**Authentification Guide**

La sécurité concernant l'authentification est configuré dans le fichier config/packages/security.yaml Vous trouverez plus d'informations concernant ce fichier et ses différentes parties dans la documentation officiel : [Security (Symfony 4.4 Docs)](https://symfony.com/doc/4.4/security.html).

**L'entité User**

Avant toute de chose, il est nécessaire d'avoir défini une entité qui représentera l'utilisateur connecté. Cette classe doit implémenter l'interface UserInterface et donc implémenter les différentes méthodes définis dans celle-ci. Dans ce cas-ci, cette classe a déjà été implémentée et se situe dans la fichier src/Entity/User.php.

**Les Providers**

Un provider va nous permettre d'indiquer où se situe les informations que l'on souhaite utiliser pour authentifier l'utilisateur, dans ce cas-ci, on indique qu'on récupérera les utilisateurs via Doctrine grâce à l'entité User dont la propriété username sera utilisé pour s'authentifier sur le site. Attention, on peut indiquer ici la classe User car celle-ci implémente l'interface UserInterface !

# config/packages/security.yaml

providers:

users\_in\_memory: { memory: null }

users\_in\_database:

entity:

class: App\Entity\User

property: username

**Les encoders**

Un encoder va simplement nous permettre de déterminer quel est l'algorithme que l'on souhaite utiliser lors de l'encodage d'une certaine informations dans une certaine entité. Dans ce cas-ci on utilisera l'algorithme bcrypt lorsque que quelque chose doit être encoder dans l'entité App\Entity\User via UserPasswordEncoderInterface, dans ce cas-ci cela concerne le mot de passe.

# config/packages/security.yaml

encoders:

App\Entity\User:

Algorithm: bcrypt

**Les Firewalls**

Un firewall va définir comment nos utilisateurs vont être authentifiés sur certaines parties du site. Le firewall dev ne concerne que le développement ainsi que le profiler et ne devra à priori pas être modifié. Le firewall main englobe l'entièreté du site à partir de la racine défini via pattern: ^/, l'accès y est autorisé en anonyme c-à-d sans être authentifié, on y indique que c'est le provider "doctrine" qui sera utilisé. Afin de s'authentifier, on définit un formulaire de connexion via form\_login: où sont indiqués le nom des routes correspondant à ce formulaire, la route de vérification du login ainsi que la route vers laquelle l'utilisateur devra être redirigé par défaut après son authentification.

# config/packages/security.yaml

firewalls:

dev:

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

security: false

main:

anonymous: true

pattern: ^/

provider: users\_in\_database

form\_login:

login\_path: login

check\_path: login\_check

always\_use\_default\_target\_path: true

default\_target\_path: /

logout: ~

**Les Access\_Control**

Un access\_control va définir les limitations d'accès à certaines parties du site. Dans ce cas-ci, on indique que :

* L'url /login est accessible sans authentification.
* L'url /users n'est accessible qu'en étant authentifié avec un utilisateur ayant le rôle "ROLE\_ADMIN".
* Tout le reste du site n'est accessible qu'aux utilisateurs authentifiés c-à-d ayant le rôle "ROLE\_USER".

# config/packages/security.yaml

access\_control:

- { path: ^/login, roles: IS\_AUTHENTICATED\_ANONYMOUSLY }

- { path: ^/users, roles: ROLE\_ADMIN }

- { path: ^/, roles: ROLE\_USER }

**Les Role\_Hierarchy**

Un role\_hierarchy permet de s'assurer qu'un utilisateur ayant un certain rôle aura automatiquement d'autres rôles. Dans ce cas-ci, un utilisateur possédant le rôle "ROLE\_ADMIN" aura automatiquement le rôle "ROLE\_USER".

# config/packages/security.yaml

role\_hierarchy:

ROLE\_ADMIN: ROLE\_USER