Développement web côté serveur avec Symfony

Module 08 - Utilisateurs et sécurité

Objectifs

- Comprendre les concepts d'authentification et d'autorisation selon Symfony
- Savoir inscrire / connecter / déconnecter un utilisateur
- Savoir protéger des ressources du site

Le système de sécurité de Symfony

- Authentification vs Autorisation ?
- Très souple et donc assez complexe
- Plusieurs commandes *make* disponibles
- Configuration assez vaste, mais automatisée

Étapes pour créer un système d'authentification

1. Créer l'entité utilisateur
   - php bin/console make:user
2. Créer le système d'authentication
   - php bin/console make:auth
3. Créer le formulaire d'inscription
   - php bin/console make:registration-form
4. Autres
   1. Configurer la fonction "se souvenir de moi"
   2. Système d'oubli de mot de passe
   3. Lien de connexion magique
   ...

Utilisateurs et sécurité

Étape 1 : l'entité utilisateur

- Commande :
  - php bin/console make:user
- Configuration automatique dans :
  - config/packages/security.yaml
- Crée l'entité, un peu spéciale
- Ne crée pas la table !
- Pour ajouter d'autres propriétés :
  - php bin/console make:entity User

```
λ 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 pa
sswords are not needed or will be checked/hashed by some other sy
stem (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
```

Utilisateurs et sécurité

Étape 1 : l'entité utilisateur

La configuration

```yaml
security:
    # comment sont hachés nos mots de passe
    encoders:
        App\Entity\User: # pour cette entité...
            algorithm: auto # on utilise l'algo par défaut

    # d'où viennent nos utilisateurs ?
    providers:
        app_user_provider: # un nom unique
            entity: # nos utilisateurs sont en bdd
                class: App\Entity\User # c'est cette entité-là
                property: email # représentée par cette propriété-ci

    # activation du système de sécurité
    firewalls:
        # permet de désactiver sur la debug toolbar, le profiler, etc.
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        # notre pare-feu principal, activé partout par défaut
        main:
            anonymous: true
            lazy: true
            provider: app_user_provider

    # autorisations
    access_control:
```

Étape 1 : l'entité utilisateur

Le UserInterface de Symfony

- Notre entité doit l'implémenter
- Les 5 méthodes de l'interface :
  - getUsername() : retourne le nom de l'utilisateur connecté (pour la wdt)
  - getPassword() : retourne le mot de passe de l'utilisateur
  - getRoles() : retourne les rôles de l'utilisateur
  - getSalt() : inutile aujourd'hui
  - eraseCredentials : habituellement inutile

Étape 2 : système d'authentification

- Commande :
  - php bin/console make:auth
- Génère :
  - de la configuration
  - un contrôleur
  - un fichier Twig pour la connexion
  - un "Authenticator"

```
λ php bin/console make:auth

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

 The class name of the authenticator to create (e.g. AppCustomAut
henticator):
 > AppAuthenticator

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

Étape 2 : système d'authentification

La configuration

```yaml
security:
    # ...
    firewalls:
        # ...

        # notre pare-feu principal, activé partout par défaut
        main:
            anonymous: true
            lazy: true
            provider: app_user_provider
            # quelle est la classe Authenticator pour ce pare-feu ?
            guard:
                authenticators:
                    - App\Security\AppAuthenticator
            # sur quel nom de route Symfony doit-il nous connecter ?
            logout:
                path: app_logout
    # ...
```

Étape 2 : système d'authentification

Le contrôleur généré

- Vous pouvez le modifier !
- Une route pour afficher le formulaire de connexion
  - Symfony intercepte la requête pour traiter le formulaire
- Une route pour la déconnexion
  - Symfony intercepte la requête pour déconnecter l'utilisateur

Étape 2 : système d'authentification

Le fichier Twig généré

- Vous pouvez le modifier !
- Formulaire de connexion, en HTML
- Nom des <input> important
- "Remember me" à décommenter au besoin

Étape 2 : système d'authentification

L'Authenticator

- Dans src/Security/
- Gère la connexion
- Une ligne à modifier absolument :

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

    // For example : return new RedirectResponse($this->urlGenerator->generate('some_route'));
    throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
}
```

Étape 3 : formulaire d'inscription

- Commande :
  - php bin/console make:registration-fo
- Génère :
  - Un contrôleur
  - Un fichier Twig
  - Un formulaire Symfony
  - Une validation d'unicité dans notre e
  - (pas de configuration)

```
λ 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 registr
ation? (yes/no) [yes]:
 >

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

Étape 3 : formulaire d'inscription

Le contrôleur généré

- Comporte une méthode pour afficher et traiter l'inscription
- Hache le mot de passe pour nous
- Connecte l'utilisateur après inscription
- Modifiable bien sûr

Utilisateurs et sécurité

Étape 3 : formulaire d'inscription

Le fichier Twig
et le formulaire générés

- Validation du mot de passe
- Modifiables !

