

A thick teal vertical bar runs down the left side of the page. A teal arrow points to the right from this bar, containing the date.

03/06/2022

Améliorez une application existante de ToDo & Co

Projet P8 – OpenClassRooms

Notice technique

Several thin, curved teal lines originate from the bottom left and sweep upwards and to the right, creating a sense of movement.

Serge Pillay

Table des matières

Présentation du projet	2
Technologies employées	2
Librairies	2
Installation du projet	2
Paramétrage du site	3
Fichiers d'authentification et d'autorisation	3
Composant de sécurité : le fichier security.yaml.....	3
Encodage du mot de passe :	3
Définition de l'entité utilisée pour retrouver les utilisateurs	3
Le pare-feu	3



Présentation du projet

Technologies employées

Le présent projet a été mise à jour vers une version de Symfony stable et dont la maintenabilité est prévue jusqu'en 2025.

- Symfony CLI version 5.4.8 (c) 2017-2022 Symfony SAS (2022-04-20T07:48:08Z - stable)
- PHP 7.4.30 (cli) (built: Jun 7 2022 16:24:55) (ZTS Visual C++ 2017 x64)
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.30, Copyright (c), by Zend Technologies
with Xdebug v3.1.5, Copyright (c) 2002-2022, by Derick Rethans
with blackfire v1.78.0~win-x64-zts74, <https://blackfire.io>, by Blackfire
- MySQL 5.7.31 Community Server (GPL)

Librairies

Les librairies employées ont été installées avec Composer

(<https://getcomposer.org/>)

Les librairies nécessaires dans le futur de l'application devront être également installées avec Composer. Les librairies installables avec Composer sont listées sur le site

<https://packagist.org>.

Pour les tests, la librairie PHPUnit est installée. Pour avoir un rendu HTML de la couverture des tests, vous devrez installer xDebug sur votre serveur (<https://xdebug.org/docs/install>)

Installation du projet

L'installation du projet est décrite en détails dans le fichier README.md à la racine du dépôt Github (<https://github.com/Urza45/todolist#readme>)

1. Cloner le Repository sur votre serveur web : **git clone git@github.com:Urza45/todolist.git**
2. Installez les dépendances, dans une invite de commande : **composer install**
(Composer doit être installés sur votre serveur)
3. Configurer la connexion BDD sur le fichier **.env**
4. Créer une base de données, migrer les tables et charger les fixtures, dans une invite de commande : **composer prepare**
5. Compte créé par les fixtures :
 - Login : admin , mot de passe : admin
 - Login : user1 , mot de passe : admin

Paramétrage du site

Fichiers d'authentification et d'autorisation

Type	Fichier	Description
Configuration	config/packages/security.yaml	Configuration du processus d'authentification
Entité	src\Entity\User.php	Entité utilisateur
Contrôleur	src\Controller\SecurityController.php	Contrôleur connexion/déconnexion
Authentification	src\Security\AppAuthenticator.php	Méthodes du processus d'authentification
Autorisation	src\Security\Voter\TaskVoter.php	Méthodes d'autorisations suivant les rôles des utilisateurs
Vue	templates/security/login.html.twig	Template du formulaire de connexion.

Composant de sécurité : le fichier security.yaml

Encodage du mot de passe :

Le mot de passe est crypté dans la base de données pour plus de sécurité.

L'encryptage est en mode automatique ce qui est le mode par défaut pour Symfony 5. Il est bien sûr possible de changer le mode d'encryptage dans le fichier security.yaml. (Voir la documentation <https://symfony.com/doc/5.4/security/passwords.html>)

```
1 security:
2     enable_authenticator_manager: true
3     # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
4     password_hashers:
5         Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: "auto"
```

Définition de l'entité utilisée pour retrouver les utilisateurs

```
# https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: username
```

Le provider utilise la classe User. Ici on peut configurer :

- L'entité sélectionnée pour créer l'utilisateur
- La propriété pour l'identification de l'utilisateur (ici username)
- L'ORM sélectionnée pour la connexion à la BDD de celui-ci (Doctrine).

Le pare-feu

Un seul pare-feu est actif sur chaque requête : Symfony utilise la clé de modèle pour trouver la première correspondance (vous pouvez également faire correspondre par hôte ou d'autres choses).

Le pare-feu de développement est vraiment un faux pare-feu : il s'assure que vous ne bloquez pas accidentellement les outils de développement de Symfony - qui possèdent des URL comme `/_profiler` et `/_wdt`.

Toutes les URL réelles sont gérées par le pare-feu principal

(Aucune clé de modèle ne signifie qu'elle correspond à toutes les URL).

Un pare-feu peut avoir plusieurs modes d'authentification, c'est-à-dire qu'il permet plusieurs façons de poser la question "Qui êtes-vous ?".

