La sécurité et la gestion des utilisateurs dans Symfony

1. Objectifs

- Comprendre et mettre en œuvre des pratiques de sécurité avancées ainsi que des techniques efficaces de gestion des utilisateurs dans Symfony
- Configurer et personnaliser les fonctionnalités de sécurité intégrées de Symfony.
- Gérer les autorisations et les rôles des utilisateurs de manière flexible et sécurisée.

2. 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.
- Pour cela, dans votre terminal, tapez la ligne de commande suivante :**`composer require symfony/security-bundle`**

3. Créer une classe d'utilisateurs

- Nous allons créer une classe d'utilisateurs en utilisant la commande make:user. Cette commande générera une classe d'utilisateurs pour la sécurité et mettra automatiquement à jour le fichier **security.yaml**.
- Pour exécuter cette commande, ouvrez votre terminal de commande et saisissez la ligne suivante :**`symfony console make:user`**. Vous pouvez aussi utiliser une autre commande: **`php bin/console make:user`**
- Cette commande nous permet de créer une entité "**User**". Nous passons par cette commande et non par la commande de création d'entité de base, car Symfony va en profiter pour générer un fichier important et relier l'ensemble avec l'entité "User".
- Répondez à la série de questions qui suivra.
- Détaillons ces questions/réponses :
  - *The name of the security user class (e.g. User) [User] :*

- Il s'agit ici de définir un nom pour notre entité.
- *Do you want to store user data in the database (via Doctrine)? (yes/no) [yes] :*
- Souhaitez-vous stocker les données de vos utilisateurs en base de données ? ?... Oui !
- *Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email] :*
- Il nous demande quelle propriété nous voulons utiliser comme nom d'affichage unique pour l'utilisateur : l'e-mail me convient très bien.
- *Does this app need to hash/check user passwords? (yes/no) [yes] :*
- Ici, la demande est de savoir si Symfony doit hacher et vérifier les mots de passe utilisateur. Évidemment, pour plus de sécurité, nous répondons yes.

◦ Maintenant que nous avons généré l'entité, nous devons mettre à jour la base de données :

- Pour créer une migration et appliquer les modifications à la base de données, exécutez les commandes suivantes dans votre terminal :

```
symfony console make:migration
symfony console doctrine:migrations:migrate
```

- Vous pouvez également utiliser les commandes alternatives suivantes :

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

- Si vous avez déjà des entités et que votre base de données contient déjà des tables SQL, vous pouvez utiliser la commande suivante pour mettre à jour le schéma de la base de données :

```
symfony console doctrine:schema:update --force
```

- Les commandes alternatives sont les suivantes :

```
php bin/console doctrine:update:schema --force
```

- Ces commandes permettent de synchroniser le schéma de votre base de données avec les entités de votre application Symfony. Assurez-vous de les exécuter après avoir effectué des modifications au niveau des entités afin de maintenir la cohérence de votre base de données.

apcpedagogie.com

- *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.*

4. Analyse du fichier security.yaml

- Maintenant, regardons le fichier security.yaml qui se trouve dans packages du dossier config : ***config\packages\security.yaml.***

```
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\Utilisateurs
                property: email
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider

            # activate different ways to authenticate
            # https://symfony.com/doc/current/security.html#the-firewall

            # https://symfony.com/doc/current/security/impersonating_user.html
            # switch_user: true
```

apcpedagogie.com

```
    # Easy way to control access for large sections of your site
    # Note: Only the *first* access control that matches will be used
    access_control:
        # - { path: ^/admin, roles: ROLE_ADMIN }
        # - { path: ^/profile, roles: ROLE_USER }

when@test:
    security:
        password_hashers:
            # By default, password hashers are resource intensive and take time.
This is
            # important to generate secure password hashes. In tests however, secure
hashes
            # are not important, waste resources and increase test times. The
following
            # reduces the work factor to the lowest possible values.
            Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
                algorithm: auto
                cost: 4 # Lowest possible value for bcrypt
                time_cost: 3 # Lowest possible value for argon
                memory_cost: 10 # Lowest possible value for argon
```

- 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.

5. Les 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 :

apcpedagogie.com

```
access_control:
       - { path: ^/, roles: ROLE_USER }
       - { path: ^/login, roles: PUBLIC_ACCESS }
       - { path: ^/users, roles: ROLE_ADMIN}
   role_hierarchy:
   ROLE_ADMIN:       ROLE_USER
```

6. Conclusion

> ◦ *Et voila, grâce a ces 4 commandes vous avez:*
>
> - *une entité User*
> - *une page de connexion*
> - *une page d'enregistrement*
> - *une page de récupération de mot de passe*

◦ Voici la liste des commandes utilisées au long de cet article. Vous pouvez les exécuter et vous laissez guider :

```
composer require symfonycasts/reset-password-bundle
php bin/console make:user
php bin/console make:security
composer require form validator
php bin/console make:registration-form
php bin/console make:reset-password
```

Cours Symfony

Source : https://apcpedagogie.com/la-securite-et-la-gestion-des-utilisateurs-dans-symfony · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com