Documentation Technique - Implémentation de l'Authentification

Table des Matières

Vue d'ensemble

L'application ToDo & Co utilise **Symfony Security Component** pour gérer l'authentification et l'autorisation. Le système s'appuie sur :

- **Base de données** : Stockage des utilisateurs (Doctrine ORM)
- **Formulaires** : Interface de connexion sécurisée
- **Chiffrement** : BCrypt pour les mots de passe
- **Rôles** : ROLE_USER et ROLE_ADMIN pour les permissions

Architecture de l'authentification

Composants Principaux

Fichiers clés et modifications

1. `app/config/security.yml` ⚙ Configuration

C'est le **fichier maître** de toute l'authentification.

Où : `app/config/security.yml`

Qu'est-ce qui s'y passe :

- Configuration du chiffrement des mots de passe (BCrypt)
- Définition du provider de sécurité (Doctrine + User)
- Configuration du firewall (routes protégées/non protégées)
- Contrôle d'accès par rôle

Exemple de configuration :

```
security:
    encoders:
        AppBundle\Entity\User: bcrypt    # Utilise BCrypt pour hasher les mots de passe

    providers:
        doctrine:
            entity:
                class: AppBundle:User    # Classe User depuis Doctrine
                property: username       # Propriété utilisée pour chercher l'utilisateur

    firewalls:
        main:
            anonymous: ~                 # Accès anonyme autorisé initialement
            form_login:
                login_path: login        # Route du formulaire de connexion
                check_path: login_check  # Route qui traite la soumission
            logout: ~                    # Activation du logout

    access_control:
        - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }  # Login sans rôle
        - { path: ^/users, roles: ROLE_ADMIN }                    # Admin only
        - { path: ^/, roles: [ROLE_USER, ROLE_ADMIN] }           # Toutes les pages protégées
```

Pourquoi c'est important : C'est ici qu'on définit qui peut accéder à quoi. Toute nouvelle restriction d'accès se fait ici.

---

2. `src/AppBundle/Entity/User.php` 🗄️ Modèle de Données

C'est l'**entité Doctrine** représentant un utilisateur en base de données.

Où : `src/AppBundle/Entity/User.php`

Qu'est-ce qui s'y passe :

- Propriétés : `id`, `username`, `password`, `email`, `roles`
- Implémente `UserInterface` (interface Symfony Security)
- Les rôles sont stockés en JSON dans la base de données
- Les getters/setters permettent d'accéder aux données

Structure de la classe :

```
class User implements UserInterface
{
    private $id;              // ID unique en base
    private $username;        // Identifiant unique (login)
    private $password;        // Mot de passe hashé (BCrypt)
    private $email;           // Email unique
    private $roles = [];      // Tableau de rôles (JSON en BDD)

    // Méthodes obligatoires de UserInterface
    public function getUsername(): ?string { }
    public function getPassword(): ?string { }
    public function getRoles(): array { }
    public function getSalt() { }
    public function eraseCredentials() { }
}
```

Pourquoi c'est important :

- Symfony Security utilise cet objet User pour authentifier
- Les rôles sont lus depuis cet objet
- Le password doit être hashé (jamais en clair)

À retenir :

- La propriété `username` est utilisée pour chercher l'utilisateur
- La propriété `password` doit être hashée avec BCrypt
- Les `roles` déterminent ce que l'utilisateur peut faire

---

3. `src/AppBundle/Controller/SecurityController.php` — Authentification

C'est le **contrôleur qui gère le login/logout**.

