Documentation Technique : L'Implémentation de l'Authentification ToDoList

Introduction :

L'authentification est un élément essentiel de nombreuses applications Web, permettant de sécuriser l'accès aux fonctionnalités réservées aux utilisateurs enregistrés. Dans cette documentation, nous allons expliquer comment l'authentification a été mise en œuvre dans notre application Symfony, en fournissant des instructions détaillées pour les développeurs juniors qui rejoindront l'équipe.

Fichiers à modifier :

1. ***SecurityController :*** Ce contrôleur gère les actions liées à l'authentification, telles que l'affichage de la page de connexion, la vérification des informations d'identification soumises par l'utilisateur et la déconnexion. Vous pouvez trouver ce fichier à l'emplacement : **src/Controller/SecurityController.php**.
2. ***security.yaml :*** Ce fichier de configuration contient les paramètres de sécurité de l'application, tels que les fournisseurs d'utilisateurs, les pares-feux et les contrôles d'accès. Vous pouvez trouver ce fichier à l'emplacement : **config/packages/security.yaml**.
3. ***Entity\User :*** Cette entité représente l'utilisateur de l'application et contient les informations telles que le nom d'utilisateur, le mot de passe haché, les rôles, etc. Vous pouvez trouver ce fichier à l'emplacement : **src/Entity/User.php**.

```php
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;

You, 7 days ago | 1 author (You)
#[ORM\Table('user')]
#[ORM\Entity]
#[UniqueEntity('email')]
You, 7 days ago | 1 author (You)
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue(strategy: 'AUTO')]
    #[ORM\Column(type: 'integer')]
    private ?int $id = null;
```

Pour le hachage du mot de passe (très important en termes de sécurité !), assurez-vous que votre classe User implémente la PasswordAuthenticatedUserInterface.

Ensuite, configurez le hasheur de mot de passe à utiliser pour cette classe. Si votre fichier **security.yaml** n'était pas déjà préconfiguré comme ceci :

4. ***.env et .env.local :*** Ces fichiers de configuration contiennent les variables d'environnement, y compris les informations de connexion à la base de données. Vous devez configurer ces fichiers avec les paramètres appropriés pour permettre la connexion à la base de données.

Comment s'opère l'authentification :

L'authentification dans notre application Symfony est basée sur le mécanisme de formulaire de connexion. Voici les étapes générales du processus d'authentification :

1. L'utilisateur accède à la page de connexion (templates/security/login.html.twig) en utilisant l'URL appropriée (/login).

2. Lorsque le formulaire de connexion est soumis, les informations d'identification de l'utilisateur (nom d'utilisateur et mot de passe) sont envoyées au serveur.

3. Le contrôleur SecurityController vérifie les informations d'identification fournies en utilisant les fonctionnalités de sécurité de Symfony.

4. Si les informations d'identification sont valides, l'utilisateur est authentifié et une session est créée pour lui.

5. Si les informations d'identification sont invalides, l'utilisateur est redirigé vers la page de connexion avec un message d'erreur approprié.

6. Une fois authentifié, l'utilisateur peut accéder aux fonctionnalités réservées aux utilisateurs enregistrés.

Stockage des utilisateurs :

Les utilisateurs de notre application sont stockés dans une base de données à l'aide de **Doctrine**, qui est l'ORM (Object-Relational Mapping) par défaut de Symfony. L'entité User est utilisée pour représenter les utilisateurs et leurs informations sont stockées dans une table correspondante dans la base de données.

La configuration de Doctrine et la création de la table des utilisateurs sont effectuées dans les fichiers de configuration et les migrations de Doctrine.

Connexion à la base de données :

Pour que l'application puisse se connecter à la base de données, vous devez configurer les fichiers .env (l'exemple) et .env.local (le fichier à configurer) avec les informations de connexion appropriées. Ces fichiers contiennent des variables d'environnement telles que **DATABASE_URL** qui spécifient l'URL de connexion à la base de données. Vous devez fournir les informations de votre propre base de données, y compris le nom d'utilisateur, le mot de passe, le nom de la base de données, l'hôte, etc.

```
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8&charset=utf8mb4"
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=15&charset=utf8"
###< doctrine/doctrine-bundle ###
```

Fichier .env. Dans le fichier .env.local (dupliquer .env), modifier le nom, le mot de passe (s'il y en a un). Le deuxième "app" correspond au nom de la base de données que vous aurez créée dans phpMyAdmin (comme dans cet exemple).

Conclusion :

Cette documentation a présenté les principales composantes de l'implémentation de l'authentification dans notre application Symfony **ToDoList**. En suivant les instructions fournies et en examinant les fichiers mentionnés, les développeurs juniors devraient être en mesure de comprendre comment l'authentification a été mise en œuvre, où effectuer les modifications nécessaires et comment les utilisateurs sont stockés.

Documentation Symfony : https://symfony.com/doc/current/security.html