

SYMFONY

La sécurité

Anthony PARIS - Formateur

PRÉSENTATION

Le contrôle de la sécurité sous Symfony est très avancé mais également très simple.

Pour cela Symfony distingue :

l'authentification

l'autorisation

PRÉSENTATION

L'authentification est le processus qui va définir qui vous êtes, en tant que visiteur.

L'enjeu est vraiment très simple : soit vous ne vous êtes pas identifié sur le site et vous êtes un anonyme, soit vous vous êtes identifié (via le formulaire d'identification ou via un cookie « Se souvenir de moi ») et vous êtes un membre du site. C'est ce que la procédure d'authentification va déterminer. Ce qui gère l'authentification dans Symfony s'appelle un firewall, ou un pare-feu en français.

Ainsi vous pourrez sécuriser des parties de votre site Internet juste en forçant le visiteur à être un membre authentifié. Si le visiteur l'est, le firewall va le laisser passer, sinon il le redirigera sur la page d'identification. Cela se fera donc dans les paramètres du firewall

PRÉSENTATION

L'autorisation est le processus qui va déterminer si vous avez le droit d'accéder à la ressource (la page) demandée. Il agit donc après le firewall. Ce qui gère l'autorisation dans Symfony s'appelle l'access control.

Par exemple, un membre identifié lambda aura accès à la liste de sujets d'un forum, mais ne peut pas supprimer de sujet. Seuls les membres disposant des droits d'administrateur le peuvent, c'est ce que l'access control va vérifier.

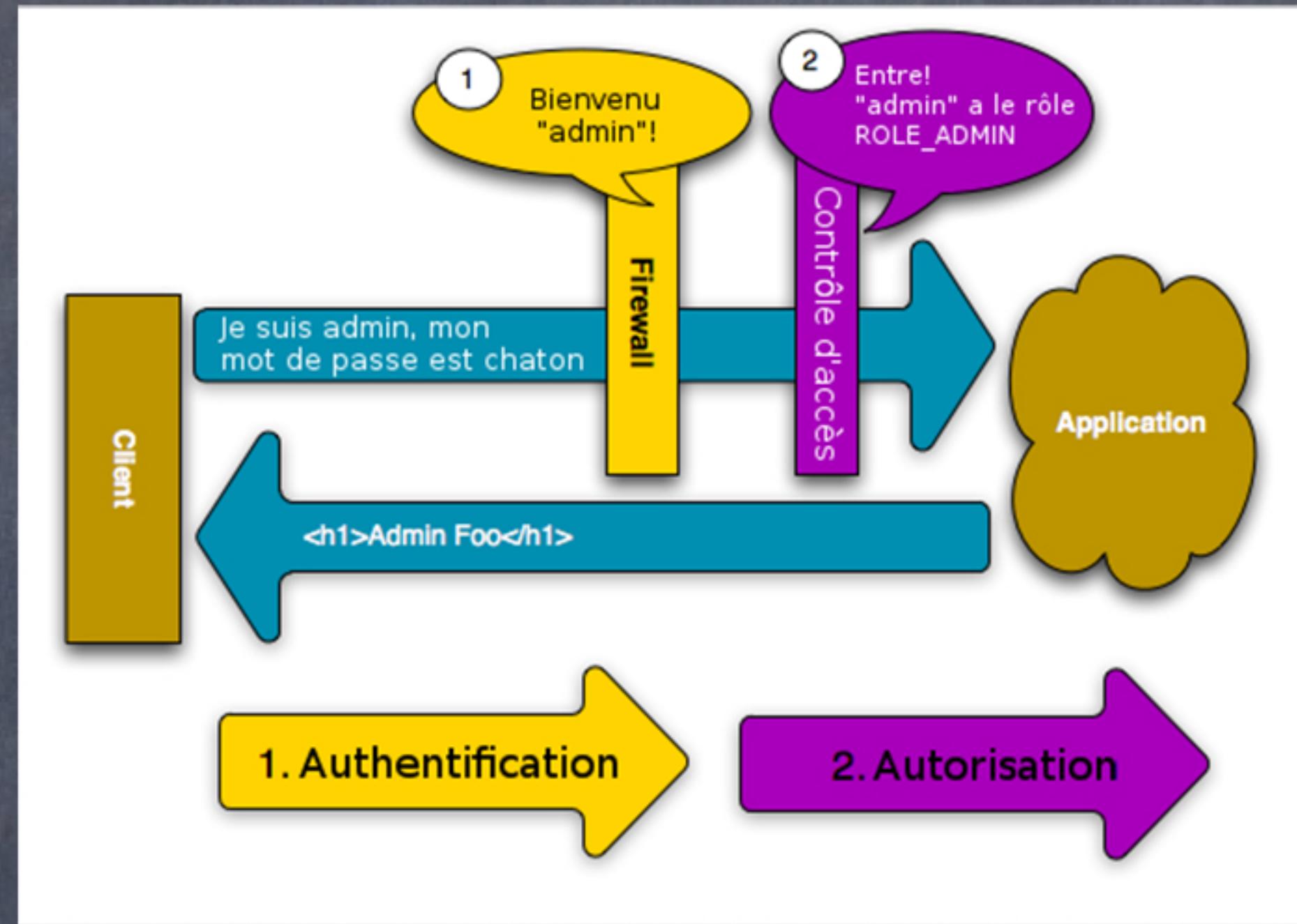
PRÉSENTATION

Lorsqu'un utilisateur tente d'accéder à une ressource protégée, le processus est le suivant :

1. Un utilisateur veut accéder à une ressource protégée ;
2. Le firewall redirige l'utilisateur au formulaire de connexion ;
3. L'utilisateur soumet ses informations d'identification (par exemple login et mot de passe) ;
4. Le firewall authentifie l'utilisateur ;
5. L'utilisateur authentifié renvoie la requête initiale ;
6. Le contrôle d'accès vérifie les droits de l'utilisateur, et autorise ou non l'accès à la ressource protégée.

PRÉSENTATION

Exemple : Je suis identifié, et je veux accéder à la page/admin/foo qui requiert des droits que j'ai.



1. L'utilisateur admin s'identifie, et il tente d'accéder à la page/admin/foo. D'abord, le firewall confirme l'authentification d'admin. Ici aussi, c'est bon, il laisse donc passer admin.
2. Le contrôle d'accès regarde si la page/admin/foorequiert des droits d'accès : oui, il faut le rôle-ROLE_ADMIN, qu'admin a bien. Il laisse donc passer l'utilisateur.
3. L'utilisateur admin a alors accès à la page/admin/foo, car il est identifié et il dispose des droits nécessaires.

