

La sécurité et la gestion des utilisateurs

Table des matières

| | |
|--------------------------------|-----------|
| I. Contexte | 3 |
| II. La sécurité | 3 |
| A. La sécurité | 3 |
| B. Exercice : Quiz | 6 |
| III. L'authentification | 7 |
| A. L'authentification | 7 |
| B. Exercice : Quiz | 10 |
| IV. Essentiel | 10 |
| V. Auto-évaluation | 11 |
| A. Exercice | 11 |
| B. Test | 12 |
| Solutions des exercices | 12 |

I. Contexte

Durée : 1 h

Environnement de travail :

PC ou Mac avec 8 Go de RAM minimum

IDE : VSCode ou autre

Pré requis :

Notion de PHP

Notion de POO

Contexte

La sécurité est une question importante lorsque l'on détient une application web. Il est probable que nous acceptions et que nous permettions que notre page d'accueil soit accessible à tout « *surfeur* » sans aucune restriction. Cependant, nous ne voulons pas pour autant que le site soit accessible à des personnes non connectées, voilà pourquoi il est nécessaire de créer un formulaire d'inscription et d'authentification.

De plus, même authentifié, il y a certaines pages que vous voudrez peut-être réserver à des administrateurs, à des membres de la société ou à d'autres membres. C'est pour cela qu'un système de rôles sera nécessaire.

Alors quels outils pouvons-nous utiliser dans une application Symfony 5.4 pour une bonne gestion des utilisateurs et garantir toutes les sécurités nécessaires ? C'est ce que nous verrons à travers ce cours.

II. La sécurité

A. La sécurité

Security-bundle

Afin de pouvoir nous concentrer sur la gestion des utilisateurs et la sécurité, nous allons partir d'un projet qui a déjà été créé avec une connexion à la base de données et makerbundle installée.

Commençons par installer le pack de sécurité fourni par Symfony ainsi que l'entité *User*. Grâce à ce pack de sécurité, nous pouvons faire un `make:user` à la place d'une création habituelle d'entité avec `make:entity`. De cette façon, *User* sera créé avec des outils de sécurité.

Méthode

Pour cela, dans votre terminal, tapez la ligne de commande suivante :

```
1 $ composer require symfony/security-bundle
2 $ php bin/console make:user
```

Le terminal nous demande de choisir un nom et nous propose *User*, ce qui est très bien alors validez.

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

Puis, il vous demande si vous voulez stocker les données utilisateur dans la base de données par Doctrine. Validez également.

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

Maintenant on vous demande d'entrer la propriété qui sera unique pour *User* et il vous propose par défaut *email*. Validez.

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

Autre point important, on vous demande maintenant si vous voulez *hasher* (crypter) le mot de passe d’User. Vous pouvez également valider.

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

C’est un succès, vous êtes prévenu qu’une entité et un repository User et UserRepository ont été créés et que le fichier `security.yaml` a été modifié.

Méthode

Votre entité User est assez sommaire, vous pouvez ajouter d’autres propriétés avant de faire une migration et mettre à jour la base de données. Voici les commandes qui vous seront nécessaires :

```
1 $ symfony console make:entity //choisir User et ajouter vos propriétés
2 $ symfony console make:migration
3 $ symfony console doctrine:migrations:migrate
```

Qu’est-ce que security-bundle a apporté ? Tout d’abord, dans l’entité User qui vient d’être créée, on constate que les interfaces `UserInterface` et `PasswordAuthenticatedUserInterface` ont été implémentées.

L’implémentation de ces interfaces ajoute des méthodes à notre classe User :

- `getUserIdentifier()` permet de retourner identifiant unique que l’on a choisi,
- `getRoles()` permet de récupérer le rôle de l’utilisateur,
- `setRoles()` permet de modifier le rôle de l’utilisateur,
- `getPassword()` permet de récupérer le mot de passe de l’utilisateur,
- `getsalt()` est utile uniquement si nous n’avons pas de méthode de hachage,
- `eraseCredentials()` permet d’effacer les données sensibles et temporaires.

