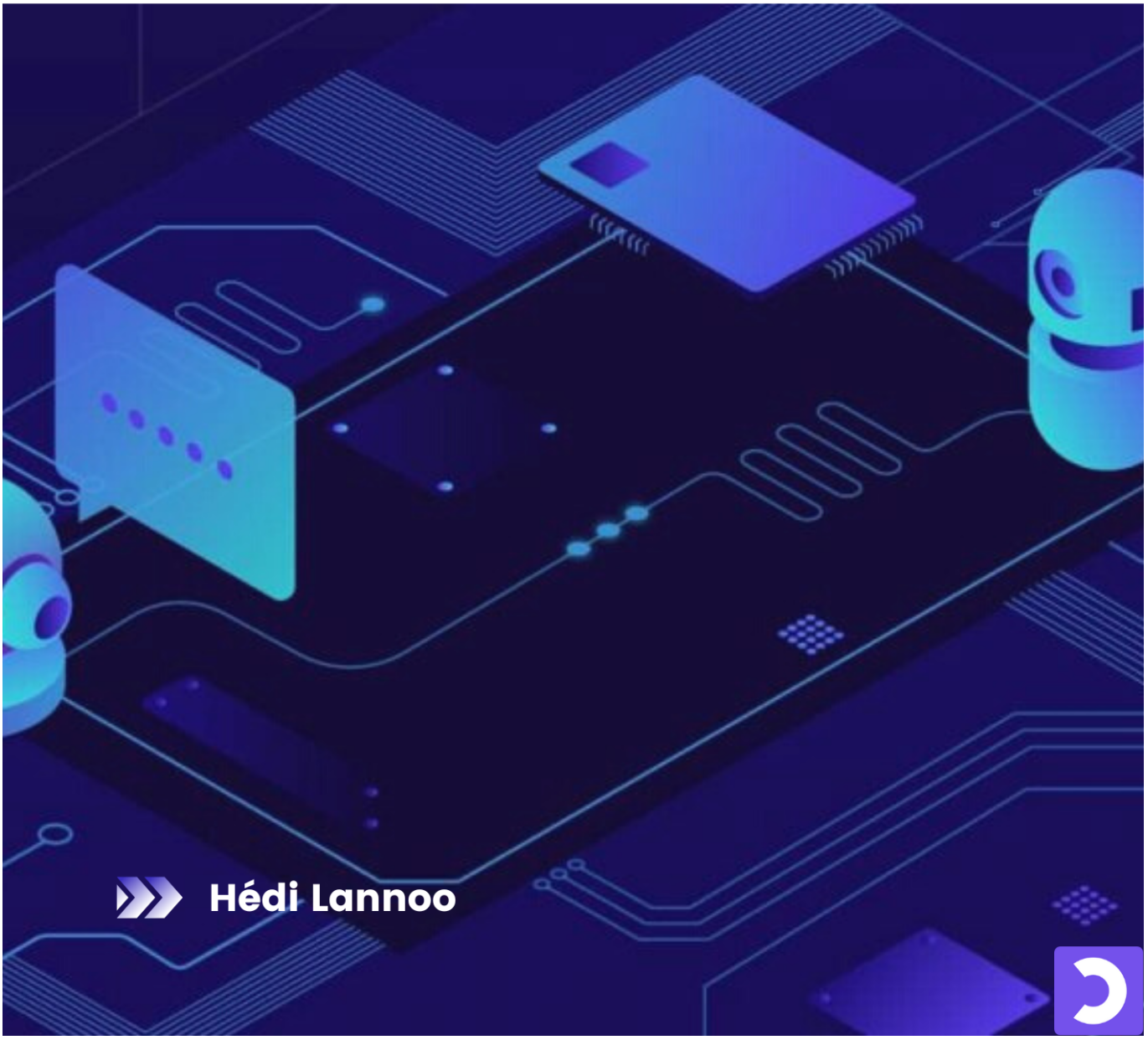
2022

DOCUMENTATION TECHNIQUE : AUTHENTIFICATION



Hédi Lannoo

SOMMAIRE

L'authentification **3**

L'utilisateur **3**

1.1 - L'entité 3

1.2 - Mise en place 4

1.3 - Gestion de la sécurité 5

Le processus d'authentification **5**

2.1 - Mise en place 5

2.2 - Le paramétrage 6

Custom_authenticator 6

User_checker 7

Access_denied_handler 8

Lazy 8

L’authentification

L'authentification est un processus permettant un système informatique en l'occurrence notre application de s'assurer de l’identité de l’utilisateur, de la légitimité de la demande d'accès faite par une entité en l'occurrence ici un potentiel utilisateur afin d'autoriser son accès ou non à des ressources du système, conformément à un paramétrage défini.

Ainsi dans cette définition nous pouvons relever 2 éléments primordiaux :

- L’utilisateur
- Le processus d’authentification

1) L’utilisateur

1.1 - L’entité

Il est définit dans notre application par une entité nommé “User” , ce ‘User” est une classe également présente en base de données, possédant différents attributs, tels que :

- id
- username
- password
- email
- rôles
- etc..

Ces attributs nous permettent de l’identifier et ainsi de lui accorder ou non certains droits et/ou de lui appliquer certaines restrictions.
Il possède également certaines méthodes nous permettant d'interagir avec lui.