PRÉSENTATION

Le processus de sécurité suit toujours ces étapes mais le nature de l'authentification peut varier. En effet, selon vos besoins, il existe de nombreuses manières d'authentifier les visiteurs, (banal formulaire d'identification, authentification avec Google, etc). Il faut retenir que la méthode choisie n'a pas d'incidence sur votre code.

Pour gérer la connexion via google, facebook, etc , il existe un package compatible SF
<https://github.com/hwi/HWIOAuthBundle> assez bien documenté.

Pour plus d'informations :

<https://symfony.com/doc/current/security.html>

MISE EN APPLICATION

Pour commencer, nous allons d'abord créer notre entité User. Les infos sur nos utilisateurs vont être enregistrées en base de données et c'est de là-bas que nous allons récupérer son login et mot de passe pour l'identifier. Pour créer l'entité User, nous allons utiliser la commande suivante:

Vous avez juste à taper «entrer» à chaque fois.

```
PS D:\wamp64\www\sf6m2icours> php bin/console make:user

The name of the security user class (e.g. User) [User]:
>

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
>

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
>

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).

Does this app need to hash/check user passwords? (yes/no) [yes]:
>

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml

Success!
```

MISE EN APPLICATION

L'entity USER a été générée automatiquement, nous remarquons qu'à la différence des autres entity elle hérite de «userInterface» définissant un pattern standard (des conditions spécifiques) à remplir pour le définir en tant que système de connexion de Symfony.

Remarquez également que le fichier config/packages/security.yaml a été mis à jour.

```
security:  
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords  
    password_hashers:  
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'  
    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider  
    providers:  
        # used to reload user from session & other features (e.g. switch_user)  
        app_user_provider:  
            entity:  
                class: App\Entity\User  
                property: email  
firewalls:  
    dev:  
        pattern: ^/(_profiler|wdt)|css|images|js/  
        security: false  
    main:  
        lazy: true  
        provider: app_user_provider
```

password_hashers : l'objet encodeur va encoder les mots de passe des utilisateurs

Voir : <https://symfony.com/doc/current/security.html>

role_hierarchy : (Pour l'instant non dispo) Défini les role disponible et la hiérarchie des roles de l'application

providers : Pour identifier les utilisateurs, le firewall demande à un provider. C'est un fournisseur d'utilisateurs. Ici ce sera l'entity User et le point d'accès, le mail (login)

firewalls : Pour identifier les utilisateurs, le firewall demande à un provider. C'est un fournisseur d'utilisateurs. Il désactive la sécurité pour certaines URL en dev pour le reste il utilise le provider «user»

acces_control : (actuellement commenté) Le contrôle d'accès vérifie que l'utilisateur a le(s) rôle(s) requis pour accéder au contenu demandé.

(deux possibilités d'utilisation) Directement dans ce fichier ou directement dans le controller en annotations.

MISE EN APPLICATION

Nous allons ajouter un attribut «username» pour disposer d'un nom utilisateur lors de l'inscription. Pour vous montrer qu'il est possible d'ajouter ses propres attributs.

```
PS D:\wamp64\www\sf6m2icours> php bin/console make:entity User

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> username

Field type (enter ? to see all types) [string]:
>

Field length [255]:
> 120

Can this field be null in the database (nullable) (yes/no) [no]:
>

updated: src/Entity/User.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>
```

MISE EN APPLICATION

Pensez à lancer une «migration» pour répercuter les changements dans le BDD

```
PS D:\wamp64\www\sf4m2icours> php bin/console make:migration  
  
Success!  
  
Next: Review the new migration "src/Migrations/Version20200524082136.php"  
Then: Run the migration with php bin/console doctrine:migrations:migrate  
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```

```
PS D:\wamp64\www\sf4m2icours> php bin/console doctrine:migrations:migrate  
  
Application Migrations  
  
WARNING! You are about to execute a database migration that could result in schema changes and data loss. Are you sure you  
Migrating up to 20200524082136 from 0  
  
++ migrating 20200524082136  
  
    -> CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL, email VARCHAR(180) NOT NULL, roles JSON NOT NULL, password VARCHAR(255) NOT NULL, UNIQUE INDEX UNIQ_8D93D649E7927C74 (email), PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE=InnoDB  
  
++ migrated (took 467.1ms, used 18M memory)  
  
-----  
  
++ finished in 488.7ms  
++ used 18M memory  
++ 1 migrations executed  
++ 1 sql queries
```

MISE EN APPLICATION

Revenons rapidement sur le «providers» : (security.yaml)

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

Property définit la propriété qui sera associée à la connexion avec le mot de passe. Il est possible de la changer ou de personnaliser la méthode de connexion avec une requête custom dans le repository «user»

https://symfony.com/doc/current/security/user_provider.html

En utilisant une méthode pourtant le nom «loadUserByUsername(\$argument)» il faudra simplement supprimer la «property» dans le providers pour activer cette méthode.

Cela permet de véritablement personnaliser sa méthode de connexion en fonction de paramètre (activation du compte, pays, role, etc).

MISE EN APPLICATION

Le «firewalls» définit deux sections : DEV pour le développement afin d'autoriser certaines ressources comme le profiler sans se soucier du fait d'être connecté ou non et la section main (pour tout le reste)

Toutes les URL réelles sont gérées par le «main» pare-feu (aucun pattern clé signifie qu'il correspond à toutes les URL). Un pare-feu peut avoir plusieurs modes d'authentification, c'est-à-dire qu'il permet plusieurs façons de poser la question "Qui êtes-vous ?".

Souvent, l'utilisateur est inconnu (c'est-à-dire qu'il n'est pas connecté) lorsqu'il visite votre site Web pour la première fois. Si vous visitez votre page d'accueil en ce moment, vous y aurez accès et vous verrez que vous visitez une page derrière le pare-feu dans la barre d'outils :

Visiter une URL sous un pare-feu ne nécessite pas nécessairement que vous soyez authentifié (par exemple, le formulaire de connexion doit être accessible ou certaines parties de votre application sont publiques).

The screenshot shows a browser window with a status bar at the bottom. The status bar includes icons for security, network speed (90 in 14.23 ms), and memory usage (n/a). A red arrow points to a message in the status bar: "Authenticated No Firewall name main". Another red arrow points from this message to a code snippet on the right.