```php
$builder
    ->add('email')
    ->add('agreeTerms', CheckboxType::class, [
        'mapped' => false,
        'constraints' => [
            new IsTrue([
                'message' => 'You should agree to our terms.',
            ]),
        ],
    ])
    ->add('plainPassword', PasswordType::class, [
        // instead of being set onto the object directly,
        // this is read and encoded in the controller
        'mapped' => false,
        'constraints' => [
            new NotBlank([
                'message' => 'Please enter a password',
            ]),
            new Length([
                'min' => 6,
                'minMessage' => 'Your password should be at least {{ limit }} characters',
                // max length allowed by Symfony for security reasons
                'max' => 4096,
            ]),
        ],
    ])
;
```

Étape 3 : formulaire d'inscription

Déconnexion

- Envoyer l'utilisateur à la route "`app_logout`"
- Afficher un simple lien !

Démonstration

Autorisation et rôles : accéder à l'utilisateur connecté

- On récupère l'instance de notre entité au complet
- Dans un contrôleur avec :
  - `$this->getUser()`
- Dans un fichier Twig avec :
  - `app.user`

Utilisateurs et sécurité

Autorisations et rôles

- Les rôles : des chaînes commençant par ROLE_
- Plusieurs rôles possibles
- Stockés en base de données ou pas

Autorisations et rôles : restreindre l'accès à une page

- Plusieurs stratégies, principalement en fonction du rôle
- Dans le contrôleur, en PHP ou en annotations

```php
/**
 * @Route("/admin", name="admin_dashboard")
 */
public function dashboard(): Response
{
    $this->denyAccessUnlessGranted("ROLE_ADMIN");

    //ou

    $isAdmin = $this->isGranted("ROLE_ADMIN");
    if (!$isAdmin){
        throw new AccessDeniedException("Réservé aux admins !");
    }

    return $this->render('admin/dashboard.html.twig');
}
```

```php
/**
 * @IsGranted("ROLE_ADMIN")
 * @Route("/admin", name="admin_dashboard")
 */
public function dashboard(): Response
{
    return $this->render('admin/dashboard.html.twig');
}
```

Utilisateurs et sécurité

Autorisations et rôles : restreindre l'accès à une page

- Dans `security.yml` :

```yaml
security:
    # ...

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

- Plus centralisé
- Automatisé grâce à l'expression rationnelle

Utilisateurs et sécurité

Vérifier un rôle dans Twig

```twig
{# teste si un utilisateur est connecté #}
{% if app.user %}
    <a href="{{ path('app_logout') }}">Déconnexion</a>
{% endif %}

{# teste si l'utilisateur connecté possède ce rôle #}
{% if is_granted("ROLE_ADMIN") %}
    <a href="{{ path('admin_dashboard') }}">Back-office</a>
{% endif %}
```

Hiérarchisation des rôles

- Un rôle peut hériter de toutes les permissions d'un autre

#5D7A63

```yaml
security:
    # ...

    role_hierarchy:
        ROLE_ADMIN: ROLE_USER
        ROLE_SUPER_ADMIN: ROLE_ADMIN
```

Démonstration

Se souvenir de moi

- Système déjà mis en place par Symfony
- Décommenter la case à cocher dans le formulaire de connexion
- Activer le système dans la configuration :

```yaml
security:
    # ...

    firewalls:
        main:
            # ...
            remember_me:
                secret:   '%kernel.secret%'
                lifetime: 6048000 # 10 semaines
                path:     /
```

Récupération de mots de passe oubliés

- Difficile, long et risqué à coder soi-même
- Module complémentaire à installer avec :
  - composer require `symfonycasts/reset-password-bundle`
- Puis générer le système avec :
  - php bin/console make:reset-password
  - Suivre les indications !
- Génère un système sécuritaire de jetons de sécurité, avec expiration

Liens de connexion magiques

- Envoie un lien par mail, permettant de se connecter
- Quelques étapes simples :
  - Activer le système dans security.yaml
  - Créer une page où l'utilisateur renseigne son mail
  - Envoyer un message avec le lien
  - Créer une route de vérification, interceptée par Symfony
- Aucune table n'est créée dans la base de données

Démonstration

Conclusion

- Vous savez configurer le système de sécurité de Symfony pour un formulaire de connexion classique
- Vous savez authentifier un utilisateur
- Vous savez refuser l'accès de certaines ressources à un utilisateur
- Vous êtes en mesure de mettre en place des fonctionnalités modernes reliées à la sécurité