Documentation d'Authentification pour les Développeurs Juniors

Introduction

Cette documentation vise à expliquer l'implémentation de l'authentification dans notre application Symfony à tout développeur junior rejoignant l'équipe. Vous y trouverez des informations sur les fichiers impliqués, le processus d'authentification et la gestion des utilisateurs.

I. Concepts de Base

L'authentification est le processus de vérification de l'identité d'un utilisateur dans un système informatique. Il implique généralement l'utilisation d'informations d'identification telles que des noms d'utilisateur, des adresses e-mail et des mots de passe pour valider l'identité de l'utilisateur.

Utilisation de Symfony Security : Symfony Security est un composant puissant qui simplifie la gestion de l'authentification et de l'autorisation dans les applications Symfony. Il fournit une infrastructure flexible pour gérer les utilisateurs, les rôles et les autorisations, ce qui permet aux développeurs de définir facilement des règles de sécurité spécifiques pour leur application.

Configuration de Sécurité : La configuration de la sécurité dans Symfony se fait principalement à travers le fichier security.yaml. Ce fichier permet de définir différents aspects de la sécurité de l'application, tels que les fournisseurs d'utilisateurs, les pare-feux (firewalls), les contrôles d'accès (access controls), et les mécanismes d'authentification personnalisés.

Personnalisation de l'Authentification : Symfony permet aux développeurs de personnaliser le processus d'authentification en utilisant des authentificateurs personnalisés. Cela leur permet de mettre en place des mécanismes d'authentification spécifiques à leur application, tels que l'authentification par e-mail, par SMS ou par jeton.

Gestion des Utilisateurs et des Rôles : Dans Symfony, la gestion des utilisateurs et des rôles est généralement effectuée à travers des entités d'utilisateur et des annotations de sécurité telles que @IsGranted. Les développeurs peuvent créer, modifier et supprimer des utilisateurs, ainsi qu'attribuer des rôles et des autorisations en fonction des besoins de leur application.

II. Implémentation de l'Authentification

L'authentification a été mise en œuvre en suivant les étapes suivantes :

1. Création de l'entité User à l'aide de la commande symfony console make:user.
2. Ajout du champs supplémentaires email à l'entité User en utilisant la commande symfony console make:entity. (le champs rôles est implémenté automatiquement par le maker)
3. Utilisation du Maker Bundle pour la création de l'authentificateur avec la commande symfony console make:auth.
4. Configuration automatique des fichiers de sécurité (security.yaml, etc.) par le Maker Bundle.

III. Processus d'Authentification

L'authentification des utilisateurs se déroule comme suit :

L'utilisateur accède à la page de connexion (/login) où il saisit son nom d'utilisateur et son mot de passe.
Les informations saisies sont envoyées au contrôleur SecurityController qui gère le processus d'authentification.
Le contrôleur utilise l'authenticator UserAuthenticator pour vérifier les informations d'identification de l'utilisateur.
Si les informations sont valides, l'utilisateur est redirigé vers la page principale de l'application.
Sinon, un message d'erreur est affiché.

Les principaux fichiers impliqués dans l'authentification sont :

SecurityController.php : Ce contrôleur gère les routes de connexion et de déconnexion de l'utilisateur.
User.php : Cette entité représente les utilisateurs de l'application et contient leurs informations telles que le nom d'utilisateur, le mot de passe et les rôles.
UserAuthenticator.php : Cet authenticateur gère le processus d'authentification des utilisateurs.

IV. Le fichier config > packages > security.yaml

password_hashers

Cette section configure les algorithmes de hachage de mot de passe utilisés pour stocker les mots de passe des utilisateurs. Dans cet exemple, les mots de passe des utilisateurs implémentant PasswordAuthenticatedUserInterface seront hachés automatiquement.

Modifications Possibles : Vous pouvez ajouter d'autres algorithmes de hachage de mot de passe selon vos besoins de sécurité, mais il est recommandé de rester avec les algorithmes recommandés par Symfony.

providers

Définit les fournisseurs de données d'authentification, dans ce cas, un fournisseur d'entités utilisateurs qui charge les utilisateurs depuis la base de données en fonction de leur nom d'utilisateur.

Modifications Possibles : Vous pouvez modifier le fournisseur d'utilisateur pour utiliser une autre source d'authentification, comme LDAP ou un service externe.

firewalls

Les pare-feu définissent les règles d'accès et de sécurité pour différentes parties de votre application. Dans cet exemple, il y a deux pare-feux : un pour l'environnement de développement et un pour l'application principale.