```
firewalls:
  dev:
    pattern: ^/(_(profiler|wdt)|css|images|js)/
    security: false
  main:
    anonymous: lazy
    provider: app_user_provider
```

MISE EN APPLICATION

Dans la profiler symfony, vous pouvez visualiser que nous passons par le firewall «main»

Security

Token	Firewall	Listeners	Authenticators	Access Decision
main		Security enabled		Stateless

Configuration

Key	Value
provider	security.user.provider.concrete.app_user_provider
context	main
entry_point	(none)
user_checker	security.user_checker
access_denied_handler	(none)
access_denied_url	(none)
authenticators	(none)

INSCRIPTION D'UN UTILISATEUR

Nous allons maintenant commencer par permettre à nos utilisateurs de pouvoir s'inscrire sur notre site. Nous avons déjà notre entité User, ce que nous allons donc faire, c'est d'abord de créer un formulaire d'inscription où l'utilisateur va devoir renseigner ces informations en fonction de notre entité User, puis nous allons créer un contrôleur pour traiter le formulaire et enregistrer l'utilisateur.

Nous avons une commande spéciale qui va nous permettre de générer le formulaire d'inscription, son contrôleur et sa vue.

1. On s'assure qu'un utilisateur ne s'inscrit pas avec deux fois avec le même mail
2. On envoie pas de mail de vérification (car en local c'est assez long de paramétrier un mail test voir mailhog : <https://github.com/mailhog/MailHog>)
2. On souhaite que l'utilisateur se connecte automatiquement après l'inscription
On nous annonce qu'aucun «authenticators» n'a été trouvé, nous le créerons ensuite.

```
PS D:\wamp64\www\sf6m2icours> php bin/console make:registration-form

Creating a registration form for App\Entity\User

Do you want to add a @UniqueEntity validation annotation on your User class to make sure duplicate accounts aren't created? (yes/no) [yes]:
>

Do you want to send an email to verify the user's email address after registration? (yes/no) [yes]:
> no

Do you want to automatically authenticate the user after registration? (yes/no) [yes]:
>

! [NOTE] No Guard authenticators found - so your user won't be automatically authenticated after registering.
```

What route should the user be redirected to after registration?:

```
[0] _preview_error
[1] _wdt
[2] _profiler_home
[3] _profiler_search
[4] _profiler_search_bar
[5] _profiler_phpinfo
[6] _profiler_xdebug
[7] _profiler_search_results
[8] _profiler_open_file
[9] _profiler
[10] _profiler_router
[11] _profiler_exception
[12] _profiler_exception_css
[13] homepage
[14] article_add
[15] article_show
[16] article_edit
[17] article_remove
[18] fixadd
> 13
```

```
updated: src/Entity/User.php
created: src/Form/RegistrationFormType.php
created: src/Controller/RegistrationController.php
created: templates/registration/register.html.twig
```

INSCRIPTION D'UN UTILISATEUR

Nous pouvons accéder à l'url register grâce à notre nouveau controller «RegistrationController», La page est vide car notre vue générée n'est pas conforme au niveau des blocs (ici body et nous utilisons content)
Profitons-en pour améliorer légèrement l'affichage.

```
{% extends 'base.html.twig' %}

{% block title %}Création de compte{% endblock %}

{% block content %}

<div class="container">
    <h1>Création de compte</h1>
    {{ form_start(registrationForm) }}
        {{ form_row(registrationForm.email) }}
        {{ form_row(registrationForm.plainPassword, {
            label: 'Mot de passe'
        }) }}
        {{ form_row(registrationForm.agreeTerms) }}

        <button type="submit" class="btn">Création de compte</button>
    {{ form_end(registrationForm) }}
</div>
{% endblock %}
```

INSCRIPTION D'UN UTILISATEUR

Nous disposons d'un formulaire de création de compte prêt à l'emploi et d'un contrôleur ou nous ajouterons un message flash ainsi qu'une redirection vers l'accueil

```
#Route('/register', name: 'app_register')
public function register(Request $request, UserPasswordHasherInterface $userPasswordHasher, EntityManagerInterface $entityManager): Response
{
    $user = new User();
    $form = $this->createForm(RegistrationFormType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        // encode the plain password
        $user->setPassword(
            $userPasswordHasher->hashPassword(
                $user,
                $form->get('plainPassword')->getData()
            )
        );

        $entityManager->persist($user);
        $entityManager->flush();
        // do anything else you need here, like send an email

        $this->addFlash('info', "Votre compte a été créé avec succès");
        return $this->redirectToRoute('homepage');
    }

    return $this->render('registration/register.html.twig', [
        'registrationForm' => $form->createView(),
    ]);
}
```

INSCRIPTION D'UN UTILISATEUR

Modifions notre «`RegistrationFormType`» pour ajouter des contraintes et traductions

```
$builder
    ->add('email', EmailType::class, ['label'=>'Votre adresse mail'])
    ->add('username', TextType::class, [
        'label' => 'Votre nom d\'utilisateur',
        'constraints' => [
            new Length([
                'min' => 2,
                'minMessage' => 'Votre nom d\'utilisateur doit contenir au moins {{ limit }} caractères.',
            ]),
        ],
    ])
    ->add('agreeTerms', CheckboxType::class, [
        'label' => 'Conditions générales',
        'mapped' => false,
        'constraints' => [
            new IsTrue([
                'message' => 'Vous devez accepter les CGV',
            ]),
        ],
    ])
    ->add('plainPassword', PasswordType::class, [
        // instead of being set onto the object directly,
        // this is read and encoded in the controller
        'label' => "Votre mot de passe",
        'mapped' => false,
        'constraints' => [
            new NotBlank([
                'message' => 'Entrez un mot de passe',
            ]),
            new Length([
                'min' => 6,
                'minMessage' => 'Votre mot de passe doit contenir minimum {{ limit }} caractères',
                // max length allowed by Symfony for security reasons
                'max' => 4096,
            ]),
        ],
    ]),
];
```

Ce fichier définit le formulaire à afficher, ici les trois champs du formulaire sont `email`, `agreeTerms` et `plainPassword`.

Nous rajoutons `username` avec une contrainte sur la longueur (autre moyen que par l'annotation sur l'`entity`) (on devra modifier le formulaire pour afficher ce champs)

Sur le `plainPassword`, il y a un attribut «`mapped`» à `false` ce champ ne sera pas directement rattaché à l'objet (`User`) la valeur de ce champ sera récupérée dans le contrôleur pour l'encoder puis l'attribuer à l'utilisateur avec `$user->setPassword($encoded)`.
Pareil pour «`agreeTerms`» ce champ ne sera pas récupéré dans l'`entité` (non présent en base)

Les contraintes ici sont exactement la même chose que ce que nous avions déjà réalisé dans les annotations.

INSCRIPTION D'UN UTILISATEUR

Nous modifions notre vue relative au formulaire «register.html.twig» pour prendre en compte le «username»

