```

```

        id="email"
        name="username"
        [(ngModel)]="credentials.username" // Liaison bidirectionnelle des
données
        required
        class="form-control"
    />
</div>

<div class="form-group">
    <label for="password">Mot de passe:</label>
    <input
        type="password"
        id="password"
        name="password"
        [(ngModel)]="credentials.password"
        required
        class="form-control"
    />
</div>

<!-- Bouton de soumission avec un état de chargement -->
<button type="submit" class="btn btn-primary" [disabled]="loading">
    @if (loading) {
        Connexion en cours...
    } @else {
        Se connecter
    }
</button>
</form>
</div>
`,
styles: [`
    /* Styles de base pour le formulaire de connexion */
    .login-container { max-width: 400px; margin: auto; padding: 2rem; border: 1px
solid #ddd; border-radius: 8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); }
    .form-group { margin-bottom: 1rem; }
    .form-control { width: 100%; padding: 0.5rem; border: 1px solid #ddd; border-
radius: 4px; }
    .btn { width: 100%; padding: 0.75rem; background-color: #007bff; color:
white; border: none; border-radius: 4px; cursor: pointer; }
    .btn:disabled { background-color: #ccc; }
    .alert { padding: 0.75rem; margin-bottom: 1rem; border-radius: 4px; }
    .alert-danger { background-color: #f8d7da; color: #721c24; }
`]

```

```

}))
export class LoginComponent {
  credentials = { username: '', password: '' }; // Stocke les informations de
connexion
  loading = false; // Indique si la requête d'authentification est en cours
  error = '';      // Message d'erreur à afficher en cas d'échec de connexion

  constructor(private authService: AuthService, private router: Router) {}

  onSubmit(): void {
    // Vérifie que les champs ne sont pas vides
    if (!this.credentials.username || !this.credentials.password) {
      this.error = 'Veuillez remplir tous les champs';
      return;
    }

    this.loading = true; // Active l'état de chargement
    this.error = '';     // Réinitialise le message d'erreur

    // Appelle le service d'authentification
    this.authService.login(this.credentials).subscribe({
      next: () => {
        this.router.navigate(['/welcome']); // Redirection en cas de succès
      },
      error: (err) => {
        this.error = 'Identifiants non reconnus'; // Affiche un message d'erreur
        this.loading = false;
        console.error('Erreur de connexion:', err);
      }
    });
  }
}

```

1.6.4.2 Fichier src/app/welcome/welcome.component.ts à utiliser

```

//welcome.component.ts (Composant de bienvenue)
// Import des modules nécessaires depuis Angular
import { Component } from '@angular/core';      // Permet de définir un
composant Angular
import { CommonModule } from '@angular/common';  // Fournit des directives
Angular de base comme *ngIf, *ngFor, etc.

```

```

import { AuthService } from '../services/auth.service'; // Service
d'authentification pour gérer la déconnexion
import { Router } from '@angular/router'; // Service de routage pour
naviguer entre les pages de l'application

@Component({
  selector: 'app-welcome', // Nom du sélecteur HTML pour ce composant
  (utilisable comme <app-welcome></app-welcome>)
  standalone: true, // Rend le composant autonome, sans besoin d'être
déclaré dans un module Angular
  imports: [CommonModule], // Import des fonctionnalités communes d'Angular
  template: ` // Définition du template HTML directement dans le
fichier TypeScript
    <div class="welcome-container"> <!-- Conteneur principal pour le style -->
      <h2>Bienvenue!</h2> <!-- Titre de la page de bienvenue -->
      <p>Vous êtes maintenant connecté.</p> <!-- Message d'accueil -->

