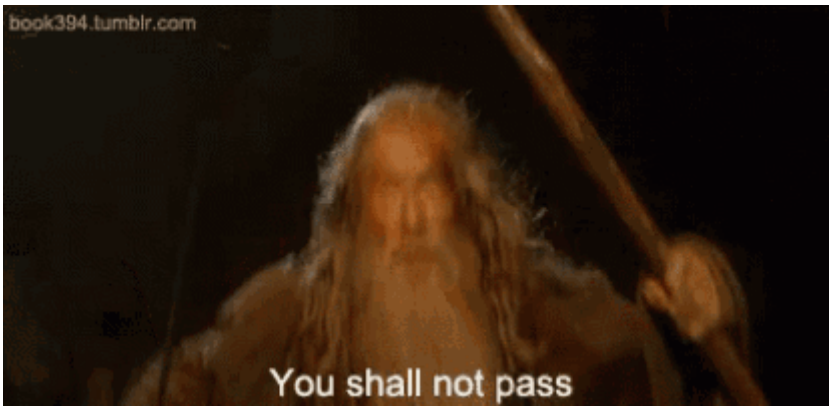
TP4: Identification d'utilisateurs

Objectif

Il est temps d'ajouter un système d'authentification à notre application afin de sécuriser des pages et actions.



(*j'espère que vous l'avez celui là*)

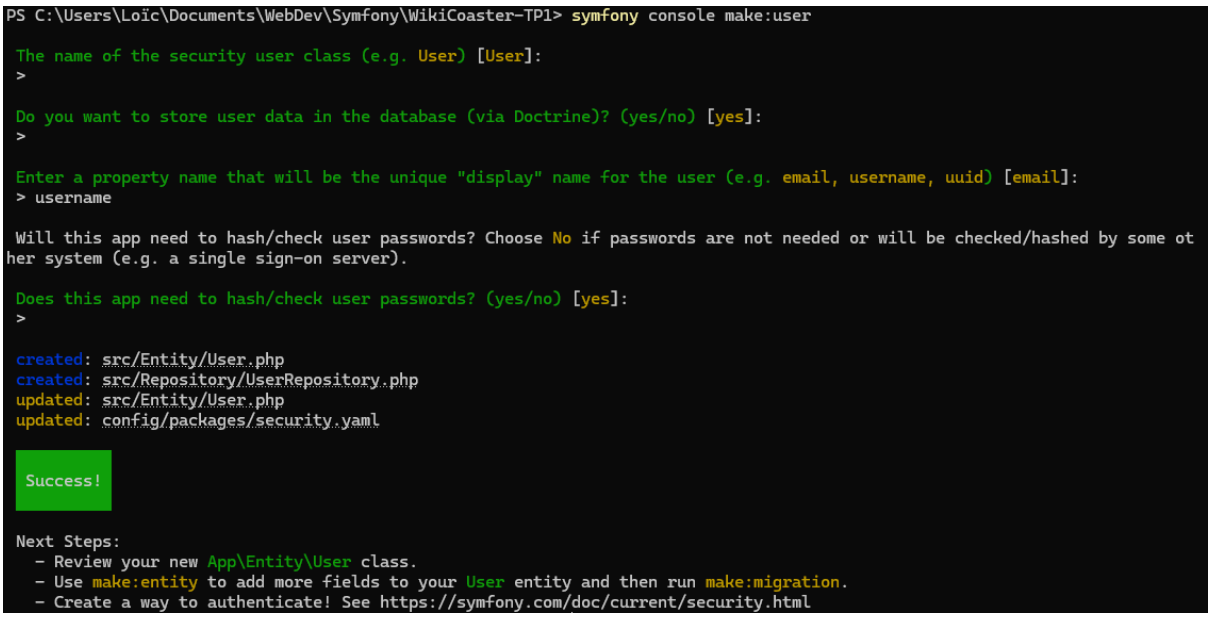
L'entité User

Ici nous n'allons pas passer par la commande de création d'entité car User va contenir des méthodes supplémentaires pour gérer les rôles et le mot de passe par exemple.

Entrez cette commande pour créer dans un premier temps une classe qui va gérer les données utilisateurs.

```
symfony console make:user
```

Suivez les étapes pour créer un utilisateur qui sera **stocké dans la base de données**. Utilisez l'**username** comme valeur à afficher.