```
{% extends 'base.html.twig' %}

{% block title %}Création de votre compte{% endblock %}

{% block content %}


<h1 class="text-center">Création de votre compte</h1>
    <hr>

    {{ form_start(registrationForm) }}
        {{ form_row(registrationForm.email) }}
        {{ form_row(registrationForm.username) }}
        {{ form_row(registrationForm.plainPassword) }}
        {{ form_row(registrationForm.agreeTerms, {"label_attr": {"class": "checkbox-custom"}})}}

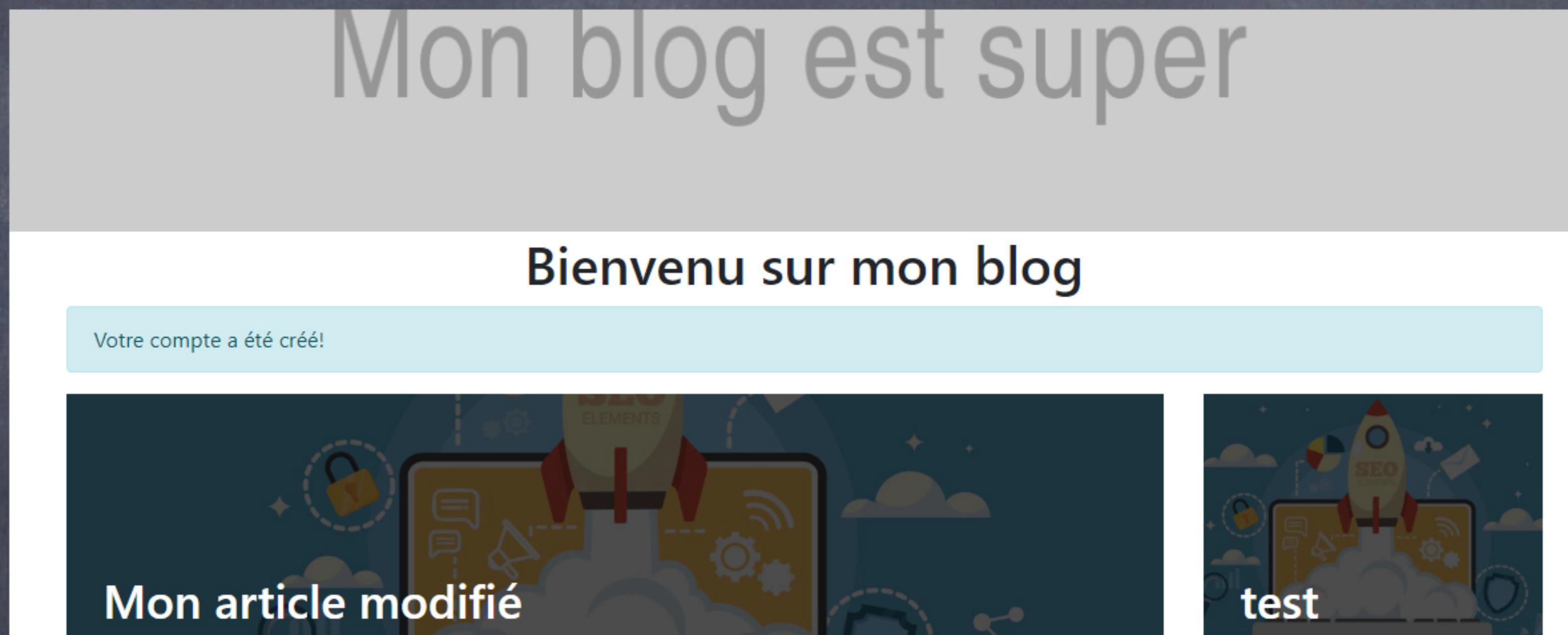
        <button type="submit" class="btn btn-primary">Création de compte</button>
    {{ form_end(registrationForm) }}


{% endblock %}
```

The screenshot shows a user registration form on a white background. At the top, there is a label 'Votre adresse mail' followed by an input field containing 'testrefffffsn@hotmail.fr'. Below that is a label 'Votre nom d'utilisateur' followed by an input field containing 'a', which is highlighted with a red border and has a red error message below it: 'ERREUR Votre nom d'utilisateur doit contenir au moins 2 caractères.' Further down, there is a label 'Votre mot de passe' followed by an empty input field. At the bottom left, there is a checked checkbox labeled 'Conditions générales'. At the bottom right, there is a blue button labeled 'Création de compte'.

INSCRIPTION D'UN UTILISATEUR

Je peux maintenant ajouter un nouvelle utilisateur, je serai redirigé vers la homepage avec un message flash.



En base je peux voir :

id	email	roles	password	username
1	test@test.fr		\$2y\$13\$SZle.K6cZORb526rOfM1F.HAwc7CwJFFbwHhiB/.9g...	Anthony

INSCRIPTION D'UN UTILISATEUR

Si vous souhaitez modifier le message d'erreur de l'adresse mail en doublon, rendez-vous dans l'entité User

```
You, il y a 1 seconde | 1 author (You)
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[UniqueEntity(fields: ['email'], message: 'Compte utilisateur déjà créé avec ce mail')]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
```

Documentation :

<https://symfony.com/doc/current/reference/constraints/UniqueEntity.html>

Vous souhaitez traduire votre projet en plusieurs langues :

<https://symfony.com/doc/current/translation.html>

CONNEXION UTILISATEUR

Pour créer le formulaire de connexion, nous allons utiliser la commande make:auth

```
PS D:\wamp64\www\sf6m2icours> php bin/console make:auth
```

```
What style of authentication do you want? [Empty authenticator]:  
[0] Empty authenticator  
[1] Login form authenticator  
> 1
```

```
The class name of the authenticator to create (e.g. AppCustomAuthenticator):  
> LoginFormAuthenticator
```

```
Choose a name for the controller class (e.g. SecurityController) [SecurityController]:  
>
```