Modifications Possibles : Vous pouvez ajouter de nouveaux pare-feux pour des parties spécifiques de votre application et configurer les options d'authentification, comme les authentificateurs personnalisés et les options de rappel de connexion.

access_control

Contrôle l'accès aux différentes parties de l'application en fonction du rôle de l'utilisateur. Dans cet exemple, il est actuellement commenté, mais vous pouvez décommenter et ajouter des règles pour restreindre l'accès à certaines URL en fonction des rôles des utilisateurs.

Modifications Possibles : Ajoutez des règles d'accès pour restreindre l'accès à certaines parties de votre application en fonction des rôles des utilisateurs.

when@test

Cette section de la configuration de sécurité est utilisée spécifiquement lors de l'exécution de tests. Elle ajuste les paramètres de hachage de mot de passe pour réduire le temps d'exécution des tests en utilisant des valeurs moins sécurisées mais plus rapides.

Modifications Possibles : Vous pouvez ajuster les paramètres de hachage de mot de passe pour les tests selon les besoins de votre application, mais assurez-vous de maintenir un équilibre entre sécurité et performances.

V. Stockage des Utilisateurs

Les utilisateurs sont stockés dans la base de données MySQL. L'entité User contient les informations des utilisateurs telles que leur nom d'utilisateur, leur mot de passe (hashé), leur email et leurs rôles.

VI. Gestion des Droits et des Rôles

Dans notre application, nous avons mis en place un système de gestion des droits et des rôles pour contrôler l'accès aux fonctionnalités en fonction du niveau d'autorisation de l'utilisateur. Les rôles déterminent quelles actions un utilisateur peut effectuer dans l'application.

Nous avons défini deux rôles principaux :

ROLE_USER : Ce rôle est attribué par défaut à tous les utilisateurs enregistrés dans l'application. Il leur donne accès aux fonctionnalités de base.
ROLE_ADMIN : Ce rôle est réservé aux administrateurs de l'application. Les utilisateurs avec ce rôle ont un accès étendu et des autorisations supplémentaires
Les fichiers impliqués dans la gestion des droits et des rôles sont principalement :

UserController.php : Ce contrôleur gère les actions liées à la gestion des utilisateurs, telles que la création, la modification et la suppression. Les annotations @IsGranted('ROLE_ADMIN') sont utilisées pour restreindre l'accès à certaines actions aux seuls utilisateurs ayant le rôle administrateur.
TaskController.php : Ce contrôleur gère les actions liées aux tâches, comme la création, la modification et la suppression. De même, les annotations @IsGranted('ROLE_USER') sont

utilisées pour restreindre l'accès à certaines actions aux utilisateurs enregistrés.
UserFormType.php : Ce formulaire permet la création et la modification des utilisateurs, en incluant une option pour attribuer le rôle administrateur à un nouvel utilisateur.

Modification des Fichiers

Les développeurs juniors peuvent modifier ces fichiers pour ajuster les autorisations en fonction des besoins de l'application. Par exemple :

- Ajouter de nouvelles actions dans les contrôleurs et définir les annotations @IsGranted appropriées pour restreindre l'accès.

Modifier le formulaire UserFormType pour inclure d'autres options de rôles ou modifier le comportement en fonction des besoins spécifiques de l'application.

VII. Sécurité et Bonnes Pratiques

Bonnes Pratiques de Sécurité : Pour garantir la sécurité de l'authentification dans une application Symfony, il est essentiel de suivre les bonnes pratiques de sécurité. Cela comprend le hachage sécurisé des mots de passe, la protection contre les attaques par force brute, l'utilisation de connexions sécurisées (HTTPS) et la sensibilisation aux vulnérabilités courantes telles que les injections SQL et les attaques de script intersites (XSS).

Tests d'Authentification : Les tests d'authentification sont importants pour valider le bon fonctionnement du processus d'authentification dans une application Symfony. Les développeurs doivent écrire des tests unitaires et fonctionnels pour tester différents scénarios d'authentification, y compris les cas d'utilisation normaux, les scénarios d'erreur et les situations de bord. Cela garantit que l'authentification fonctionne correctement et en toute sécurité dans toutes les situations.

Conclusion

Il est essentiel de maintenir la sécurité de l'application en garantissant que seuls les utilisateurs autorisés peuvent effectuer des actions sensibles. Par conséquent, lors de l'ajout ou de la modification des autorisations, il est important de vérifier attentivement les implications de ces changements et de s'assurer qu'ils sont conformes aux exigences de sécurité de l'application.

En suivant ces pratiques, les développeurs peuvent contribuer efficacement à la gestion des droits et des rôles dans notre application Symfony tout en garantissant sa sécurité et sa fonctionnalité. Si des questions subsistent, n'hésitez pas à demander à un membre plus expérimenté de l'équipe pour obtenir de l'aide ou des éclaircissements supplémentaires.