N'oubliez pas de mettre à jour la DB.

Formulaire de création de compte

Maintenant générez le système d'inscription:

```
symfony console make:registration
```

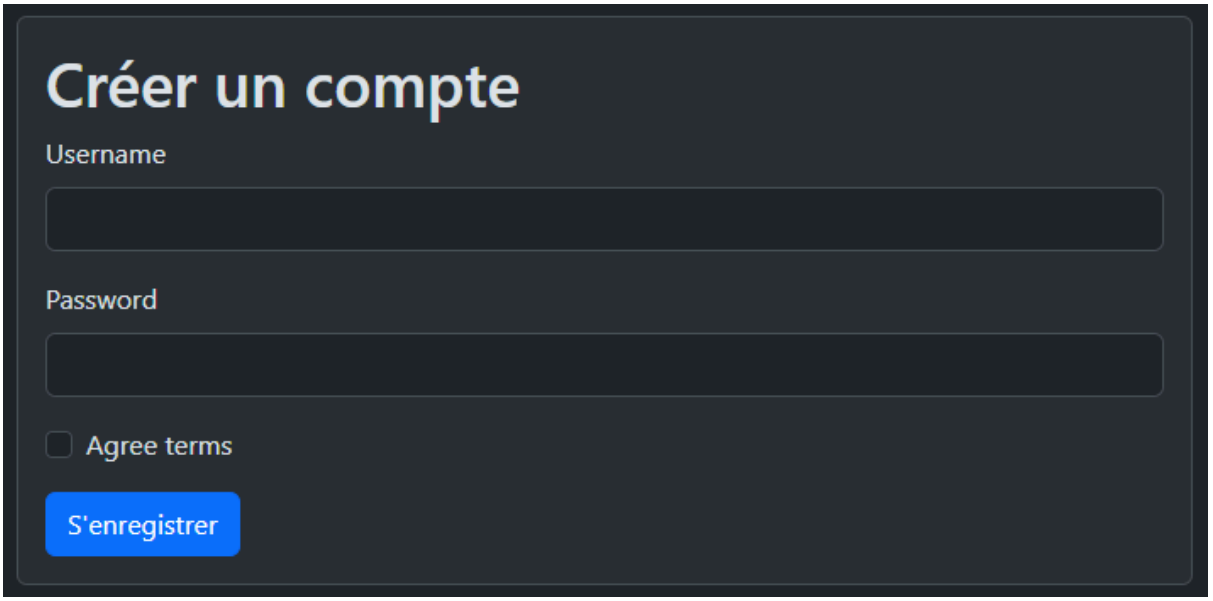
Répondez **oui** pour ajouter la contrainte "UniqueEntity", ça permet d'empêcher la création de compte avec un mail qui est déjà enregistré.

Répondez **non** pour l'envoi de mail pour vérifier l'inscription.

Choisissez enfin un nom de route pour la redirection après inscription: **app_app_index.**

La commande a créé ou modifié 4 fichiers:

- L'entité User
- Un formulaire d'inscription (ResgistrationType.php)
- Un contrôleur
- Une vue affichant le formulaire (que vous pouvez modifier)