```
Do you want to generate a '/logout' URL? (yes/no) [yes]:  
>
```

```
created: src/Security/LoginFormAuthenticator.php  
updated: config/packages/security.yaml  
created: src/Controller/SecurityController.php  
created: templates/security/login.html.twig
```

Success!

1. Quel type d'authenticator vous voulez créer?
Choisissez Login form authenticator. Pour disposer d'une base

2. Saisir le nom de la classe qui va servir d'authenticator ici LoginFormAuthenticator

3. Le nom du contrôleur rattaché à la connexion, par défaut il vous propose SecurityController c'est parfait.

4. Enfin, est-ce que nous voulons d'une route pour la déconnexion? Dans notre cas c'est intéressant

CONNEXION UTILISATEUR

Cette commande va générer 3 nouveaux fichiers:

- LoginFormAuthenticator.php dans src/Security/ ce fichier permet de récupérer l'utilisateur dans la base données, en fonction de l'email qu'il a renseigné, vérifier que le mot de passe est conforme à celui enregistré, ...
- SecurityController.php dans src/Controller/, c'est dans ce fichier que les méthodes pour la connexion login() et la déconnexion logout() sont définies.
- login.html.twig dans templates/security/ la vue qui affiche le formulaire de connexion.

La commande a aussi modifié le fichier security.yaml qui se trouve dans config/packages/ et y a rajouté l'authenticator ainsi qu'une entrée pour le logout.

CONNEXION UTILISATEUR

Nous allons modifier notre fichier security/login.html.twig afin de le rendre compatible avec notre template

```
{% extends 'base.html.twig' %}

{% block title %}Connexion! | {{parent()}}{% endblock %}

{% block content %}
<div class="container mb-5">
    <form method="post">
        {% if error %}
            <div class="alert alert-danger mt-5">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
        {% endif %}

        {% if app.user %}
            <div class="mb-3">
                Vous êtes connecté : {{ app.user.username }}, <a href="{{ path('app_logout') }}>Déconnexion</a>
            </div>
        {% endif %}

        <h1 class="text-center">Formulaire de connexion</h1>
        <hr>
        <div class="form-group">
            <label for="inputEmail">Email</label>
            <input type="email" value="{{ last_username }}" name="email" id="inputEmail" class="form-control" required autofocus>
        </div>
        <div class="form-group">
            <label for="inputPassword">Password</label>
            <input type="password" name="password" id="inputPassword" class="form-control" required>
        </div>
        <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}>
        <button class="btn btn-lg btn-primary" type="submit">Connexion</button>
    </form>
</div>
{% endblock %}
```

CONNEXION UTILISATEUR

Notez que ce formulaire n'est pas généré en PHP car il reste complexe et pour vous montrer qu'il est toujours possible de travailler avec du formulaire standard.
En me rendant sur l'url login (via votre menu)

Formulaire de connexion

Email

Password

CONNEXION UTILISATEUR

Quand l'utilisateur se connecte avec succès, il faut que nous le redirigions vers la page d'accueil, pour cela, nous allons ouvrir le fichier LoginFormAuthenticator.php qui se trouve dans src/Security/ et modifier la fonction onAuthenticationSuccess() comme ceci:

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    // For example:
    return new RedirectResponse($this->urlGenerator->generate('homepage'));
}
```

CONNEXION UTILISATEUR

Nous pouvons maintenant nous connecter, nous allons modifier notre menu afin de prendre en compte la connexion, création et déconnexion

BlogController.php > menu()

Nous allons modifier le menu en ajoutant les bonnes routing ainsi qu'un attribut user permettant de savoir si on souhaite l'affichage de la page en étant connecté ou non

```
public function menu()
{
    $listMenu = array(
        array('title' => 'Mon blog', 'texte'=>'Accueil', 'url'=>$this->generateUrl('homepage',[])),
        array('title' => 'Connexion', 'texte'=>'Connexion', 'url'=>$this->generateUrl('app_login',[]), 'user'=>false),
        array('title' => 'Création de compte', 'texte'=>'Création de compte', 'url'=>$this->generateUrl('app_register',[]), 'user'=>false),
        array('title' => 'Déconnexion', 'texte'=>'Déconnexion', 'url'=>$this->generateUrl('app_logout',[]), 'user'=>true)
    );

    return $this->render("parts/menu.html.twig", array(
        'listMenu' => $listMenu
    ));
}
```

CONNEXION UTILISATEUR

Nous allons modifier la vue «menu.html.twig»

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
<a class="navbar-brand" href="/">Mon Site Internet</a>
<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent">
    <span class="navbar-toggler-icon"></span>
</button>

<div class="collapse navbar-collapse " id="navbarSupportedContent">
    <ul class="navbar-nav ml-auto mr-5">
        {% for menu in listMenu %}
            {% if app.user and (menu.user is not defined or menu.user is same as (true)) %}
                <li class="nav-item {% if loop.index == 1 %} active{% endif %}">
                    <a class="nav-link" href="{{menu.url}}" title="{{menu.title}}">{{menu.texte}}</a>
                </li>
            {% elseif app.user is null and (menu.user is not defined or menu.user is same as (false)) %}
                <li class="nav-item {% if loop.index == 1 %} active{% endif %}">
                    <a class="nav-link" href="{{menu.url}}" title="{{menu.title}}">{{menu.texte}}</a>
                </li>
            {% endif %}
        {% endfor %}
    </ul>
</div>
</nav>
```

Pour récupérer l'utilisateur connecté dans une vue twig il suffit de faire {{ app.user }}, et vous pouvez accéder à tous ses attributs comme son email avec {{ app.user.email }}.

Si vous êtes dans un contrôleur qui hérite de la classe AbstractController de Symfony, pour récupérer l'utilisateur connecté vous faites \$this->getUser(). Là aussi vous pouvez avoir son adresse email avec \$this->getUser()->getEmail().

Vous pouvez maintenant vous connecter, déconnecter en voyant un menu différent. La déconnexion est gérée dynamiquement par symfony.

ADMINISTRATION ET RESTRICTION

Avec symfony il existe des packages prêt à l'emploi pour gérer le back office comme sonata ou easyadmin dans notre cas nous allons le concevoir nous-mêmes, afin de comprendre le fonctionnement global.

Pour faire cela nous allons :

- Créer un nouveau controller Admin
- Créer une méthode admin() dans src/Controller/AdminController.php
- Créer la vue index.html.twig dans templates/admin/ qui va afficher la liste des articles et utilisateurs

ADMINISTRATION ET RESTRICTION

Dans notre controller blog nous ajoutons une nouvelle route + méthode

```
class AdminController extends AbstractController
{
    #[Route('/admin', name: 'app_admin')]
    public function index(ManagerRegistry $doctrine): Response
    {
        $articles = $doctrine->getRepository(Article::class)->findBy(
            [],
            ['lastUpdateDate' => 'DESC']
        );

        $users = $doctrine->getRepository(User::class)->findAll();

        return $this->render('admin/index.html.twig', [
            'articles' => $articles,
            'users' => $users
        ]);
    }
}
```

Permettant de récupérer les articles par ordre de date de dernière modification ainsi que la liste des utilisateurs que nous passerons à notre nouvelle vue.

Nous allons créer cette nouvelle vue dans templates/admin/index.html.twig

ADMINISTRATION ET RESTRICTION

