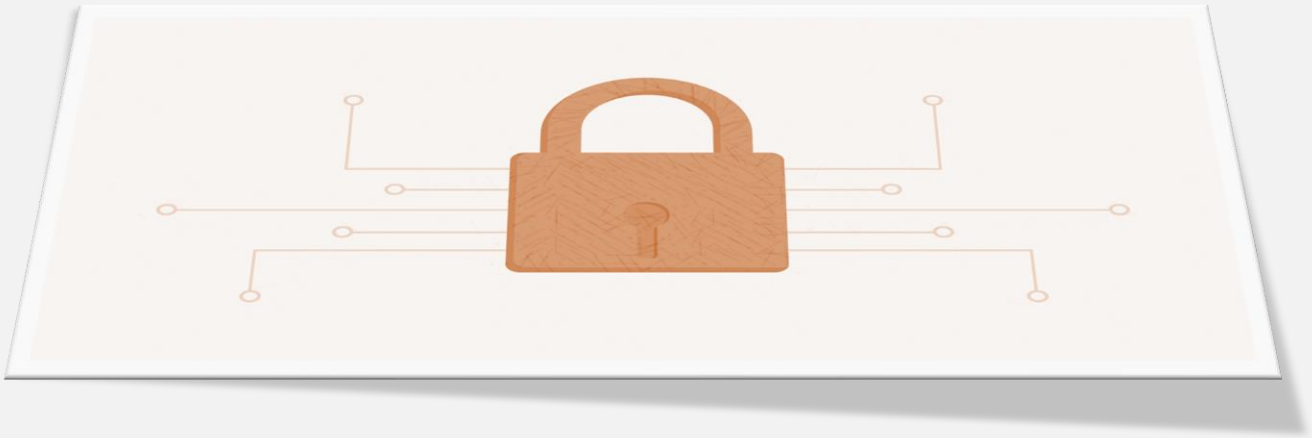

Documentation technique : Guide d’authentification

20 AOUT 2023

MEHDI HADDOU | DAPS – P8

GUIDE D'AUTHENTIFICATION

I. INTRODUCTION

Cette documentation a pour but de guider les futurs développeurs juniors à comprendre comment l'authentification a été implémentée dans le projet Symfony 5.4. Elle explique quel fichier a été modifié, comment l'authentification fonctionne, où sont stockées les informations des utilisateurs et comment assurer une collaboration harmonieuse et de maintenir la qualité du code.

II. FICHIER MODIFIÉ

Pour mettre en place l'authentification, le fichier *config/packages/security.yaml* a été modifié. Ce fichier contient la configuration de sécurité de l'application, où les paramètres relatifs à l'authentification et aux rôles d'accès sont définis.

III. PROCESSUS D'AUTHENTIFICATION

Le processus d'authentification dans l'application Symfony est géré par le composant de sécurité intégré. Voici comment il est configuré dans le fichier *config/packages/security.yaml* :

```yaml
# config/packages/security.yaml
security:
    enable_authenticator_manager: true
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: username
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider
            custom_authenticator: App\Security\SecurityAuthenticator
            logout:
                path: logout
                target: homepage

    access_control:
        - { path: ^/login, roles: PUBLIC_ACCESS }
        - { path: ^/users, roles: ROLE_ADMIN }
        - { path: ^/coverage, roles: PUBLIC_ACCESS }
        - { path: ^/, roles: IS_AUTHENTICATED_FULLY }
```

Explications :

```
enable_authenticator_manager: true
```

« enable_authenticator_manager » indique à Symfony d'utiliser le gestionnaire d'authentification.

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

« password_hashers » configure le hachage des mots de passe.

```
providers:
    app_user_provider:
        entity:
            class: App\Entity\User
            property: username
```

« providers » spécifie le fournisseur d'utilisateurs pour l'authentification. Dans ce cas, « app_user_provider » utilise l'entité « User » comme classe d'utilisateur et la propriété « username » pour l'identifiant de connexion.