Champ Email

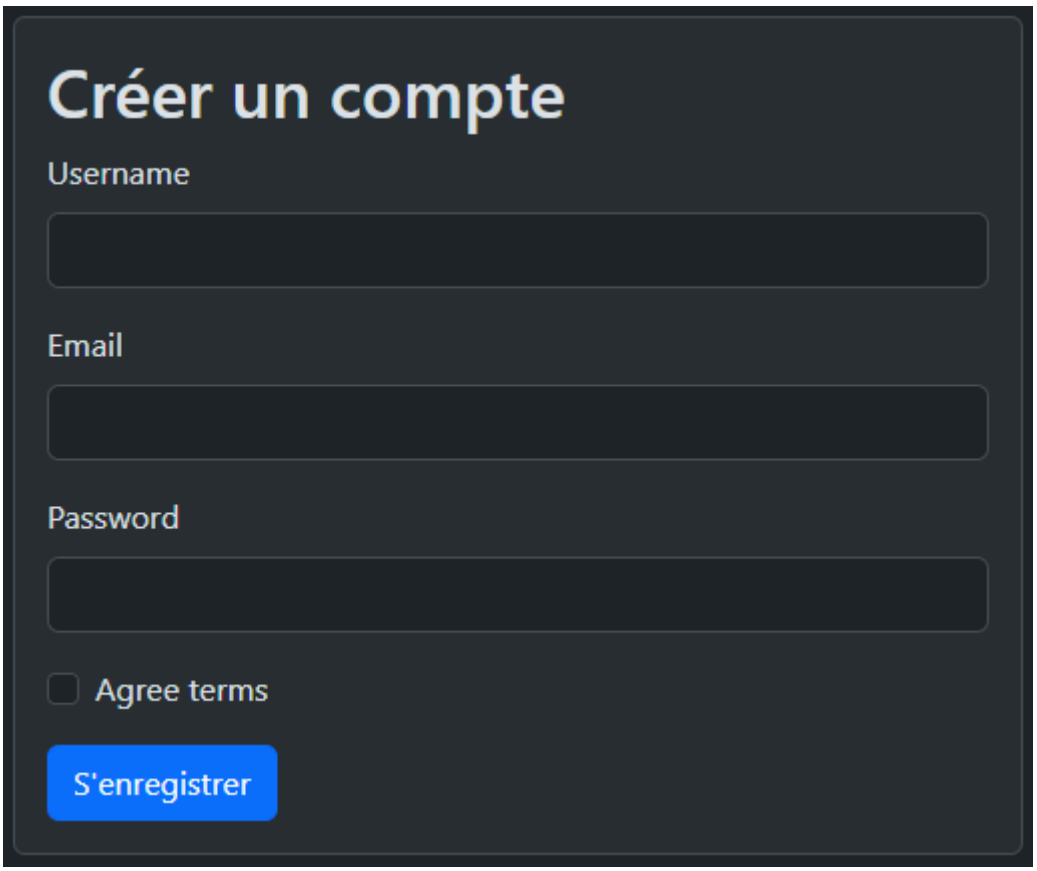
Ajoutez un champ email à l'entité User:

name	type	nullable
email	string(255)	not null

Comme ce champ ne doit pas être 'null', vous devez ajouter le champ correspondant dans le formulaire d'inscription.

Tester l'inscription

Testez maintenant le formulaire et vérifiez que l'utilisateur est bien enregistré dans la DB.



⚔️ Défi: ajoutez une nouvelle contrainte dans l'entité pour afficher une erreur s'il existe déjà un compte avec cette adresse email.

Formulaire de login

Il reste à faire le formulaire d'authentification:

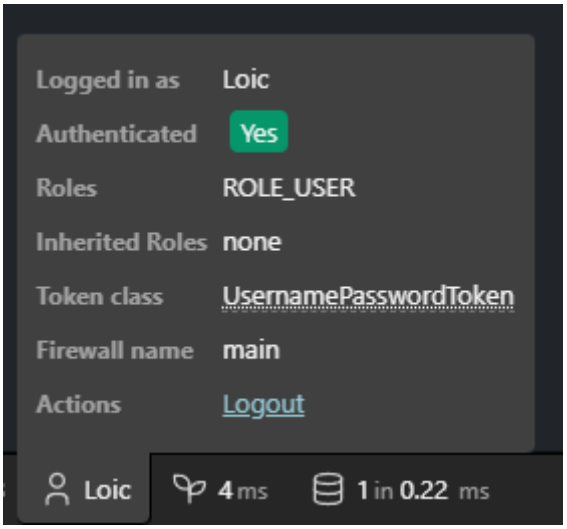
```
symfony console make:security:form-login
```

```
PS C:\Users\Loïc\Documents\WebDev\Symfony\WikiCoaster-TP1> symfony console make:security:form-login

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

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

Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]:
>

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

Success!

Next: Review and adapt the login template: security/login.html.twig to suit your needs.
```

La commande a ajouté ou modifié 3 fichiers:

- *SecurityController*
- Une vue pour le formulaire de login
- Modifié le fichier *security.yaml*

Testez le formulaire, une fois identifié vous devriez le voir dans la barre de débug.



CRUD User

Maintenant que vous avez l'entité User, vous pouvez maintenant générer le CRUD et ajouter le lien vers l'index dans le menu de l'application.