```
{% extends "base.html.twig" %}

{% block title %}Admin. | {{ parent() }}{% endblock %}

{% block content %}
    <div class="container">
        <div class="row">
            <div class="col ">
                <h1 class="text-center">Administration</h1>
                <hr>
            </div>
        </div>
        <div class="row">
            <div class="col">
                <div class="col ">
                    <h2 class="text-center">::Articles::</h2>
                    <a href="{{ path('article_add') }}" class="btn btn-primary mx-auto d-block">Ajouter un article ?</a>
                    <hr>
                </div>
                <div class="table-responsive">
                    <table class="table table-striped table-dark">
                        <thead>
                            <tr>
                                <th>#</th>
                                <th>Titre</th>
                                <th>Publié</th>
                                <th>Date de publication</th>
                                <th>Date de modification</th>
                                <th>Actions</th>
                            </tr>
                        </thead>
                        {% set i = 1 %}
                        {% for article in articles %}
                            <tr>
                                <td>{{ i }}</td>
```

ADMINISTRATION ET RESTRICTION

```
<td>{{ i }}</td>
<td><a href="{{ path('article_show', {'id': article.id, 'title' : article.title}) }}" target="_blank">{{ article.title }}</a></td>
<td>
    <span class="badge {{ article.isPublished ? 'bg-success' : 'bg-danger' }}">
        {{ article.isPublished ? 'oui' : 'non' }}
    </span>
</td>
<td>{{ article.isPublished ? article.publicationDate|date('d/m/Y') : '-' }}</td>
<td>{{ article.lastUpdateDate|date('d/m/Y') }}</td>
<td>
    <a class="badge bg-info" href="{{ path('article_edit', {'id': article.id}) }}>Edition</a>
    <a class="badge bg-danger" href="{{ path('article_remove', {'id': article.id}) }}>Suppression</a>
</td>
</tr>
{% set i = i+1 %}
{% endfor %}
</table>
</div>
</div>
</div>

<div class="row">
<div class="col mt-5 mb-5">
    <div class="col ">
        <h2 class="text-center">::Utilisateurs::</h2>
        <hr>
    </div>
    <div class="table-responsive">
        <table class="table table-striped table-dark">
            <thead>
                <tr>
                    <th>#</th>
                    <th>Adresse email</th>
                    <th>Username</th>
                    <th>Roles</th>
                    <th>Actions</th>
```

ADMINISTRATION ET RESTRICTION

```
<th>Roles</th>
<th>Actions</th>
</tr>
</thead>

{% set i = 1 %}
{% for user in users %}
<tr>
<td>{{ i }}</td>
<td>{{ user.email }}</td>
<td>{{ user.username }}</td>
<td>
    {% for role in user.roles %}
        <span class="badge blue">
            {{role}}
        </span>
    {% endfor %}
</td>
<td>
    <a class="badge bg-danger" href="">Suppression</a>
</td>
</tr>
{% set i = i+1 %}
{% endfor %}
</table>
</div>
</div>
</div>
</div>
{% endblock %}
```

ADMINISTRATION ET RESTRICTION

Le code est certes long mais rien de complexe à ce niveau.

Vous remarquerez le terme «role» qui permettant de définir des niveaux d'accès par défaut. Le rôle d'une personne connectée est ROLE_USER ce qui permet de faire la différence entre une personne non connectée et une personne connectée.

Il est possible de définir de nombreux rôles (modérateur, admin, superadmin, consultant,...)

test@test.fr	
Username	Authenticated
<hr/>	
Property	Value
Roles	["ROLE_USER"]
Inherited Roles	none
Token	Symfony\Component\Security\Http\Authenticator\Token\PostAuthenticationToken {#775 ▾} -user: App\Entity\User {#795 ...} -roleNames: [▶] -attributes: [] -firewallName: "main" }

ADMINISTRATION ET RESTRICTION

Votre vue admin :

Administration

::Articles::

Ajouter un article ?

#	Titre	Publié	Date de publication	Date de modification	Actions
1	Mon article modifié	oui	23/05/2020	23/05/2020	Edition Suppression
2	test	oui	23/05/2020	23/05/2020	Edition Suppression
3	test	oui	23/05/2020	23/05/2020	Edition Suppression
4	test	oui	23/05/2020	23/05/2020	Edition Suppression
5	hfhfhf	oui	23/05/2020	23/05/2020	Edition Suppression
6	hfhfhf	oui	23/05/2020	23/05/2020	Edition Suppression
7	hfhfhf	oui	23/05/2020	23/05/2020	Edition Suppression
8	rt	oui	23/05/2020	23/05/2020	Edition Suppression
9	bb	oui	23/05/2020	23/05/2020	Edition Suppression
10	test	oui	23/05/2020	23/05/2020	Edition Suppression
11	test	oui	23/05/2020	23/05/2020	Edition Suppression
12	test	oui	23/05/2020	23/05/2020	Edition Suppression
13	test	oui	23/05/2020	23/05/2020	Edition Suppression
14	test	oui	23/05/2020	23/05/2020	Edition Suppression
15	test	oui	23/05/2020	23/05/2020	Edition Suppression
16	test	oui	23/05/2020	23/05/2020	Edition Suppression
17	test	oui	23/05/2020	23/05/2020	Edition Suppression
18	test	oui	23/05/2020	23/05/2020	Edition Suppression

::Utilisateurs::

#	Adresse email	Username	Roles	Actions
1	testres@hotmail.fr	aaaa	ROLE_USER	Suppression
2	testresn@hotmail.fr	aaaa	ROLE_USER	Suppression

ADMINISTRATION ET RESTRICTION

Maintenant que nous avons notre espace d'administration, nous allons interdire l'accès à cet espace. Si vous remarquez bien à ce niveau, tout le monde peut y accéder, même si vous êtes pas connecté.

C'est grâce aux roles que nous pouvons définir une interdiction.

Un rôle est une chaîne commençant par ROLE_ exemple ROLE_ADMIN, ROLE_EDITOR, ... le tout en majuscule.

Pour attribuer un rôle à un utilisateur, nous ajoutons le rôle au tableau \$roles[] de l'objet User.

Nous allons interdire l'accès à l'espace d'administration à une personne qui n'est pas connectée et qui n'a pas le rôle ROLE_ADMIN.

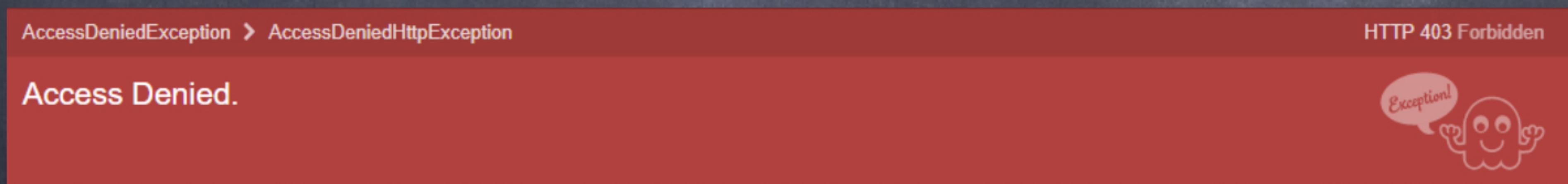
Pour cela, nous pouvons soit définir un contrôle d'accès dans le fichier config/packages/security.yaml ou directement dans le contrôleur

ADMINISTRATION ET RESTRICTION

La méthode la plus utilisée reste via le fichier config/packages/security.yaml
En bas du fichier vous trouverez un attribut `acces_control` il n'y a qu'à décommenter la ligne. Vous pouvez rajouter autant de lignes que vous le souhaitez
Si vous possédez des routes telles que /admin/XXXX elles seront également protégées