1.2 - Mise en place

Afin de faciliter sa création nous avons installé le maker-bundle de symfony permettant de réaliser certaines actions directement dans la console via :

- *composer require symfony/maker-bundle*

Désormais nous allons créer un utilisateur via le maker avec la commande :

- php/bin console make:user

Ainsi différentes questions vont nous être posées (en anglais) :

Le nom de classe que nous voulons créer comme entité lié à la sécurité de l’application
    - User

Souhaitons nous enregistrer l’entité ‘User’ en base de données (via doctrine)
    - Oui

Entrer le champ permettant d'identifier l'utilisateur, c'est avec celui-ci qu'il se connectera en plus de son mot de passe :

- email

Voulons nous crypter les mots de passe avant de les enregistrer en base de données

- Oui

Voulez vous utiliser l'algorithme de hashage par défaut :

- Oui

Vous pouvez désormais ajouter différents attributs supplémentaires via la commande :

- php bin/console make/entity

Celle-ci vous demandera l'entité à modifier, en l'occurrence il s'agit ici du User, il vous sera demandé un nom pour le nouvel attribut ajouté, son type, sa longueur et s'il peut être null en base de données.

Dans mon application, j'ai ainsi ajouté les attributs :
:

- username (string, 255, not nullable) => Pseudo utilisateur
- token (string,255, nullable) => Permet une vérification lors de l'inscription
- task (OneToMay) => Permet de le relier à des taches
- isVerified (boolean) => permet de savoir s'il s'agit d'un utilisateur vérifier

Ainsi la classe user à été crée, nous pouvons désormais la pousser sur notre base de données, celle-ci ayant au préalable été configurer dans votre fichier .env

- php bin/console make:migration => Créer un fichier de migration vers la BDD
- php bin/console doctrine:migration migrate => Effectue la migration du fichier

1.3 - Gestion de la sécurité

La création de la classe User via la commande "make:user" a généré un "providers" dans le fichier security.yaml :

```yaml
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

Ce provider nous permet de définir ou se situe les informations pour authentifier l'utilisateur, ici dans l'entité User avec comme identifiant principal l'email.
Plus globalement ce fichier nous permet de gérer les paramètres de sécurité de notre application.

2) Le processus d'authentification

2.1 - Mise en place

Ce processus permet de gérer/d'autoriser l'entrée de l' utilisateur préalablement défini dans notre application.
Pour faciliter sa création nous allons également utiliser une commande via le maker :

- *php bin/console make:auth*

La commande va alors ouvrir une boîte de dialogue nous demandant de choisir entre un authentificateur vide [0], ou un authentificateur par formulaire de login |1] :

- 1

Il nous sera alors demandé de choisir un nom pour cette nouvelle classe :

- LoginFormAuthenticator

Ainsi dans le security.yaml sera ajouté un main firewall ou pare-feu principal en français, avec l'authentificateur créé :

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        lazy: true
        provider: app_user_provider
        form_login: ~
        custom_authenticator:
            - App\Security\LoginFormAuthenticator
        user_checker: App\Security\UserChecker
        access_denied_handler: App\Security\AccessDeniedHandler
        logout:
            path: security_logout
            target: security_login
```

2.2 - Le paramétrage

A) Custom_authenticator

Le custom_authenticator appelle le LoginFormAuthenticator qui lui contient différentes méthodes :

```php
public function supports(Request $request): ?bool
{
    return $request->attributes->get( key: '_route') === 'security_login'
        && $request->isMethod( method: 'POST');
}

public function authenticate(Request $request): Passport
{
    $email = $request->get( key: 'login')['email'];

    return new Passport(
        new UserBadge($email),
        new PasswordCredentials($request->get( key: 'login')['password']),
        [
            new CsrfTokenBadge( csrfTokenId: 'authenticate', $request->get( key: '_csrf_token'))
        ]
    );
}

public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{

    return new RedirectResponse($this->urlGenerator->generate( name: 'task_list'));

}

public function onAuthenticationFailure(Request $request, AuthenticationException $exception): ?Response

{

    return $request->getSession()->set(Security::AUTHENTICATION_ERROR,$exception);

}
```

- La fonction support vérifie que nous sommes bien sur la route “security_login”

- La fonction authenticate va quant à elle vérifier que l’email, le password sont fournis et qu’il correspondent à un utilisateur. Le csrf_token permet de contrer la faille du même nom grâce à un token en vérifiant sa conformité.

- La fonction onAuthenticationSuccess, s’occupe de l’action à mener en cas de succès, ici l’utilisateur sera redirigé vers la liste des tâches.

- La fonction onAuthenticationFailure, va quant à elle lever une exception en cas d’échec.

B) User_checker

Est ajouté au firewall un **user_checker** :

```php
class UserChecker implements UserCheckerInterface
{
    public function checkPreAuth(UserInterface $user): void
    {
        if (!$user instanceof AppUser) {
            return;
        }

        if (!$user->isVerified()) {
            // the message passed to this exception is meant to be displayed to the user
            throw new CustomUserMessageAccountStatusException( message: "Votre compte utilisateur n'est pas vérifié");
        }
    }
```

Dans notre cas, celui-ci va simplement vérifier avant d'opérer l'authentification, si l'utilisateur n'est pas un utilisateur vérifié, s' il ne l'est pas il lèvera une exception et bloquera le reste de l'authentification.

C) Access_denied_handler

L' **access_denied_handler** va quant à lui se charger plus globalement de rediriger tous les accès refusés vers la page de login avec un message associé :

```php
public function handle(Request $request, AccessDeniedException $accessDeniedException): ?Response
{
    $this->addFlash( type: 'error',  message: "Vous ne disposez pas des droits requis pour réaliser cette action");

    return new RedirectResponse($this->urlGenerator->generate( name: 'security_login'));
}
```

D) Lazy

Pour le paramètre **lazy : true** il permet l'injection de services un peu lourds, en l'occurrence ici du mailer pour notre classe EmailVerifier sans avoir à l'utiliser directement.

Ainsi, il ressemble et agit comme le mailer, sauf que le mailer n'est pas réellement instancié tant que l'on interagit pas avec le proxy d'une manière ou d'une autre.