Sécuriser des pages

Les rôles

Dans Symfony, l'accès aux pages peut être géré via des rôles, se sont simplement des chaînes de caractères stockés dans le champ *roles* de la table *User,* les chaînes doivent être préfixé par "ROLE_" et sont en majuscule.

L'entité User possède une méthode *getRoles* qui retourne un array avec les rôles et ajoute automatiquement le rôle "*ROLE_USER*", vous pouvez donc tester si un utilisateur est authentifié grâce au test de ce rôle.

L'attribut "*isGranted*" permet de tester si l'utilisateur à le rôle utilisateur par défaut, si ce n'est pas le cas le visiteur sera automatiquement redirigé vers le formulaire d'authentification.

```
use Symfony\Component\Security\Http\Attribute\IsGranted;
```

```
#[IsGranted('ROLE_USER')]
public function add(Request $request, EntityManagerInterface $em): Response
```

Ajoutez cette condition dans la vue pour afficher ou non les boutons pour modifier ou supprimer un coaster:

```
{% if is_granted('ROLE_USER') %}
```

Donnée auteur d'une fiche coaster

Ajoutez un nouveau champ "*author*" dans l'entité *Coaster*

ManyToOne	User

Une fois la DB mise à jour, vous pouvez définir l'utilisateur connecté automatiquement en tant qu'auteur du coaster dans la page de création en utilisant la méthode suivante:

```
$user = $this->getUser();
```

Des voters pour plus de sécurité

Les *voters* sont des objets qui vont contenir une logique métier de droit d'accès.

Lorsqu'il y a un test d'accès (*isGranted*), les voters sont interrogés et (par défaut) si un *voter* refuse l'accès, l'utilisateur sera rediriger vers une erreur 403 (Forbidden)

Un utilisateur à le droit de modifier la fiche d'un coaster seulement s'il remplit l'une des conditions suivantes:

- Il est administrateur
- Il est l'auteur de la fiche

Ce test, c'est le *voter* qui va s'en charger.

```
symfony console make:voter
```

Nommez le "*CoasterVoter*", regardez le fichier créé, il contient une classe avec 2 méthodes:

Supports

Cette méthode va retourner un booléen pour déterminer si le voter est concerné par là demande:

Demande	Voter concerné ?
Modifier un coaster	OUI
Supprimer un Parc	NON

Pour y répondre, la méthode à besoin de savoir l'action (voir, modifier etc.) et le sujet, c'est à dire quel objet.

voteOnAttribute

Cette méthode est appelée si la méthode *supports* retourne *true,* elle permet de valider l'accès, c'est ici que va contenir la logique métier.

Modifiez cette méthode pour faire en sorte que seuls les utilisateurs qui ont créé la fiche du coaster peuvent la modifier.

Controller

Vous pouvez maintenant utiliser la méthode suivante dans le contrôleur, celle-ci générera une erreur "access denied" si la méthode *voteOnAttribute* du voter retourne false.

```php
$this->denyAccessUnlessGranted(CoasterVoter::EDIT, $coaster);
```

Rôle Admin

Ajoutez directement depuis l'éditeur de DB: ROLE_ADMIN

Modifier le voter

Le voter doit accepter la modification du coaster si l'utilisateur possède le rôle admin. Utilisez l'objet *AuthorisationChecker* pour tester le rôle.

```php
use Symfony\Component\Security\Core\Authorization\AuthorizationCheckerInterface;

//...
public function __construct(
  private readonly AuthorizationCheckerInterface $authorizationChecker
) {}
```

Note: le mot clé "readonly" à été introduit dans PHP8.1, il permet d'empêcher la modification de propriété après l'initialisation de l'objet.