Où : `src/AppBundle/Controller/SecurityController.php`

Qu'est-ce qui s'y passe :

- `loginAction()` : Affiche le formulaire de connexion
- `loginCheckAction()` : Traite la soumission du formulaire (Symfony le fait automatiquement)
- `logoutAction()` : Déconnecte l'utilisateur (Symfony le fait automatiquement)

Exemple de code :

```
class SecurityController extends Controller
{
    public function loginAction(Request $request)
    {
        // Récupère les erreurs de connexion (identifiants invalides, etc.)
        $authenticationUtils = $this->get('security.authentication_utils');
        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render(
            'AppBundle:Security:login.html.twig',
            ['last_username' => $lastUsername, 'error' => $error]
        );
    }

    // Les actions loginCheck et logout sont gérées automatiquement
    // par le security firewall, pas besoin de code personnalisé
}
```

Pourquoi c'est important :

- C'est la porte d'entrée du système d'authentification
- Symfony gère automatiquement la vérification du mot de passe
- Les erreurs d'authentification sont capturées et affichées

---

4. `src/AppBundle/Form/UserType.php` — Gestion des Utilisateurs

C'est le **formulaire pour créer/éditer des utilisateurs**.

Où : `src/AppBundle/Form/UserType.php`

Qu'est-ce qui s'y passe :

- Permet de créer un nouvel utilisateur
- Permet d'éditer les propriétés d'un utilisateur
- Champ `roles` : choix entre ROLE_USER et ROLE_ADMIN
- Validation des données

Exemple de code :

```
class UserType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('username', TextType::class)
            ->add('email', EmailType::class)
            ->add('password', PasswordType::class)
            ->add('roles', ChoiceType::class, [
                'choices' => [
                    'Utilisateur' => 'ROLE_USER',
                    'Administrateur' => 'ROLE_ADMIN'
                ],
                'multiple' => false,
                'expanded' => false
            ]);
    }
}
```

Pourquoi c'est important :

- C'est le formulaire qui crée/édite les utilisateurs
- Le champ `roles` permet de choisir le rôle

---

5. `src/AppBundle/Controller/UserController.php` 🔐 Gestion des Accès

C'est le **contrôleur qui gère les utilisateurs**.

Où : `src/AppBundle/Controller/UserController.php`

Qu'est-ce qui s'y passe :

- `listAction()` : Affiche la liste des utilisateurs
- `createAction()` : Crée un nouvel utilisateur
- `editAction()` : Édite un utilisateur existant
- Hachage du mot de passe avec BCrypt

Contrôle d'accès :

```
public function listAction()
{
    $this->denyAccessUnlessGranted('ROLE_ADMIN');  // Seul un ROLE_ADMIN peut accéder
    // ...
}
```

Pourquoi c'est important :

- C'est ici que les utilisateurs sont créés/édités
- Le mot de passe doit être hashé avant la sauvegarde
- La protection ROLE_ADMIN assure que seuls les admins gèrent les utilisateurs

---

Flux d'authentification

1️⃣ Utilisateur accède au formulaire de login

```
Utilisateur → GET /login
       ↓
SecurityController::loginAction()
       ↓
Affichage du formulaire Twig
```

2️⃣ Utilisateur soumet ses identifiants

```
Utilisateur → POST /login_check (username + password)
       ↓
Symfony Security Firewall
       ↓
1. Cherche l'utilisateur en base via le "username"
2. Compare le password soumis avec celui en base (BCrypt)
       ↓
Mot de passe correct ?
       ├─ OUI → Session créée, utilisateur connecté
       └─ NON → Erreur, retour au formulaire de login
```

3️⃣ Utilisateur accède à une page protégée

```
Utilisateur → GET /tasks (par exemple)
       ↓
Symfony Security Firewall
       ↓
L'utilisateur a-t-il une session valide ?
       ├─ OUI → Vérification du rôle (ROLE_USER ou ROLE_ADMIN)
       │   ├─ A le rôle ? → Accès autorisé
       │   └─ N'a pas le rôle ? → Erreur 403 (Forbidden)
       └─ NON → Redirection vers /login
```

4️⃣ Utilisateur se déconnecte

```
Utilisateur → GET /logout
       ↓
Symfony Security Firewall
       ↓
Session supprimée, cookies d'authentification effacés
       ↓
Redirection vers /
```

Stockage des utilisateurs

Où sont stockés les utilisateurs ?