```
access_control:  
    - { path: ^/admin, roles: ROLE_ADMIN }  
    # - { path: ^/profile, roles: ROLE_USER }
```

A partir de ce moment, vous ne pouvez plus accéder à la page d'administration car vous ne possédez pas le rôle admin.



ADMINISTRATION ET RESTRICTION

Nous allons créer une commande Symfony pour nous permettre d'ajouter un rôle à n'importe quel utilisateur en renseignant son adresse email et le rôle que nous voulons lui rajouter.

Nous avons besoin au préalable de rajouter une méthode `addRoles()` de l'entity `User.php` qui se situe dans `src/Entity`, cette fonction va prendre en paramètre un rôle et l'ajouter au tableau `$roles[]` de l'utilisateur.

Vous pouvez ajouter cette méthode en dessous de «`setRoles`» pour plus de visibilité.

```
public function addRoles(string $roles): self
{
    if (!in_array($roles, $this->roles)) {
        $this->roles[] = $roles;
    }

    return $this;
}
```

ADMINISTRATION ET RESTRICTION

Nous allons créer une commande symfony pour cela lancez la commande :

```
PS D:\wamp64\www\sf4m2icours> php bin/console make:command  
Choose a command name (e.g. app:fierce-gnome):  
> app:user:role  
created: src/Command/UserRoleCommand.php  
  
Success!
```

Voici une documentation relative au custom commande dans symfony :
<https://symfony.com/doc/current/console.html>

Ici dans notre cas nous allons fournir deux arguments (Le mail et le rôle)
Nous allons vérifier le mail donc récupération de l'entityManager avec L'autowiring
nous le récupérerons en tant qu'argument du constructeur ensuite avec une requête,
nous vérifierons si le mail est bien présent dans notre base, si cela est le cas nous
ajouterons le rôle sinon nous renverrons une erreur.

A noter ici : pas de persist car le user est récupéré de la base (doctrine en a déjà
connaissance à ce moment)

ADMINISTRATION ET RESTRICTION

Dans src/Commande/UserRoleCommand.php

```
class UserRoleCommand extends Command
{
    protected static $defaultName = 'app:user:role';

    private $em;

    public function __construct(EntityManagerInterface $em)
    {
        $this->em = $em;
        parent::__construct();
    }

    protected function configure()
    {
        $this
            ->setDescription('Ajouter un rôle à un utilisateur')
            ->addArgument('email', InputArgument::REQUIRED, 'Adresse e-mail de votre utilisateur')
            ->addArgument('roles', InputArgument::REQUIRED, 'Le rôle défini.')
    }

    protected function execute(InputInterface $input, OutputInterface $output): int
    {
        $io = new SymfonyStyle($input, $output);
        $email = $input->getArgument('email');
        $roles = $input->getArgument('roles');

        $userRepository = $this->em->getRepository(User::class);
        $user = $userRepository->findOneByEmail($email);

        if ($user) [
            $user->addRoles($roles);
            $this->em->flush();

            $io->success('Le rôle a été ajouté à votre utilisateur');
        ] else {
            $io->error('Aucun utilisateur associé à ce mail.');
        }
    }

    return 0;
}
```

ADMINISTRATION ET RESTRICTION

Pour tester cette commande :

```
PS D:\wamp64\www\sf4m2icours> php bin/console app:user:role anthon@gmail.com ROLE_ADMIN
```

[ERROR] Aucun utilisateur associé à ce mail.

```
PS D:\wamp64\www\sf4m2icours> php bin/console app:user:role anthonyparis59@gmail.com ROLE_ADMIN
```

[OK] Le rôle a été ajouté à votre utilisateur

```
PS D:\wamp64\www\sf4m2icours>
```

A partir de maintenant vous pouvez visualiser la page admin qui ne sera pas accessible aux personnes non connectés ainsi qu'au utilisateur connecté n'ayant pas le rôle admin.

Pensez à vous déco-reco.

En cas de problème visualisez dans votre BDD le rôle ajouté.

```
iprimer 4 anthonyparis59@gmail.com ["ROLE_ADMIN"] $argon2id$v=19$m=65536,t=4,p=1$WHRnb1VwQ1VXcU1lWWh... anthony
```

ADMINISTRATION ET RESTRICTION

Comme vu dans security.yaml nous pouvons contrôler l'accès à nos pages, il est possible de le faire directement dans le contrôleur sous deux formes.
Voyons cela dans BlogController > add

```
#[Route(path: '/ajouter', name: 'article_add')]
public function add(Request $request, ManagerRegistry $doctrine)
{
    //VERIFICATION DU ROLE
    //SI NON ADMIN ERREUR
    $this->denyAccessUnlessGranted('ROLE_ADMIN');

    $article = new Article();
    $form = $this->createForm(ArticleType::class, $article);
```

Si vous n'avez pas le bon rôle vous serez redirigé vers /login
Si vous rencontrez une erreur de non définition de «article» dans la vue, ajoutez ceci :
(parts/form_article.html.twig) [Il est possible qu'une erreur se déclenche]

```
<h2 class="text-center">::IMAGE::</h2>
{% if article is defined and article.image.chemin is not null %}


PIXEL ONLINE CREATION


```

ADMINISTRATION ET RESTRICTION

L'autre manière est de le définir comme attribut rajoutons cela à la méthode edit()
Pensez à ajouter l'importation de la classe.