      <!-- Bouton de déconnexion qui appelle la méthode logout() lors du clic -->
      <button class="btn btn-danger" (click)="logout()">
        Se déconnecter
      </button>
    </div>
  `,
  styles: [`
    /* Styles CSS pour la mise en forme de la page de bienvenue */
    .welcome-container {
      max-width: 600px; /* Limite la largeur du conteneur */
      margin: 2rem auto; /* Centre le conteneur verticalement et
horizontalement */
      padding: 2rem; /* Espace intérieur autour du contenu */
      text-align: center; /* Centre le texte à l'intérieur du conteneur */
    }

    .btn-danger {
      background-color: #dc3545; /* Couleur de fond rouge pour le bouton de
déconnexion */
      color: white; /* Couleur du texte en blanc */
      padding: 0.75rem 1.5rem; /* Espacement intérieur pour agrandir le bouton
*/
      border: none; /* Suppression des bordures par défaut */
      border-radius: 4px; /* Coins légèrement arrondis pour un effet plus
doux */
      cursor: pointer; /* Change le curseur en "main" lors du survol */
    }
  `]
})

```

```
})
export class WelcomeComponent {
  // Injection des services nécessaires via le constructeur
  constructor(
    private authService: AuthService, // Service pour gérer les opérations
    d'authentification (login/logout)
    private router: Router           // Service de navigation pour rediriger
    l'utilisateur après la déconnexion
  ) {}

  // Méthode appelée lorsque l'utilisateur clique sur le bouton "Se déconnecter"
  logout(): void {
    this.authService.logout();          // Appelle la méthode de déconnexion du
    service AuthService
    this.router.navigate(['/login']); // Redirige l'utilisateur vers la page de
    connexion après déconnexion
  }
}
```

1.6.4.3 Fichier `src/app/user-list/user-list.component.ts` à utiliser

```
// user-list.component.ts (Composant détaillé de la liste des utilisateurs)
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClientModule } from '@angular/common/http';
import { UserService } from '../services/user.service';
import { User } from '../models/user.model';

@Component({
  selector: 'app-user-list',
  standalone: true,
  imports: [CommonModule, HttpClientModule],
  template: `
    <div class="container">
      <h3 class="text-center">Liste des utilisateurs</h3>

      <!-- Affiche les messages d'erreur -->
      @if (error) {
        <div class="alert alert-danger">
          {{ error }}
        </div>
      }
    </div>
  `
})
export class UserListComponent implements OnInit {
  users: User[] = [];
  error: string | null = null;

  ngOnInit(): void {
    this.loadUsers();
  }

  loadUsers(): void {
    this.userService.getUsers().subscribe({
      next: (users) => {
        this.users = users;
      },
      error: (err) => {
        this.error = err.message;
      }
    });
  }
}
```

```

    </div>
  }

  <!-- Affiche un message de chargement -->
  @if (loading) {
    <div class="text-center">
      Chargement...
    </div>
  }

  <!-- Affiche la table des utilisateurs -->
  @if (users.length > 0) {
    <table class="table table-striped">
      <thead>
        <tr>
          <th>Email</th>
          <th>Rôles</th>
        </tr>
      </thead>
      <tbody>
        @for (user of users; track user.id) {
          <tr>
            <td>{{ user.email }}</td>
            <td>{{ user.roles.join(', ') }}</td>
          </tr>
        }
      </tbody>
    </table>
  } @else {
    <div class="text-center">
      Aucun utilisateur trouvé
    </div>
  }
</div>
`,
  styles: [`
    .container { padding: 20px; max-width: 800px; margin: 0 auto; }
    .table { margin-top: 20px; }
    .alert { padding: 10px; border-radius: 4px; margin-bottom: 20px; }
    .alert-danger { background-color: #f8d7da; border-color: #f5c6cb; color:
#721c24; }
  `]
})
export class UserListComponent implements OnInit {
  users: User[] = []; // Liste des utilisateurs typée

```



