Documentation Technique - Implémentation de l'Authentification

3 JUIN 2024

TODO & Co
Créé par : Portemer Frederic

Introduction

Cette documentation explique comment l'implémentation de l'authentification a été réalisée dans l'application ToDo & Co. Elle est destinée aux développeurs débutants avec le framework Symfony.

Table des matières

Fichiers Modifiés et Raisons

1. config/packages/security.yaml

Ce fichier contient la configuration de la sécurité pour l'application. Il définit les fournisseurs d'utilisateurs, les firewalls, et les règles d'accès.

2. src/Entity/User.php

Ce fichier contient l'entité User qui représente les utilisateurs de l'application. Il inclut les champs pour le nom d'utilisateur, le mot de passe, l'email, et les rôles.

3. src/Security/UserAuthenticator.php

Ce fichier contient la classe UserAuthenticator qui gère l'authentification des utilisateurs. Il définit comment les utilisateurs sont authentifiés et redirigés après la connexion ou la déconnexion.

4. src/Controller/SecurityController.php

Ce fichier contient le contrôleur SecurityController qui gère les routes de connexion et de déconnexion.

5. config/routes.yaml

Ce fichier contient les définitions de routes pour les pages de connexion et de déconnexion.

Processus d'Authentification

- **Étapes de l'authentification**

1. Formulaire de Connexion:

- L'utilisateur accède à la page de connexion via /login.
- Il remplit le formulaire avec son nom d'utilisateur et son mot de passe.

2. Vérification des Informations d'Identification:

- Les informations du formulaire sont envoyées à UserAuthenticator.
- UserAuthenticator utilise le UserProvider défini dans security.yaml pour charger l'utilisateur à partir de la base de données.
- Le mot de passe est vérifié.

3. Gestion de la Session:

- Si les informations d'identification sont correctes, l'utilisateur est authentifié et une session est créée.
- L'utilisateur est redirigé vers la page d'accueil ou une autre page définie.

4. Déconnexion:

- Lorsqu'un utilisateur accède à /logout, la session est invalidée et l'utilisateur est redirigé.

- ## Exemple de Code

  ‘ config/packages/security.yaml ‘

```yaml
security:
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'

    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        main:
            lazy: true
            provider: app_user_provider
            custom_authenticator: App\Security\UserAuthenticator
            logout:
                path: app_logout
                target: app_homepage

    access_control:
        - { path: ^/tasks, roles: ROLE_USER }
        - { path: ^/users, roles: ROLE_ADMIN }
```

‘ src/Controller/SecurityController.php ‘

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class SecurityController extends AbstractController
{
    #[Route('/login', name: 'login')]
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security/index.html.twig', [
            'last_username' => $lastUsername,
            'error' => $error,
        ]);
    }

    #[Route('/logout', name: 'logout', methods: ['GET'])]
    public function logout(): void
    {
        // Le framework Symfony gère la déconnexion automatiquement
    }
}
```

Stockage des Utilisateurs

Les utilisateurs sont stockés dans la base de données, dans une table appelée user. L'entité User est définie dans src/Entity/User.php.

- **Exemple de l'entité User**

```php
<?php

namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

#[ORM\Entity]
#[ORM\Table(name: "user")]
#[UniqueEntity("email")]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue(strategy: "AUTO")]
    #[ORM\Column(type: "integer")]
    private $id;

    #[ORM\Column(type: "string", length: 25, unique: true)]
    #[Assert\NotBlank(message: "Vous devez saisir un nom d'utilisateur.")]
    private $username;

    #[ORM\Column(type: "string", length: 64)]
    private $password;

    #[ORM\Column(type: "string", length: 60, unique: true)]
    #[Assert\NotBlank(message: "Vous devez saisir une adresse email.")]
    #[Assert\Email(message: "Le format de l'adresse n'est pas correct.")]
    private $email;

    #[ORM\Column(type: "json")]
    private $roles = [];

    #[ORM\OneToMany(targetEntity: Task::class, mappedBy: "user", cascade: ["remove"])]
    private $tasks;

    public function __construct()
    {
        $this->tasks = new ArrayCollection();
    }

    // Méthodes de getter et setter...
}
```

Contribuer au Projet

Pour contribuer à ce projet, suivez les étapes ci-dessous :

1. **Forkez le repository** : Cliquez sur le bouton "Fork" en haut de la page du repository.
2. **Clonez votre fork** : Clonez le repository forké sur votre machine locale

```
git clone https://github.com/votre-utilisateur/ToDo-Co.git
cd ToDo-Co
```

3. **Créez une branche pour votre fonctionnalité :**
   Créez une nouvelle branche pour travailler sur votre fonctionnalité ou correction de bug.

```
git checkout -b feature/nom-de-la-fonctionnalité
```

4. **Faites vos modifications** : Apportez les modifications nécessaires dans le code.
5. **Commitez vos changements** : Enregistrez vos modifications avec un message de commit descriptif.

```
git add .
git commit -m "Description des changements"
```

6. **Poussez votre branche** : Envoyez vos modifications sur GitHub

```
git push origin feature/nom-de-la-fonctionnalité
```

7. **Ouvrez une Pull Request** : Ouvrez une Pull Request sur GitHub pour que vos modifications soient examinées et fusionnées.

Processus de Qualité

- **Revue de Code** : Chaque Pull Request doit être examinée et approuvée par au moins un autre développeur.
- **Tests** : Assurez-vous que tous les tests passent avant de soumettre une Pull Request. Ajoutez des tests pour toute nouvelle fonctionnalité ou correction de bug.
- **Conformité aux Standards** : Assurez-vous que votre code est conforme aux standards de codage de Symfony et respecte les bonnes pratiques de développement.

Audit de Qualité et de Performance

Pour garantir la qualité et la performance de l'application, il est important de réaliser des audits réguliers. Utilisez des outils comme Codacy ou CodeClimate pour auditer la qualité du code, et des outils comme Blackfire ou New Relic pour profiler les performances de l'application.

Exemple d'Audit de Qualité

1. **Codacy** :
    - **Intégrez Codacy avec votre repository GitHub.**
    - **Configurez les règles de qualité du code selon les standards de Symfony.**
    - **Analysez les rapports de Codacy et corrigez les problèmes signalés.**
2. **CodeClimate** :
    - **Intégrez CodeClimate avec votre repository GitHub.**
    - **Configurez les paramètres de qualité du code.**
    - **Examinez les rapports de CodeClimate et apportez les corrections nécessaires.**

Exemple d'Audit de Performance

1. **Blackfire** :
    - **Installez l'agent Blackfire sur votre serveur de développement.**
    - **Profiler l'application en exécutant des scénarios d'utilisation.**
    - **Analysez les résultats et identifiez les goulots d'étranglement.**
2. **Symfony Profiler** :
    - **Utilisez le profiler intégré de Symfony pour analyser les performances des requêtes et des processus.**
    - **Examinez les rapports générés et optimisez le code en conséquence.**

En suivant ces directives, vous pourrez contribuer efficacement au projet ToDo & Co et garantir une qualité et des performances optimales.