Maintenant, regardons le fichier `security.yaml` qui se trouve dans packages du dossier config
config/packages/security.yaml.

```
1 security:
2   enable_authenticator_manager: true
3   # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
4   password_hashers:
5     Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
6   # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
7   providers:
8     # used to reload user from session & other features (e.g. switch_user)
9     app_user_provider:
10      entity:
11        class: App\Entity\User
12        property: email
13  firewalls:
14    dev:
15      pattern: ^/(_(profiler|wdt)|css|images|js)/
16      security: false
17  main:
18    lazy: true
19    provider: app_user_provider
20
21    # activate different ways to authenticate
22    # https://symfony.com/doc/current/security.html#the-firewall
23
```

```

24         # https://symfony.com/doc/current/security/impersonating_user.html
25         # switch_user: true
26
27     # Easy way to control access for large sections of your site
28     # Note: Only the *first* access control that matches will be used
29     access_control:
30         # - { path: ^/admin, roles: ROLE_ADMIN }
31         # - { path: ^/profile, roles: ROLE_USER }

```

Comme vous le constatez dans security, il y a :

- `enable_authenticator_manager: true`, il sert à activer le workflow d'authentification.
- `password_hashers`, il précise l'interface de hachage choisi et son mode, ici en « *auto* ».
- `providers`, doctrine va pouvoir connaître l'entité désignée pour la connexion ainsi que sa propriété qui désigne l'entité unique choisie.
- `firewalls`, le pare-feu gère toutes les URL et s'assure de l'authentification ; il permet d'utiliser plusieurs modes d'authentification.
- `access_control`, il permet de définir des restrictions d'accès sur des routes en fonction du rôle de l'utilisateur.

Rôles

Le rôle est une propriété importante pour gérer les utilisateurs. On peut définir les routes qui sont accessibles suivant le rôle qui est attribué à l'utilisateur dans le fichier security.yaml. Voici un exemple ci-dessous :

```

1     access_control:
2         - { path: ^/, roles: ROLE_USER }
3         - { path: ^/login, roles: PUBLIC_ACCESS }
4         - { path: ^/users, roles: ROLE_ADMIN}
5
6
7     role_hierarchy:
8         ROLE_ADMIN:       ROLE_USER

```

L'ensemble du site n'est accessible qu'aux utilisateurs connectés qui ont le rôle `ROLE_USER`. Les pages commençant par `/login` qui font pourtant partie du site sont accessibles à tous, tandis que la pages commençant par le chemin `/users` ne sont accessibles qu'aux utilisateurs qui ont le `ROLE_ADMIN`.

`role_hierarchy` ici permet de donner à `ROLE_ADMIN` toutes les autorisations de `ROLE_USER`.

Il est aussi possible de restreindre une route en mettant une annotation ou un attribut avant la méthode du contrôleur qui gère la route (par exemple : `# [IsGranted('ROLE_ADMIN')]`) ou d'utiliser les rôles comme condition dans un fichier twig.

Exemple

```

1 {% if is_granted('ROLE_ADMIN') %}
2     <a href="{{ path('user_create') }}" class="btn btn-primary">Créer un utilisateur </a>
3 {% endif %}

```

Autres outils de sécurité

Il faut savoir que Symfony apporte certains outils liés à la sécurité :

- `HttpFoundation` par exemple gère les sessions pour nous. Plus besoin, comme dans PHP, d'utiliser `session_start()` et d'autres fonctions similaires de la superglobale `$_SESSION`. Pour utiliser les sessions à partir d'un contrôleur vous pouvez taper un argument `request` de cette façon : `$session = $request->getSession();`.

- La protection CSRF peut être activée grâce à `security-csrf`. CSRF (Cross-Site Request Forgery) est un type de vulnérabilité des services d'authentification web. Ainsi `security-csrf` crée un token nommé par défaut `_csrf_token` qui permet, entre autres et pour faire simple, de vérifier qu'un formulaire vient bien de notre site et de la bonne personne. Pensez à vérifier que `csrf_protection` est à `true` dans le fichier `framework.yaml`.

```
1 framework:
2   secret: '%env(APP_SECRET)%'
3   csrf_protection: true
4   http_method_override: false
```

Pour installer ces packs, voici les commandes nécessaires :

```
1 $ composer require symfony/http-foundation
2 $ composer require symfony/security-csrf
```

B. Exercice : Quiz

[solution n°1 p.13]

Question 1

La méthode `getUserIdentifier()` permet de récupérer :

- ☐ L'email
- ☐ L'username
- ☐ Cela dépend

Question 2

Le rôle `ROLE_ADMIN` est supérieur au rôle `ROLE_USER`.