Base de données → Table `user` (MySQL)

Structure de la table

```
CREATE TABLE user (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(25) UNIQUE NOT NULL,
    password VARCHAR(64) NOT NULL,
    email VARCHAR(60) UNIQUE NOT NULL,
    roles JSON,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Exemple de données

```
INSERT INTO user VALUES (
    1,
    'john_doe',
    '$2y$13$...',  -- Password hashé en BCrypt (jamais en clair)
    'john@example.com',
    '["ROLE_USER"]',  -- JSON avec les rôles
    '2024-01-15 10:30:00'
);

INSERT INTO user VALUES (
    2,
    'admin',
    '$2y$13$...',  -- Password hashé en BCrypt
    'admin@example.com',
    '["ROLE_ADMIN"]',  -- Admin a le rôle administrateur
    '2024-01-15 10:30:00'
);
```

Comment les données sont récupérées

Via Doctrine ORM :

```
// Dans SecurityController ou ailleurs
$user = $this->getDoctrine()
    ->getRepository('AppBundle:User')
    ->findOneBy(['username' => $username]);

// Récupérer les rôles
$roles = $user->getRoles();  // Retourne ['ROLE_ADMIN']
```

Rôles et autorisations

Rôles disponibles

Rôle	Description	Accès
ROLE_USER	Utilisateur normal	Pages publiques + créer/éditer ses tâches
ROLE_ADMIN	Administrateur	Tout + gestion des utilisateurs + supprimer toutes les tâches

Comment les rôles sont utilisés

Dans `security.yml` :

```
access_control:
    - { path: ^/users, roles: ROLE_ADMIN }      # Seuls les ROLE_ADMIN
    - { path: ^/, roles: [ROLE_USER, ROLE_ADMIN] }  # Les deux rôles
```

Dans les contrôleurs :

```
public function deleteTaskAction(Task $task)
{
    // Seul le créateur ou un admin peut supprimer
    if ($task->getAuthor() !== $this->getUser() && !$this->isGranted('ROLE_ADMIN')) {
        throw $this->createAccessDeniedException();
    }
    // Suppression...
}
```

Dans les templates Twig :

```
{% if is_granted('ROLE_ADMIN') %}
    <a href="/users">Gestion des utilisateurs</a>
{% endif %}
```

Guide du développeur

Pour un développeur junior rejoignant le projet

❓ "Comment fonctionne la connexion ?"

Réponse :

1. L'utilisateur remplit le formulaire sur `/login`
2. Symfony vérifie l'username + password en base de données
3. Si correct, une session est créée
4. L'utilisateur est redirigé vers `/`

Fichiers à regarder :

- app/config/security.yml - Configuration
- src/AppBundle/Controller/SecurityController.php - Code du login

❓ "Comment ajouter un nouvel utilisateur ?"

Réponse : Via la page `/users/create` ou avec du code :

```
$user = new User();
$user->setUsername('new_user');
$user->setEmail('new@example.com');
$user->setPassword($passwordEncoder->encodePassword($user, 'plainPassword'));
$user->setRoles(['ROLE_USER']);

$em = $this->getDoctrine()->getManager();
$em->persist($user);
$em->flush();
```

Fichiers à modifier :

- src/AppBundle/Controller/UserController.php - Logique de création
- src/AppBundle/Form/UserType.php - Formulaire

❓ "Comment ajouter une restriction d'accès pour une nouvelle page ?"

Réponse : Modifier `app/config/security.yml` :

```
access_control:
    - { path: ^/my-admin-page, roles: ROLE_ADMIN }  # Ajouter cette ligne
    - { path: ^/, roles: [ROLE_USER, ROLE_ADMIN] }
```

Fichiers à modifier :

- app/config/security.yml - Ajouter la restriction
- Optionnellement src/AppBundle/Controller/MyController.php - Ajouter `$this->denyAccessUnlessGranted('ROLE_ADMIN');`

❓ "Comment vérifier si l'utilisateur actuel a un rôle particulier ?"

Réponse :

```
// Dans un contrôleur
if ($this->isGranted('ROLE_ADMIN')) {
    // L'utilisateur est admin
}

