Guide de l'authentification sur le projet Todo & Co

La version markdown de cette doc est plus complète et contient des exemples de code, merci de la privilégier.

Introduction

Ce guide est destiné aux futurs collaborateurs du projet Todo & Co. Il vous aidera à comprendre le fonctionnement du système d'authentification basé sur Symfony 6.

1. Configuration de la sécurité : config/packages/security.yaml

Introduction

- password_hashers: Définit les algorithmes de hachage pour les mots de passe. Cela assure que les mots de passe sont stockés de manière sécurisée.
- providers: Définit comment les utilisateurs sont chargés à partir de la base de données. Permet de configurer différents types de stockage d'utilisateurs.
- firewalls: Établit les règles pour l'accès aux différentes parties de l'application. Vous pouvez configurer les firewalls pour différentes routes ou sections.
- access_control: Contrôle l'accès en fonction des rôles attribués aux utilisateurs. Utilisé pour restreindre l'accès à certaines routes aux utilisateurs ayant un rôle spécifique.

Conclusion

Cette section vous donne un aperçu de la configuration de la sécurité dans le projet Todo & Co avec Symfony 6. Les points clés à retenir sont les algorithmes de hachage de mots de passe, la manière dont les utilisateurs sont chargés à partir de la base de données, les firewalls, et les contrôles d'accès basés sur les rôles. Ces configurations sont essentielles pour le bon fonctionnement et la sécurité de votre application. Rendez-vous sur la version markdown de cette doc si vous voulez voir un exemple de code.

2. Controller SecurityController : src/Controller/SecurityController.php

Introduction

- login(): Cette méthode est responsable de la gestion de la page de connexion. Elle utilise AuthenticationUtils pour récupérer les erreurs d'authentification et le dernier nom d'utilisateur saisi, qui seront ensuite passés à la vue.
- logout(): Bien que cette méthode semble vide, elle est en fait gérée automatiquement par Symfony. Le mécanisme de déconnexion est configuré dans le fichier `security.yaml`.

Conclusion

Les méthodes login() et logout() du SecurityController sont centrales pour la gestion de la connexion dans Symfony 6. La première gère la page de connexion, tandis que la seconde est gérée automatiquement par Symfony. Rendez-vous sur la version markdown de cette doc si vous voulez voir un exemple de code.

3. Authentificateur : src/Security/LoginFormAuthenticator.php

Introduction

- authenticate(): Cette méthode est appelée lorsqu'un utilisateur tente de se connecter. Elle crée un *passeport* qui encapsule les données de l'utilisateur. Le passeport contient des badges, qui sont des composants responsables de collecter des informations spécifiques sur l'utilisateur (comme le nom d'utilisateur, le mot de passe, les jetons CSRF, etc.).
- onAuthenticationSuccess(): Cette méthode est appelée lorsque l'authentification est réussie. Symfony utilise les informations contenues dans le TokenInterface pour connaître les rôles et les permissions de l'utilisateur. Ensuite, selon la configuration ou la logique métier, cette méthode redirige l'utilisateur vers une page appropriée.

Conclusion

Cette section donne un aperçu de la logique d'authentification personnalisée dans Symfony, en se concentrant sur l'utilisation de "passeports" et "badges" pour la collecte et la validation des données d'authentification. Après une authentification réussie, les rôles et les permissions de l'utilisateur sont encapsulés dans un TokenInterface, ce qui permet à Symfony de gérer les accès. Rendez-vous sur la version markdown de cette doc si vous voulez voir un exemple de code.

4. Entity User : src/Entity/User.php

Introduction

- UserInterface(): Cette interface est fondamentale dans le système de sécurité de Symfony. Elle formalise les méthodes que votre classe User doit implémenter pour interagir avec le système de sécurité de Symfony. Les méthodes les plus notables sont :
  - getUserIdentifier(): Utilisée pour identifier l'utilisateur.
  - getRoles(): Utilisée pour obtenir les rôles attribués à l'utilisateur.
  - getPassword(): Utilisée lors de l'authentification pour valider le mot de passe de l'utilisateur.
  - eraseCredentials(): Utilisée pour effacer les données sensibles une fois qu'elles ne sont plus nécessaires.

Méthodes importantes

- getEmail(): Utilisée pour obtenir l'adresse e-mail de l'utilisateur, souvent utilisée comme identifiant unique dans des applications modernes.
- getRoles(): Retourne un tableau des rôles attribués à cet utilisateur. Les rôles sont essentiels dans le système de sécurité pour déterminer ce que l'utilisateur est autorisé à faire.
- getPassword(): Retourne le mot de passe haché de l'utilisateur. Ce mot de passe est utilisé lors de l'authentification pour valider que l'utilisateur est bien celui qu'il prétend être.

Conclusion

Dans cette section, nous avons exploré les responsabilités de la classe User dans le cadre du système de sécurité de Symfony. Elle implémente plusieurs interfaces essentielles et méthodes pour gérer des aspects tels que l'identification, l'authentification par mot de passe et les rôles de l'utilisateur.

5. Vue Login : templates/security/login.html.twig

Introduction

La vue Twig représente le formulaire de connexion dans l'application. Elle utilise le moteur de templates Twig pour rendre les données dynamiques.

Conclusion

La directive `{% if error %}` vérifie s'il y a une erreur d'authentification et, si c'est le cas, affiche un message d'erreur. La directive `{% if app.user %}` vérifie si l'utilisateur est déjà connecté.
Le champ caché `_csrf_token` ajoute une mesure de sécurité contre les attaques CSRF. Il est généré par Symfony et doit correspondre à celui stocké côté serveur pour que le formulaire soit accepté.
Rendez-vous sur la version markdown de cette doc si vous voulez voir un exemple de code.

6. Conclusion Générale

Cette documentation a abordé les éléments clés de l'authentification dans Symfony 6 : de la configuration dans security.yaml, au contrôleur de sécurité et aux entités utilisateur, jusqu'à la vue de connexion. Chaque composant joue un rôle spécifique pour sécuriser l'application, que ce soit en gérant le hachage des mots de passe, en authentifiant les utilisateurs, ou en contrôlant les accès. L'objectif est de fournir une compréhension claire de chaque composant pour faciliter la maintenance et l'évolution du système d'authentification.