- ☐ Vrai
- ☐ Faux

Question 3

Il ne faut jamais toucher le fichier `security.yaml`.

- ☐ Vrai
- ☐ Faux

Question 4

À quoi sert le firewall qui se trouve dans le fichier `security.yaml` ?

- ☐ Il protège toutes les URL via les droits d'accès et s'assure de l'authentification
- ☐ Il coupe toutes les attaques des cybercriminels
- ☐ Aucun des deux

Question 5

Dès qu'on modifie une entité, il faut immédiatement créer une migration et la faire migrer.

- ☐ Vrai
- ☐ Faux

III. L'authentification

A. L'authentification

Méthode make:auth

Nous allons maintenant installer l'authentification. Mais d'abord, installons « twig » ainsi que « doctrine annotations » qui seront nécessaires au bon fonctionnement d'authentification. Tapez donc les commandes suivantes :

```
1 $ composer require twig
2 $ composer require doctrine/annotations
3 $ symfony console make:auth
```

```
1 What style of authentication do you want? [Empty authenticator]:
2 [0] Empty authenticator
3 [1] Login form authenticator
```

C'est la première question qu'il pose lorsqu'on installe l'authentification. Il demande si on veut qu'il génère un formulaire d'authentification. Tapez 1 pour la création de celui-ci.

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

Ensuite, il nous demande de choisir un nom pour authenticator. Choisissez ce que vous voulez, ici on a choisi UserAuthenticator.

```
1 Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
2 Do you want to generate a '/logout' URL? (yes/no) [yes]:
```

Il vous demande de choisir un contrôleur et il vous propose SecurityController.

Enfin, il demande si vous voulez générer une déconnexion. Vous devez valider.

Nous constatons qu'il a créé trois fichiers (UserAuthenticator, SecurityController et login.html.twig) et qu'il a modifié le fichier security.yaml.

À propos de security.yaml, ont été ajoutés dans le main du firewall le nom ainsi que son chemin d'accès de l'authenticator que nous avons créé, de même que le nom de la route de déconnexion.

Regardons de plus près le fichier SecurityController qui vient d'être créé :

Remarque SecurityController

```
1 #[Route(path: '/login', name: 'app_login')]
2 public function login(AuthenticationUtils $authenticationUtils): Response
3 {
4     // if ($this->getUser()) {
5     //     return $this->redirectToRoute('target_path');
6     // }
7
8     // get the login error if there is one
9     $error = $authenticationUtils->getLastAuthenticationError();
10    // last username entered by the user
11    $lastUsername = $authenticationUtils->getLastUsername();
12
13    return $this->render('security/login.html.twig', ['last_username' => $lastUsername,
14    'error' => $error]);
15 }
16 #[Route(path: '/logout', name: 'app_logout')]
17 public function logout(): void
18 {
19     throw new \LogicException('This method can be blank - it will be intercepted by the
logout key on your firewall.');
```

```
20 }
```

Dans la fonction `login()` :

- Vous pouvez décommenter la première condition. Elle redirige un utilisateur qui n'est pas connecté vers le nom de la route que vous souhaitez (modifier `target_path`).
- La variable `$error` récupère les éventuels messages d'erreur.
- La variable `$lastUsername` récupère le dernier utilisateur connecté.
- Dans le `return`, on appelle la vue, c'est-à-dire le fichier twig qu'on a choisi pour afficher notre rendu, et on lui passe les deux variables.

Dans la fonction `logout()` : À l'appel de cette route, Symfony déconnecte l'utilisateur et génère une éventuelle exception.

Remarque UserAuthenticator

`UserAuthenticator` sert à faire fonctionner toute l'authentification. Nous n'allons pas tout décrire cette classe, néanmoins vous devez modifier la méthode `onAuthenticationSuccess()`.