// Dans un template Twig
{% if is_granted('ROLE_ADMIN') %}
    <!-- Afficher le menu admin -->
{% endif %}
```

❓ "Où sont stockés les mots de passe ?"

Réponse :

- **Base de données** → Table `user`, colonne `password`
- **Format** : Hashé en BCrypt (jamais en clair)
- **Exemple** : `$2y$13$R9h7cIPz0gi.URNNGHQ3OPST9/PgBkqquzi.Ss7KIUgO2t0jWMUm`

❓ "Comment éditer le profil d'un utilisateur ?"

Réponse : Via la page `/users/{id}/edit`

Code :

```
$user = $this->getDoctrine()
    ->getRepository('AppBundle:User')
    ->find($id);

// Formulaire UserType
$form = $this->createForm(UserType::class, $user);

if ($form->isSubmitted() && $form->isValid()) {
    // Si le password a changé, le hasher
    if ($user->getPassword()) {
        $user->setPassword($passwordEncoder->encodePassword($user, $user->getPassword()));
    }
    $em->flush();
}
```

Fichiers à modifier :

- src/AppBundle/Controller/UserController.php - Logique d'édition
- src/AppBundle/Form/UserType.php - Formulaire

Checklist pour les développeurs juniors

✅ Avant de modifier l'authentification

- [ ] Lire app/config/security.yml
- [ ] Comprendre la classe src/AppBundle/Entity/User.php
- [ ] Regarder les contrôleurs src/AppBundle/Controller/SecurityController.php et src/AppBundle/Controller/UserController.php
- [ ] Lancer les tests : `php bin/phpunit.phar tests/AppBundle/Controller/`

✅ Avant de committer

- [ ] Tous les tests passent
- [ ] Les mots de passe sont toujours hashés (jamais en clair)
- [ ] Les restrictions d'accès sont configurées dans `security.yml`
- [ ] Pas de données sensibles en dur dans le code

✅ Avant de merger une pull request

- [ ] Revoir les modifications dans `security.yml`
- [ ] Vérifier que les rôles utilisés existent
- [ ] S'assurer que le hashage des passwords est correct

- [ ] Tests d'authentification passent

Questions fréquentes

Q: Pourquoi utiliser BCrypt ?

R: BCrypt est un algorithme de hachage cryptographique puissant et lent intentionnellement. Cela rend les attaques par force brute extrêmement difficiles. Symfony utilise automatiquement BCrypt.

Q: Puis-je modifier le rôle d'un utilisateur après sa création ?

R: Oui ! Allez sur `/users/{id}/edit` et changez le rôle. Les autorisations seront mises à jour à la prochaine requête.

Q: Que se passe-t-il si j'oublie de hasher le password ?

R: C'est une **faille de sécurité majeure**. Les mots de passe seraient stockés en clair. Utiliser toujours le service `EncoderFactory` ou `PasswordEncoder`.

Q: Comment ajouter un nouvel utilisateur en base de données directement ?

R: Via phpmyadmin ou SQL :

```
INSERT INTO user (username, email, password, roles)
VALUES ('new_user', 'new@example.com', '$2y$13$...hashedpassword...', '["ROLE_USER"]');
```

Mais c'est mieux via l'application pour eviter les erreurs de hachage !

Ressources supplémentaires

- Symfony Security Documentation
- Doctrine ORM Guide
- BCrypt Reference
- OWASP - Authentication Cheat Sheet

Résumé

Concept	Fichier	Rôle
Configuration sécurité	`app/config/security.yml`	Définir qui peut accéder à quoi
Modèle User	`src/AppBundle/Entity/User.php`	Représenter l'utilisateur en base
Login/Logout	`src/AppBundle/Controller/SecurityController.php`	Gérer l'authentification
Gestion utilisateurs	`src/AppBundle/Controller/UserController.php`	Créer/éditer utilisateurs
Formulaire utilisateurs	`src/AppBundle/Form/UserType.php`	Valider les données utilisateur
Base de données	Table `user`	Stocker les utilisateurs

Cette documentation doit permettre à tout développeur junior de comprendre et de modifier le système d'authentification !