```

loading = true;      // Indique si les données sont en cours de chargement
error = '';          // Message d'erreur à afficher en cas de problème

constructor(private userService: UserService) {}

ngOnInit(): void {
  this.loadUsers(); // Appelle la fonction de chargement des utilisateurs
}

loadUsers() {
  this.loading = true;
  this.error = '';

  this.userService.getUsers().subscribe({
    next: (response) => {
      this.users = response.member; // Stocke la liste des utilisateurs
      this.loading = false;
    },
    error: (err) => {
      this.error = 'Erreur lors du chargement des utilisateurs';
      this.loading = false;
      console.error('Erreur:', err);
    }
  });
}
}

```

1.6.4.4 Fichier src/app/users/users.component.ts à utiliser

```

import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClientModule } from '@angular/common/http';
import { UserService } from '../services/user.service';
import { User, ApiResponse } from '../models/user.model'; // Import des interfaces

@Component({
  selector: 'app-users', // Sélecteur du composant (utilisé dans le HTML parent)
  standalone: true,
  imports: [CommonModule, HttpClientModule], // Import des modules nécessaires
  templateUrl: './users.component.html', // Fichier HTML associé
  styleUrls: ['./users.component.css'] // Fichier CSS associé
})

```

```

    // Suppression de 'providers' car UserService est déjà injecté globalement
  })
  export class UsersComponent implements OnInit {
    users: User[] = []; // Tableau typé pour stocker les utilisateurs
    error: string = ''; // Message d'erreur en cas de problème
    loading: boolean = true; // Indicateur de chargement

    constructor(private userService: UserService) {}

    ngOnInit(): void {
      this.loadUsers(); // Charge les utilisateurs au démarrage du composant
    }

    loadUsers(): void {
      this.loading = true; // Active l'indicateur de chargement

      this.userService.getUsers().subscribe({
        next: (response: ApiResponse) => { // Typage de la réponse API
          this.users = response.member; // On stocke les utilisateurs dans le
          tableau
          this.loading = false; // Désactive l'indicateur de
          chargement
        },
        error: (err) => { // Gestion des erreurs
          console.error('Erreur lors du chargement des utilisateurs :', err);
          this.error = 'Impossible de charger les utilisateurs.';
          this.loading = false; // Désactive l'indicateur de
          chargement même en cas d'erreur
        }
      });
    }
  }
}

```

1.6.5 Créer le dossier « guards » et configurer auth.guards.ts

Un **Guard** est un mécanisme qui permet de **protéger les routes** dans une application Angular.

Il vérifie certaines conditions avant de permettre à l'utilisateur d'accéder à une page.

Ici, le `auth.guard.ts`

- **Vérifie si l'utilisateur est connecté** grâce à `AuthService.isAuthenticated()`.
- **Autorise ou bloque l'accès** à la route protégée.

- **Redirige vers la page de login** si l'utilisateur n'est pas authentifié.

Se placer dans « front » et exécuter la commande :

ng generate guard guards/auth

Choisir l'option CanActivate :

Which type of guard would you like to create?

>Ⓐ CanActivate

☐ CanActivateChild

☐ CanDeactivate

☐ CanMatch

1.6.5.1 Fichier `src/app/guards/auth.guard.ts` à utiliser

```
// Créer un guard pour protéger les routes
// src/app/guards/auth.guard.ts
// Importation des modules nécessaires d'Angular
import { inject } from '@angular/core'; // Permet d'injecter des
services dans une fonction
import { Router, type CanActivateFn } from '@angular/router'; // Importation du
type CanActivateFn et de Router pour gérer la navigation
import { AuthService } from '../services/auth.service'; // Importation du
service AuthService qui gère l'authentification

// Définition du guard sous forme de fonction, de type CanActivateFn
export const authGuard: CanActivateFn = (route, state) => {
  // Injection du service AuthService pour vérifier si l'utilisateur est
  authentifié
  const authService = inject(AuthService);

  // Injection du service Router pour rediriger l'utilisateur si nécessaire
  const router = inject(Router);