Pour cela, commentez la ligne d'exception qui commence par `throw`, décommenter la ligne au-dessus et mettez dans le `generate()` le nom de la route où vous souhaitez que l'utilisateur soit redirigé en cas de succès de connexion.

Remarque Login.html.twig

Maintenant regardons la page `login.html.twig` qui se trouve dans `security\` du dossier `templates\`.

Notez aussi que la page twig générée possède des classes Bootstrap qui est un framework HTML, CSS, vous bénéficierez d'une meilleure présentation si vous l'installez, mais nous ne verrons pas cette démarche ici.

```
1 {% extends 'base.html.twig' %}
2
3 {% block title %}Log in!{% endblock %}
4
5 {% block body %}
6 <form method="post">
7     {% if error %}
8         <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData,
9 'security') }}</div>
10     {% endif %}
11
12     {% if app.user %}
13         <div class="mb-3">
14             You are logged in as {{ app.user.userIdentifier }}, <a href="{{ path('app_logout') }}">Logout</a>
15         </div>
16     {% endif %}
17
18     <h1 class="h3 mb-3 font-weight-normal">Please sign in</h1>
19     <label for="inputEmail">Email</label>
20     <input type="email" value="{{ last_username }}" name="email" id="inputEmail"
21     class="form-control" autocomplete="email" required autofocus>
22     <label for="inputPassword">Password</label>
23     <input type="password" name="password" id="inputPassword" class="form-control"
24     autocomplete="current-password" required>
25     <input type="hidden" name="_csrf_token"
26         value="{{ csrf_token('authenticate') }}"
27     >
```



```

26     {#
27         ...
28     #}
29
30     <button class="btn btn-lg btn-primary" type="submit">
31         Sign in
32     </button>
33 </form>
34 {% endblock %}

```

La balise `<form method="post">` commence par 2 conditions.

La première `{% if error %}` affiche une erreur si SecurityController la détecte en la passant en paramètre à twig dans la variable `error`.

La seconde condition `{% if app.user %}` redirige vers « *logout* » si nous sommes déjà connectés.

En-dessous, nous retrouvons un formulaire HTML qui invite à entrer l'email et le mot de passe.

Méthode **make:registration-form**

Nous venons d'installer et de décrire l'authentification mais nous ne pouvons pas nous identifier car nous avons une base de données vide. Il va falloir maintenant pouvoir s'inscrire.

Dans votre terminal, tapez ces lignes de commandes (« *validator* » est nécessaire pour le formulaire qui sera créé) :

```

1 $ composer require validator
2 $ composer require form
3 $ Symfony console make:registration-form

```

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

Validez.

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

Tapez « *no* » car pour l'instant nous n'avons pas de système d'email en place.

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

Validez car on veut qu'après l'enregistrement nous soyons automatiquement connectés.

Le terminal nous explique qu'il a modifié `user` et qu'il a créé un formulaire dans `RegistrationFormType`, un contrôleur `RegistrationController` et un template `register.html.twig` :

- `RegistrationController` : ce contrôleur avec la route `/register` contient toutes les sécurités nécessaires afin de pouvoir s'enregistrer. La méthode `register()` récupère `UserPasswordHasherInterface`, `UserAuthenticatorInterface` ainsi qu'`UserAuthenticator` pour assurer cette sécurité. Afin de remplir par défaut le rôle, vous pourriez ajouter avant `entityManager.persist()` et `flush()`. `$user->setRoles(['ROLE_USER']);`.
- `RegistrationFormType` : dans ce form, il faudra ajouter avec la méthode `add()` toutes les propriétés que vous avez ajoutées à l'entité `User`, pour l'instant vous y retrouvez uniquement l'email et le mot de passe. Par défaut il a aussi créé le formulaire avec un `agreeTerms`, vous pouvez l'effacer à moins que vous ayez besoin de le paramétrer.
- `register.html.twig` : les champs du formulaire de départ sont définis dans chaque `form_row`. Supprimez le `agreeTerms` et ajoutez les champs nécessaires.