```
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

Ensuite, il suffit de rajouter l'annotation «`isGranted(ROLE_ADMIN)`»

```
#[Route(path: '/edition/{id}', name: 'article_edit', requirements: ['id' => '\d{1,8}'])]
#[IsGranted('ROLE_ADMIN')]
public function edit(Article $article, Request $request, ManagerRegistry $doctrine)
{
    $oldImage = $article->getImage()->getChemin();
    $form = $this->createForm(ArticleType::class, $article);
    // $form['image']['chemin']->setData("ok");
```

ADMINISTRATION ET RESTRICTION

Nous pouvons ajouter l'administration dans notre vue en testant le rôle (assez semblable à l'annotation dans le contrôleur)

```
<div class="collapse navbar-collapse " id="navbarSupportedContent">
  <ul class="navbar-nav ml-auto mr-5">
    {% for menu in listMenu %}
      {% if app.user and (menu.user is not defined or menu.user is same as (true)) %}
        <li class="nav-item {% if loop.index == 1 %} active{% endif %}">
          <a class="nav-link" href="{{menu.url}}" title="{{menu.title}}">{{menu.texte}}</a>
        </li>
      {% elseif app.user is null and (menu.user is not defined or menu.user is same as (false)) %}
        <li class="nav-item {% if loop.index == 1 %} active{% endif %}">
          <a class="nav-link" href="{{menu.url}}" title="{{menu.title}}">{{menu.texte}}</a>
        </li>
      {% endif %}
    {% endfor %}
    {% if is_granted('ROLE_ADMIN') %}
      <li><a class="nav-link badge badge-danger" href="{{path('admin')}}>Administration</a></li>
    {% endif %}
  </ul>
</div>
nav> You, 14 days ago • ok
```

ADMINISTRATION ET RESTRICTION

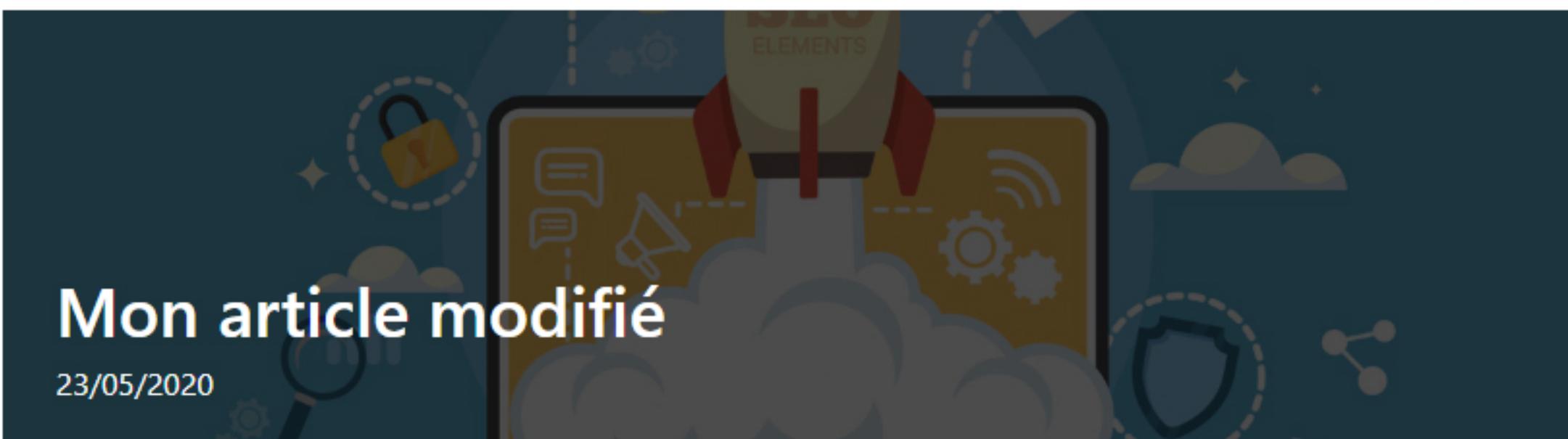
Si vous êtes connecté en tant qu'administrateur, vous verrez :

et

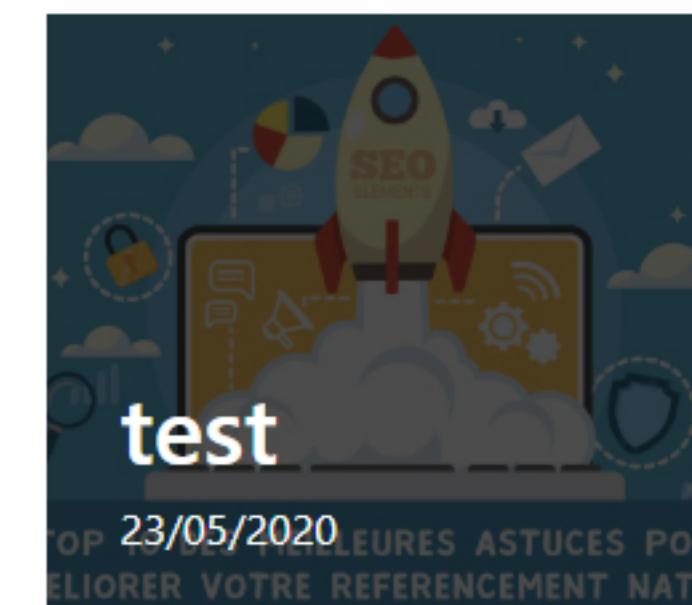
Accueil Déconnexion Administration

Mon blog est super

Bienvenu sur mon blog



Mon article modifié
23/05/2020



test
23/05/2020 MEURES ASTUCES POUR AMELIORER VOTRE REFERENCEMENT NATUREL



ADMINISTRATION ET RESTRICTION

Pour aller plus loin :

- Créer la méthode de suppression des articles (voir `remove()`)
- Créer une connexion acceptant le mail ou le username (`user_providers`)
https://symfony.com/doc/current/security/user_providers.html
- Rajouter du JS
- Créer une requête personnalisée pour afficher l'ensemble des articles avec images/catégories sur la page d'accueil

Pour rappel, les requêtes magiques 'findby' etc ne récupèrent les éléments qu'en lazy loading. Cette requête ne récupère que les articles et si vous souhaitez afficher les images, elle refera une nouvelle requête.

Il est plus intéressant de créer sa propre requête de récupération!