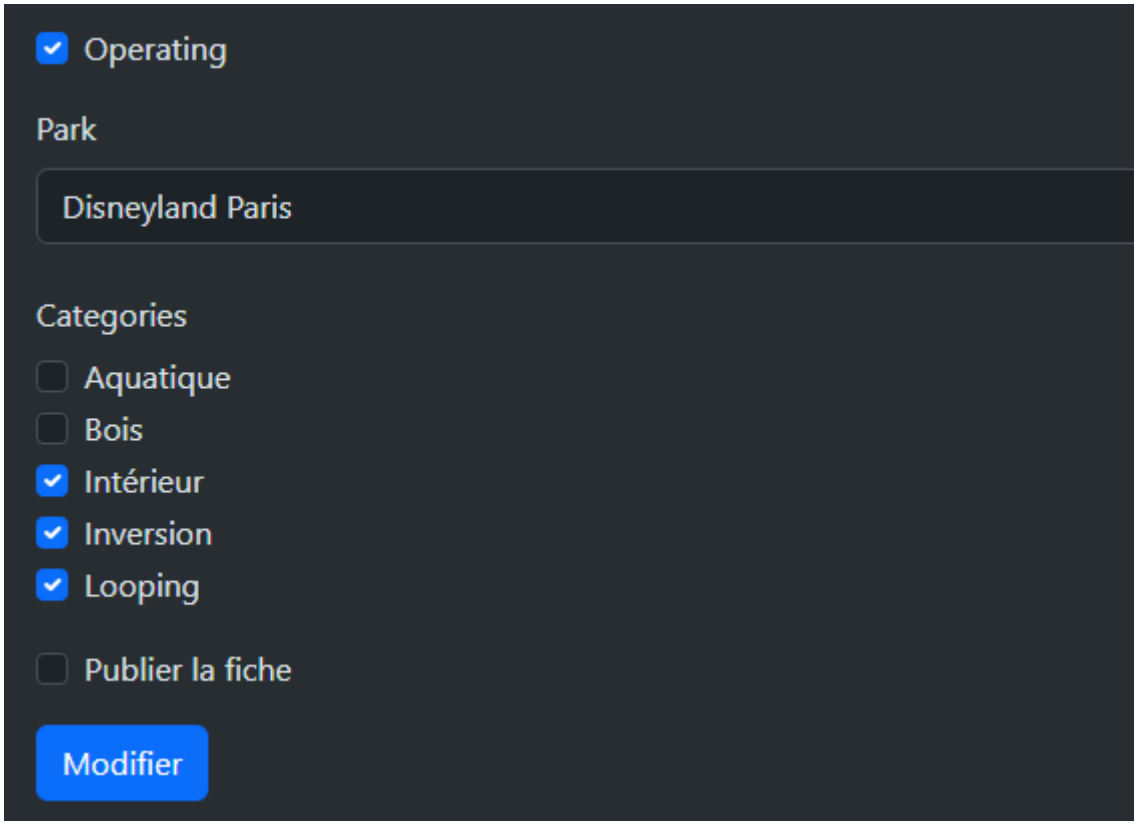
Une fois que vous avez cet objet dans la classe du voter, utilisez la méthode *isGranted* pour tester le rôle.

Publication des coasters

Pour ce dernier chapitre du TP, nous allons permettre de publier ou non une fiche d'un coaster. Par exemple, si la fiche n'est pas terminée ou contient des informations erronées, il doit être possible de ne plus la rendre visible le temps de faire les corrections.

Ajoutez une nouvelle donnée "published" dans Coaster qui contient un boolean (nullable pour le moment).

Seul un utilisateur admin peut décider de la publication ou non. Utilisez l'objet *AuthorizationCheckerInterface* avec l'injection de dépendance pour tester si l'utilisateur à bien le rôle "ROLE_ADMIN" pour afficher le champ de publication.



Il vous reste à modifier le *Voter* pour faire en sorte que seuls les admins (ou l'auteur) puissent voir un coaster non publié.

Modifier le repository

Le problème qu'il reste à régler est que les coasters qui ne sont pas publiés sont quand même retournés par la requête dans la méthode "*findFiltered*" du *CoasterRepository*.

Vous pouvez utiliser l'objet *Security* pour récupérer l'utilisateur authentifié.

```
use Symfony\Bundle\SecurityBundle\Security;

//...
```

```php
public function __construct(
    ManagerRegistry $registry,
    private readonly Security $security
){
    parent::__construct($registry, Coaster::class);
}

//...

if (!$this->security->isGranted('ROLE_ADMIN')) {
    $qb->andWhere('c.published = true OR c.author = :author')
        ->setParameter('author', $this->security->getUser())
    ;
}
```

Testez ensuite la requête: un visiteur non identifié doit voir seulement les coasters publiés dans la liste.