Vous avez maintenant tous les éléments pour que les utilisateurs puissent s'inscrire et s'authentifier. Pour une gestion complète des utilisateurs, vous pourrez rajouter des méthodes dans le contrôleur afin de pouvoir modifier ou supprimer un utilisateur.

B. Exercice : Quiz

[solution n°2 p.14]

Question 1

Il faut installer twig avant de faire la commande `php bin/console make:auth`.

- ☐ Vrai
- ☐ Faux

Question 2

En créant l'authentification et l'inscription avec makerbundle, avec quelle interface le mot de passe sera-t-il crypté ?

- ☐ UserAuthenticatorInterface
- ☐ \$encoder
- ☐ UserPasswordHasherInterface

Question 3

Utiliser `make:registration-form` ne crée qu'un `FormType`.

- ☐ Vrai
- ☐ Faux

Question 4

Par défaut, après une installation de l'authentification avec maker, quelle variable ne va pas récupérer le template ?

- ☐ last_username
- ☐ \$username
- ☐ errors

Question 5

Il n'y a pas vraiment besoin de savoir coder avec makerbundle car il peut mettre en place tout un système de connexion et d'inscription avec formulaire avec quelques lignes de commande !

- ☐ Vrai
- ☐ Faux

IV. Essentiel

La sécurité est un souci pour Sensiolabs, les développeurs du framework Symfony. Ce dernier permet d'installer de nombreux organes de sécurité. Citons HTTPFoundation, authenticator, security-csrf, UserPasswordHasherInterface, UserAuthenticatorInterface et bien d'autres systèmes de sécurité qui travaillent pour nous sécuriser et permettent de hasher les données sensibles.

Toutefois, la gestion des utilisateurs est aussi quelque chose de relativement aisé avec Symfony. Avec quelques lignes de commande on peut créer un utilisateur, une authentification et une inscription avec formulaire à l'appui. Le système de la gestion des rôles est également stratégiquement intéressant, afin de restreindre l'accès selon le rôle défini de l'utilisateur.

V. Auto-évaluation

A. Exercice

Vous êtes un développeur junior dans une agence web. On vous confie une application web Symfony 5.4. L'authentification et l'inscription ont été installées à l'aide de makerbundle, mais il reste encore beaucoup de choses à faire. L'entité User possède ces propriétés :

- \$id
- \$email
- \$roles
- \$password
- \$username (entité unique)
- \$address
- \$codeAddress
- \$city

Question 1

[solution n°3 p.15]

Modifiez le fichier RegistrationFormType

Modifiez le fichier RegistrationFormType ci-dessous afin qu'il soit fonctionnel et corresponde à l'entité User avec ses nouvelles propriétés.

```

1 public function buildForm(FormBuilderInterface $builder, array $options): void
2 {
3     $builder
4         ->add('email')
5         ->add('agreeTerms', CheckboxType::class, [
6             'mapped' => false,
7             'constraints' => [
8                 new IsTrue([
9                     'message' => 'You should agree to our terms.',
10                ]),
11            ],
12        ])
13        ->add('plainPassword', PasswordType::class, [
14            // instead of being set onto the object directly,
15            // this is read and encoded in the controller
16            'mapped' => false,
17            'attr' => ['autocomplete' => 'new-password'],
18            'constraints' => [
19                new NotBlank([
20                    'message' => 'Please enter a password',
21                ]),
22                new Length([
23                    'min' => 6,
24                    'minMessage' => 'Your password should be at least {{ limit }}
characters',
25                    // max length allowed by Symfony for security reasons
26                    'max' => 4096,
27                ]),
28            ],
29        ])
30    ;
31 }
```

Question 2

[solution n°4 p.15]

Maintenant, modifiez le fichier `register.html.twig` afin qu'il soit en adéquation avec le fichier `RegistrationFormType` et précisez s'il y a d'autres modifications à faire afin que la saisie du formulaire puisse être acceptée par Doctrine.

B. Test**Exercice 1 : Quiz**

[solution n°5 p.16]

Question 1

Dans l'entité `User`, l'entité unique est déclarée par métadonnée :

- ☐ Sur la propriété unique
- ☐ Sur la classe
- ☐ Sur les deux

Question 2

Vous avez de nombreuses propriétés dans votre entité `User`. `Registration-form` va adapter le formulaire à votre entité.

