Documentation technique projet 8

Parlons du framework symfony

Symfony est le Framework PHP n°1 pour les applications web en France et en Europe.

symfony en quelques chiffres:

+2,000 contributeurs au projet
+600,000 développeurs utilisant Symfony
+2,000,000,000 de téléchargements

Version du site

Nécessite	version	sortie	Fin des correctifs de bugs	Fin des correctifs de sécurité
PHP 7.2 ou plus	5.4.9	Nov/2021	Nov/2024	Nov/2025

Table de matière

Implémentation de l'authentification

Fichier	description
src/entity/user.php	Entité de l'utilisateur
src/controller/securityController.php	Connexion et déconnexion
src/security/LoginAppAuthenticator.php	Méthode de l'authentification
config/packages/security.yaml	Configuration de la manière de l'authentification.
templates/security/login.html.twig	formulaire de connexion

Fonctionnement de l'authentification :

l'utilisateur souhaitant se connecter saisi son username et mot de passe dans la page "se connecter"(login.html.twig) puis valide avec le bouton se connecter, **src/security/LoginAppAuthenticator.php** se charge de l'authentification.

```php
public function authenticate(Request $request): Passport
{
    $username = $request->request->get('username', '');

    $request->getSession()->set(Security::LAST_USERNAME, $username);

    return new Passport(
        new UserBadge($username),
        new PasswordCredentials($request->request->get('password', '')),
        [
            new CsrfTokenBadge('authenticate',
$request->request->get('_csrf_token')),
        ]
    );
}
```

Le rôle d'un authenticator c'est simplement d'intercepter la requête et ensuite définir et de créer un passeport avec les informations nécessaires

Security.yml :

Symfony fournit de nombreux outils pour sécuriser votre application.
dans le security.yml, dans notre cas le mot de passe est crypté de manière automatique avec le meilleur algorithm disponible car on a choisi algorithm: auto

```
security:
  enable_authenticator_manager: true
  password_hashers:
Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
      App\Entity\User:
        algorithm: auto
```

providers

Toute section sécurisée de votre application nécessite un certain concept d'utilisateur. Le fournisseur d'utilisateurs charge les utilisateurs à partir de n'importe quel stockage sur la base d'un "identifiant d'utilisateur" (dans notre cas le **username** de l'entité User de l'utilisateur ;

```
providers:
  app_user_provider:
    entity:
      class: App\Entity\User
      property: username
```

firewalls

Le pare-feu est au cœur de la sécurisation de votre application. Chaque demande dans le pare-feu est vérifiée si elle nécessite un utilisateur authentifié. Le pare-feu se charge également d'authentifier cet utilisateur (par exemple à l'aide de notre formulaire de connexion) ;

```yaml
firewalls:
  dev:
    pattern: ^/(_(profiler|wdt)|css|images|js)/
    security: false
  main:
    lazy: true
    provider: app_user_provider
    custom_authenticator: App\Security\LoginAppAuthenticator
    logout:
      path: app_logout
```

En Résumé sur l'authentification

L'utilisateur tente d'accéder à une ressource protégée (par exemple /tasks/create) ;

Le pare-feu initie le processus d'authentification en redirigeant l'utilisateur vers le formulaire de connexion (/login) ;

La page /login affiche le formulaire de connexion via la route et le contrôleur créé dans src/controller/securityController.php ;

L'utilisateur soumet le formulaire de connexion;

Le système de sécurité (custom_authenticator) intercepte la demande, vérifie les informations d'identification soumises par l'utilisateur, authentifie l'utilisateur si elles sont correctes et renvoie l'utilisateur au formulaire de connexion si elles ne le sont pas.

Stockage des utilisateurs :

Les utilisateurs sont stockés dans la table **USER** de la base de données.

Entity USER

```php
/**
* @ORM\Entity(repositoryClass=UserRepository::class)
* @ORM\Table(name="`user`")
* @UniqueEntity("email")
*/
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     * @Assert\NotBlank(message="ce champ ne peux pas être vide")
     */
    private $username;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    /**
     * @var string The hashed password
     * @ORM\Column(type="string")
     * @Assert\NotBlank(message="Vous devez saisir un mot de passe ")
     */
    private $password;
```

les utilisateur stockés dans la base avec mysQL

+ Options

	id	username	roles	password	email
Éditer Copier Supprimer	1	username0	[]	$2y$13$LIZZ5esGrBlDrPVWwz98/Omh7VLgKTsGLMRlIECFj/p...	username0@gmail.com
Éditer Copier Supprimer	2	username1	[]	$2y$13$HwRAlE5rM1/PbNeg3t1qleAnaGnS9bSp5oJ0w/ZS7EV...	username1@gmail.com
Éditer Copier Supprimer	3	username2	[]	$2y$13$RauHVBObgPImThA/s.hesu1hNbNIky.u5faF1DXb3/h...	username2@gmail.com
Éditer Copier Supprimer	4	username3	[]	$2y$13$eIUCtLLseqaVGxa5VNN/s./VOd9VcDgA5RibDbJ/JFn...	username3@gmail.com
Éditer Copier Supprimer	5	username4	[]	$2y$13$rhdcigBZPMZBK6R4tEmAoOxGuSKRpjbODdtv3oGlyUu...	username4@gmail.com
Éditer Copier Supprimer	6	username5	[]	$2y$13$mKft7BirfuApmnD0DUsQmu0hI.DInsZomTJhi7NmN97...	username5@gmail.com

Ressources protégées :

Pour accéder à une ressource protégée par exemple un rôle admin :
on fait appel à la méthode `$this->denyAccessUnlessGranted('ROLE_ADMIN');`
afin d'autoriser l'accès seulement aux utilisateurs possédant un ROLE_ADMIN
comme l'exemple ci-dessous.

```
/**
* @Route("/users", name="user_list")
*/
public function listUsers(UserRepository $userRepository):Response
{
  $this->denyAccessUnlessGranted('ROLE_ADMIN');
  return $this->render('user/list.html.twig', [
     'users' => $userRepository->findAll()
  ]);
}
```