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        lazy: true
        provider: app_user_provider
        custom_authenticator: App\Security\SecurityAuthenticator
        logout:
            path: logout
            target: homepage
```

« firewalls » définit les pare-feux de sécurité pour différents chemins de l'application. Dans cet exemple « main » est le pare-feu principal utilisé pour l'authentification :

- « lazy » indique que l'authentification ne se produit qu'en cas de besoin, améliorant les performances ;
- « provider » utilise le fournisseur d'utilisateurs « app_user_provider » défini précédemment dans notre configuration ;
- « custom_authenticator » spécifie la classe personnalisée « SecurityAuthenticator » qui gère le processus d'authentification. Cette classe permet d'effectuer les vérifications nécessaires et authentifier les utilisateurs.
- « logout » configure le chemin de déconnexion et la redirection après déconnexion vers la route définit dans « target ».

```yaml
access_control:
    - { path: ^/login, roles: PUBLIC_ACCESS }
    - { path: ^/users, roles: ROLE_ADMIN }
    - { path: ^/coverage, roles: PUBLIC_ACCESS }
    - { path: ^/, roles: IS_AUTHENTICATED_FULLY }
```

« access_control » définit les règles d'accès en fonction des rôles des utilisateurs pour différents chemins de l'application. Il est crucial de noter que l'ordre des règles dans « access_control » est important. Les règles sont évaluées dans l'ordre où elles sont définies, et la première règle correspondante est appliquée. C'est pourquoi il est essentiel de définir les règles les plus spécifiques en premier, suivi des règles plus générales. Si une règle plus générale est définie avant une règle spécifique, elle pourrait annuler les restrictions souhaitées. Cette configuration permet de contrôler l'accès aux différentes parties de l'application en fonction des rôles des utilisateurs et de garantir une sécurité adéquate.

IV. STOCKAGE DES UTILISATEURS

Les informations des utilisateurs sont stockées en base de données et sont gérées par le système ORM (Object-Relational Mapping) de Symfony, Doctrine. Les utilisateurs sont représentés par la classe « App\Entity\User », qui est une entité définie dans le projet. Cette classe définit la structure et les propriétés des utilisateurs, telles que le nom d'utilisateur, le mot de passe haché et les rôles.

Doctrine prend en charge la création, la mise à jour et la récupération des données des utilisateurs à partir de la base de données. Grâce à la configuration du fournisseur d'utilisateurs dans le fichier « security.yaml », Symfony peut interagir avec Doctrine pour authentifier les utilisateurs lors du processus d'authentification.

En résumé, les utilisateurs sont stockés en base de données et gérés par Doctrine, et la classe « App\Entity\User » représente la structure des utilisateurs dans le projet Symfony.

V. CONCLUSION

Cette documentation technique a couvert les aspects essentiels de l'implémentation de l'authentification dans l'application Symfony 5.4. Nous avons exploré les différentes étapes du processus d'authentification, en mettant en évidence les fichiers et configurations clés nécessaires à son bon fonctionnement.

De plus, nous avons abordé le stockage des utilisateurs dans la base de données à l'aide de l'entité « App\Entity\User ». Cette entité est gérée par Doctrine et fournit une structure pour stocker les informations relatives aux utilisateurs, y compris leurs identifiants et rôles.

Nous avons également présenté l'importance de suivre les règles de sécurité, en particulier l'ordre des règles d'accès dans « access_control », pour garantir un accès approprié aux différentes parties de l'application en fonction des rôles des utilisateurs.

Il est important de noter que cette documentation est destinée à faciliter la compréhension et la prise en main du processus d'authentification pour les développeurs juniors rejoignant l'équipe. Elle fournit des explications détaillées sur les étapes clés, les configurations et les éléments impliqués dans le flux d'authentification. Elle leur permettra de naviguer avec confiance à travers les configurations, les classes et les concepts clés liées à l'authentification, tout en garantissant la sécurité et l'efficacité de l'application.