- ☐ Vrai
- ☐ Faux

Question 3

Vous avez de nombreuses propriétés dans votre entité `User` et `Registration-form` n'a pas adapté le formulaire suivant votre entité, que devez-vous faire ?

- ☐ Modifier `registrationFormType` et `register.html.twig`
- ☐ Modifier `registrationFormType`
- ☐ Modifier `register.html.twig`

Question 4

Il n'existe que 2 rôles différents, `ROLE_USER` et `ROLE_ADMIN`.

- ☐ Vrai
- ☐ Faux

Question 5


Si vous n'attribuez pas un rôle à la création d'un utilisateur, vous aurez un bug.

- ☐ Vrai
- ☐ Faux

Solutions des exercices


Exercice p. 6 Solution n°1**Question 1**

La méthode `getUserIdentifier()` permet de récupérer :

- ☐ L'email
- ☐ L'username
- ☒ Cela dépend
-  En effet, `getUserIdentifier()` récupère la propriété que l'on a défini comme étant l'identifiant unique.


Question 2

Le rôle `ROLE_ADMIN` est supérieur au rôle `ROLE_USER`.

- ☒ Vrai
- ☐ Faux
-  Par défaut, le rôle `ROLE_ADMIN` est supérieur au rôle `ROLE_USER`.


Question 3

Il ne faut jamais toucher le fichier `security.yaml`.

- ☐ Vrai
- ☒ Faux
-  La plupart du temps lorsqu'on installe un bundle qui a trait à la sécurité, il va lui-même modifier ce fichier via Symfony Flex, mais vous serez amené à bien le comprendre pour parfois le modifier suivant vos besoins.


Question 4

À quoi sert le firewall qui se trouve dans le fichier `security.yaml` ?

- ☒ Il protège toutes les URL via les droits d'accès et s'assure de l'authentification
- ☐ Il coupe toutes les attaques des cybercriminels
- ☐ Aucun des deux
-  Effectivement, le pare-feu ou firewalls s'occupe de tout ce qui concerne l'authentification.

Question 5


Dès qu'on modifie une entité, il faut immédiatement créer une migration et la faire migrer.

- ☒ Vrai
- ☐ Faux
-  C'est une habitude à prendre sinon doctrine ne pourra pas fonctionner correctement.

Exercice p. 10 Solution n°2


Question 1

Il faut installer twig avant de faire la commande `php bin/console make:auth`.

- ☒ Vrai
- ☐ Faux
-  C'est vrai, car makerBundle va créer un template avec un formulaire.


Question 2

En créant l'authentification et l'inscription avec makerbundle, avec quelle interface le mot de passe sera-t-il crypté ?

- ☐ UserAuthenticatorInterface
- ☐ \$encoder
- ☒ UserPasswordHasherInterface
-  UserPasswordHasherInterface est l'interface utilisée pour hasher le mot de passe dès l'inscription afin qu'il n'apparaisse jamais en clair dans la BDD.


Question 3

Utiliser `make:registration-form` ne crée qu'un FormType.

- ☐ Vrai
- ☒ Faux
-  `Registration-form` va également créer un contrôleur et utiliser authenticator pour la sécurité.


Question 4

Par défaut, après une installation de l'authentification avec maker, quelle variable ne va pas récupérer le template ?

- ☐ last_username
- ☒ \$username
- ☐ errors
-  \$username est la variable créée dans le contrôleur mais elle va s'intituler last_username dans le template.

Question 5

Il n'y a pas vraiment besoin de savoir coder avec makerbundle car il peut mettre en place tout un système de connexion et d'inscription avec formulaire avec quelques lignes de commande !

