

ToDo & Co - Todolist

Documentation technique - implémentation de l'authentification



Nicolas Halberstadt

Table Des Matières

ToDo & Co - Todolist	1
1. Sécurité	3
2. L'entité `User`	3
3. Les providers	3
4. L'encoder	3
5. Les firewalls	4
6. L'access control	4
7. Role hierarchy	5
8. Pour aller plus loin...	5

1. Sécurité

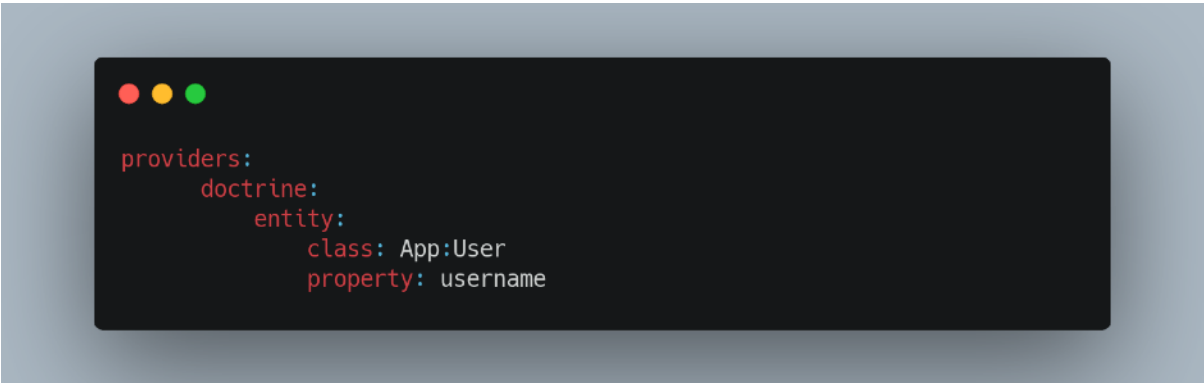
La sécurité de l'authentification de l'application est configurée dans le fichier `config/packages/security.yaml`. Plus d'informations sont disponibles concernant ce fichier sur la [documentation officielle](#).

2. L'entité `User`

L'entité `User` est la base de l'authentification Symfony. Cette classe doit implémenter l'interface `UserInterface` et implémenter les méthodes sous-jacentes de cette dernière. Le projet ToDo & Co comporte déjà cette classe, située dans le fichier `src/Entity/User.php`.

3. Les providers

Afin de compléter la configuration de l'application Symfony, nous avons besoin d'un `provider`. Il va permettre d'indiquer les informations que l'on souhaite utiliser pour authentifier les utilisateurs. Dans notre cas, nous indiquons que l'on récupérera les utilisateurs via Doctrine, dans l'entité User en utilisant la propriété Username pour s'authentifier.



```
providers:
  doctrine:
    entity:
      class: App\User
      property: username
```

4. L'encoder

L'encoder va permettre à l'application de déterminer l'algorithme choisi pour l'encodage d'une information liée à une entité. Dans notre cas, on va demander à l'application d'utiliser l'encoder par défaut de Symfony.



```
encoders:
  App\Entity\User:
    algorithm: auto
```

5. Les firewalls

Le firewall correspond au système d'authentification et permet de paramétrer la façon dont va être authentifié les utilisateurs (Token API, formulaire de connexion etc.). Chaque requête n'a qu'un seul firewall qui est choisi en fonction de la clé `schéma` du fichier de configuration. Le firewall `main` englobe toutes les requêtes, y compris lorsque l'utilisateur n'est pas authentifié. Ici, on indique le formulaire de connexion avec `form_login` puis le nom des routes qui vont servir à vérifier la connexion et la route vers laquelle vont être redirigé les utilisateurs après l'authentification.

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

  main:
    anonymous: ~
    pattern: ^/
    form_login:
      login_path: login
      check_path: login_check
      always_use_default_target_path: false
      default_target_path: /
    logout:
      path: /logout
      target: /
```

6. L'access control

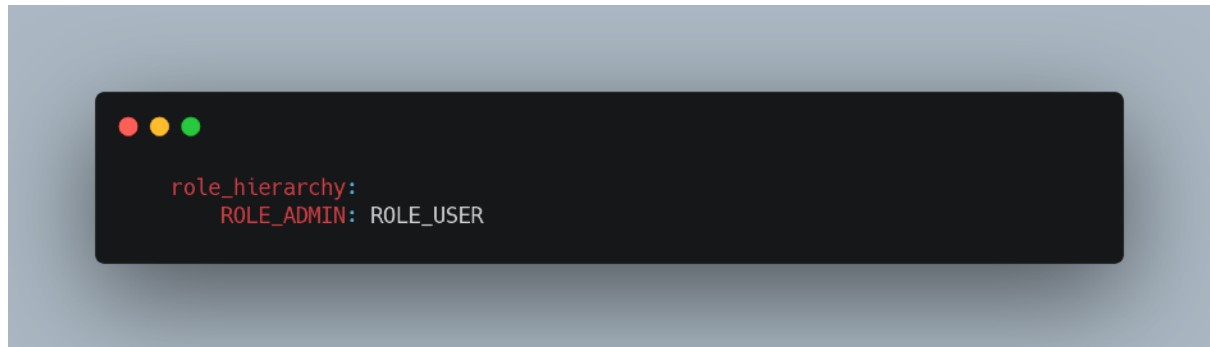
L'access control a pour but de définir des limitations a certaines url du projet. Pour mettre en place un access control, on indique :

- l'url avec la clé `path`
- Les `roles` nécessaires afin d'accéder à cette url. Le role peut-être un role prédéfini ou un état d'authentification (anonyme, entièrement authentifié...).

```
access_control:
- { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/users/create, roles: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/users, roles: ROLE_ADMIN }
- { path: ^/tasks, roles: IS_AUTHENTICATED_FULLY }
```

7. Role hierarchy

La mise en place du [role hierarchy](#) permet d'indiquer à l'application qu'un rôle utilisateur hérite d'un autre rôle. Dans notre cas, les utilisateurs ayant le rôle [ROLE_ADMIN](#) auront automatiquement les privilèges associés au rôle utilisateur



8. Pour aller plus loin...

Pour rappel, la [documentation](#) de Symfony est très complète et permet d'aller plus loin dans la personnalisation de la sécurité et de l'authentification des utilisateurs, n'hésitez pas à y jeter un oeil.