  // Vérifie si l'utilisateur est authentifié en utilisant la méthode
  isAuthenticated() du service AuthService
  if (authService.isAuthenticated()) {
```

```
    return true; // Si l'utilisateur est authentifié, on lui permet d'accéder à
la route
}

// Si l'utilisateur n'est pas authentifié, on le redirige vers la page de
connexion
return router.createUrlTree(['/login']);
};
```

1.6.6 Configuration du routing

Modifier src/app/app.routes.ts

1.6.6.1 Fichier src/app/app.routes.ts à utiliser :

```
// Mettre à jour les routes
// src/app/app.routes.ts
// Importation des modules nécessaires d'Angular
import { Routes } from '@angular/router'; // Importation du type Routes qui
permet de définir les chemins de l'application
import { LoginComponent } from '../login/login.component'; // Importation du
composant LoginComponent pour la page de connexion
import { WelcomeComponent } from '../welcome/welcome.component'; // Importation du
composant WelcomeComponent pour la page d'accueil après connexion
import { authGuard } from '../guards/auth.guard'; // Importation du guard
authGuard qui protège les routes nécessitant une authentification

// Définition des routes de l'application
export const routes: Routes = [
  // Route par défaut : redirige vers la page de login
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  // Route pour la page de connexion, le composant LoginComponent sera affiché
  { path: 'login', component: LoginComponent },
  // Route protégée : nécessite l'authentification, le composant WelcomeComponent
sera affiché uniquement si l'utilisateur est authentifié
  { path: 'welcome', component: WelcomeComponent, canActivate: [authGuard] },
];
```

1.6.7 Configuration du composant principal

Modifier src/app/app.component.ts

1.6.7.1 Fichier src/app/app.component.ts à utiliser

```
// src/app/app.component.ts
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet],
  template: `
    <div class="app-container">
      <h1>Gestion des Utilisateurs</h1>
      <router-outlet></router-outlet>
    </div>
  `,
  styles: [`
    .app-container {
      padding: 20px;
    }
    h1 {
      text-align: center;
      margin-bottom: 30px;
    }
  `]
})
export class AppComponent {}
```

1.6.8 Modifier le fichier app.component.html comme suit

```
<h1>Bienvenue sur l'application des utilisateurs</h1>
<app-user-list></app-user-list> <!-- Appel du composant -->
```

1.6.9 Modifier app.config.ts comme suit :

```
// Importation des modules nécessaires d'Angular
import { ApplicationConfig } from '@angular/core'; // Type pour la configuration
de l'application Angular
import { provideRouter } from '@angular/router'; // Fournisseur pour
configurer le routeur Angular
import { routes } from './app.routes'; // Import des routes
définies dans le fichier 'app.routes.ts'
import { provideHttpClient } from '@angular/common/http'; // Fournisseur pour la
configuration du client HTTP Angular

// Définition de la configuration de l'application Angular
export const appConfig: ApplicationConfig = {
  providers: [
    // Fournisseur pour configurer le routeur avec les routes spécifiées dans
    'app.routes.ts'
    provideRouter(routes),

    // Fournisseur pour configurer le client HTTP Angular
    provideHttpClient()
  ]
};
```

1.7 Test de l'application

1.7.1 1. Démarrer le backend

```
cd back
symfony server:start
```

1.7.2 2. Démarrer le frontend

```
cd front
ng serve
```

1.7.3 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`

2 INTERACTIONS ENTRE SYMFONY ET ANGULAR

2.1 Flux d'authentification

2.1.1 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
```

```
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
```

2.2 Flux de récupération des utilisateurs

2.2.1 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
```



```
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
```

2.3 Points d'interaction clés

2.3.1 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
```

2.3.2 Intercepteur HTTP (Angular)

// Ajout automatique du token JWT à chaque requête

Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1...

2.4 Sécurité

2.4.1 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]
}
```

2.5 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

2.6 État de l'application

2.6.1 Côté Angular

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

2.6.2 Côté Symfony

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