- ☐ Vrai
- ☒ Faux
-  Il est vrai que Symfony propose un système d'aide particulièrement efficace et sécurisé pour la gestion des utilisateurs mais ce n'est qu'un squelette, il faudra personnaliser en fonction de vos besoins les formulaires et continuer sur cette base pour faire un site de qualité.

p. 11 Solution n°3

Id et roles ne doivent pas être dans le formulaire.

```

1 public function buildForm(FormBuilderInterface $builder, array $options): void
2 {
3     $builder
4         ->add('email', EmailType::class, [
5             'label' => 'E-mail'
6         ])
7         ->add('plainPassword', PasswordType::class, [
8             'mapped' => false,
9             'attr' => ['autocomplete' => 'new-password'],
10            'constraints' => [
11                new NotBlank([
12                    'message' => 'Entrez votre mot de passe',
13                ]),
14                new Length([
15                    'min' => 6,
16                    'minMessage' => "Le mot de passe n'a pas {{ limit }} caractères",
17                    'max' => 4096,
18                ]),
19            ],
20            'label' => 'Mot de passe'
21        ])
22        ->add('username', TextType::class, [
23            'label' => 'Nom'
24        ])
25        ->add('address', TextType::class, [
26            'label' => 'Adresse'
27        ])
28        ->add('codeAddress', TextType::class, [
29            'label' => 'Code postal'
30        ])
31        ->add('city', TextType::class, [
32            'label' => 'Ville'
33        ])
34    ;
35 }

```

p. 12 Solution n°4

D'abord, il va falloir modifier dans l'entité user la métadonnée qui définit l'entité unique

```
#[UniqueEntity(fields: ['username'], message: 'il y a déjà un compte avec cet identifiant')].
```

Pour le fichier twig, vérifiez, si vous mettez des labels ou des contraintes, qu'ils ne sont pas déjà dans le fichier Form.

```

1 {% extends 'base.html.twig' %}
2
3 {% Bloc title %}Register{% endBloc %}
4
5 {% Bloc body %}
6     <h1>Inscription</h1>
7
8     {{ form_start(registrationForm) }}
9         {{ form_row(registrationForm.email) }}

```

```


10     {{ form_row(registrationForm.plainPassword) }}
11     {{ form_row(registrationForm.username) }}
12     {{ form_row(registrationForm.address) }}
13     {{ form_row(registrationForm.codeAddress) }}
14     {{ form_row(registrationForm.city) }}
15
16     <button type="submit" class="btn">Enregistrer</button>
17     {{ form_end(registrationForm) }}
18 {% endBloc %}

```

Exercice p. 12 Solution n°5


Question 1

Dans l'entité User, l'entité unique est déclarée par métadonnée :

- ☐ Sur la propriété unique
- ☒ Sur la classe
- ☐ Sur les deux
-  Si vous installez User à l'aide de maker, c'est lui qui ajoutera l'attribut ou l'annotation mais vous pouvez modifier le message, qui est par défaut en anglais.


Question 2

Vous avez de nombreuses propriétés dans votre entité User. Registration-form va adapter le formulaire à votre entité.

- ☐ Vrai
- ☒ Faux
-  Ce n'est pas actuellement le cas avec Symfony 5.4, il faudra rajouter les champs nécessaires au formulaire.

Question 3

Vous avez de nombreuses propriétés dans votre entité User et Registration-form n'a pas adapté le formulaire suivant votre entité, que devez-vous faire ?

- ☒ Modifier registrationFormType et register.html.twig
- ☐ Modifier registrationFormType
- ☐ Modifier register.html.twig
-  Il faut penser à ajouter vos champs avec la méthode add() dans registrationFormType ainsi que les appeler dans le fichier twig avec form_row par exemple.

Question 4

Il n'existe que 2 rôles différents, ROLE_USER et ROLE_ADMIN.

- ☐ Vrai
- ☒ Faux

Q Il en existe bien d'autres. Ces noms sont une convention, vous pouvez définir le nom que vous souhaitez mais il faudra qu'il soit bien identifié dans le fichier `security.yaml`.

Question 5

Si vous n'attribuez pas un rôle à la création d'un utilisateur, vous aurez un bug.

☐ Vrai

☒ Faux

Q Par défaut si vous suivez l'installation des utilisateurs et de sécurisation décrite précédemment, le rôle peut être null mais l'utilisateur ne pourra pas être autorisé à accéder à une bonne partie de l'application.