Souvent, l'utilisateur est inconnu (c'est-à-dire qu'il n'est pas connecté) lorsqu'il visite votre site Web pour la première fois. Si vous visitez votre page d'accueil en ce moment, vous y aurez accès et vous verrez que vous visitez une page derrière le pare-feu dans la barre d'outils :

The image shows a Symfony toolbar at the top with the following information: status 200, location @ homepage, response time 643 ms, memory usage 12.0 MiB, cache status 86 in 3.39 ms, user n/a, and a redirect icon with 12 ms. Below the toolbar is a dropdown menu for the firewall configuration. The menu is titled 'Authenticated' with a 'No' button and 'Firewall name' with the value 'main'. The configuration is as follows:

```
firewalls:
  dev:
    pattern: ^/(_(profiler|wdt)|css|images|js)/
    security: false
  main:
    lazy: true
    provider: app_user_provider
    custom_authenticator: App\Security\AppAuthenticator
    logout:
      path: logout
      # where to redirect after logout
      # target: app_any_route

# activate different ways to authenticate
# https://symfony.com/doc/current/security.html#the-firewall

# https://symfony.com/doc/current/security/impersonating_user.html
# switch_user: true
```

Contrôle des accès

Vous pouvez configurer l'accès de vos utilisateurs à différentes routes en fonction de leurs rôles :

```
# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
  - { path: ^/admin, roles: ROLE_ADMIN }
  - { path: ^/profile, roles: ROLE_USER }
  - { path: ^/users, roles: ROLE_ADMIN }
  - { path: ^/tasks, roles: ROLE_USER }
```

Ici, il faut avoir le rôle `ROLE_ADMIN` pour accéder aux routes commençant par `/users`.

Composant d'autorisation le fichier TaskVoter.php

La classe TaskVoter vérifie dans un premier temps que la personne qui navigue sur le site est bien une personne connectée :

```
34     $user = $token->getUser();
35     // if the user is anonymous, do not grant access
36     if (!$user instanceof UserInterface) {
37         return false;
38     }
```

Ensuite, elle vérifie suivant le critère demandé, si l'utilisateur en question a bien accès à cette fonctionnalité :

4 critères qui correspondent au CRUD d'une tâche (Create, Read, Update, Delete).

```
// ... (check conditions and return true to grant permission) ...
switch ($attribute) {
    case self::TASK_CREATE:
        // logic to determine if the user can CREATE
        // return true or false
        return $this->canCreate($task, $user);
        break;
    case self::TASK_EDIT:
        // logic to determine if the user can EDIT
        // return true or false
        return $this->canEdit($task, $user);
        break;
    case self::TASK_VIEW:
        // logic to determine if the user can VIEW
        // return true or false
        return $this->canView($task, $user);
        break;
    case self::TASK_DELETE:
        // logic to determine if the user can DELETE
        // return true or false
        return $this->canDelete($task, $user);
        break;
}
```

Si un critère correspond, on fait appel à la fonction correspondante, sinon on retourne false.

TASK_DELETE => on appelle la fonction canDelete() qui retourne un booléen.

Stockage des utilisateurs

Les données des utilisateurs sont stockées dans la base de données MySQL, dans la table User.

Dans Symfony 5 cette table est représentée par l'entité User (src\Entity\User.php)

Cette classe implémente `UserInterface` et `PasswordAuthenticatedUserInterface`, indispensable pour la création d'une classe utilisateurs et l'accès aux différentes fonctions liées à la gestion d'un utilisateur.

Le champ utilisé pour identifier de manière unique un utilisateur (en dehors de l'id) est le champ `username`. Dans l'entité cela se concrétise par l'ajout de « `unique=true` » dans l'annotation `@ORM\Column` de la variable `$username` :

```
/**
 * @ORM\Column(type="string", length=180, unique=true)
 */
private $username;
```

Un script a été écrit dans le fichier `composer.json` afin de récréer la base de données et de charger les données fictives (Cela réinitialise complètement la base de données)

```
"scripts": {
    "prepare": [
        "php bin/console doctrine:database:drop --if-exists -f",
        "php bin/console doctrine:database:create",
        "php bin/console doctrine:schema:update -f",
        "php bin/console doctrine:fixtures:load -n"
    ],
}
```

Fonctionnement de l'authentification

Page d'authentification

Dans un premier temps, la personne va devoir rentrer ces identifiants sur la page **login.html.twig** à la route **/login**. Pour accéder à cette page, la méthode **login** est exécutée, sur le Controller : **SecurityController**.

Cette méthode sert à générer la page, et à envoyer des données à la vue avec un `render`.

Soumission du formulaire

Quand le formulaire est validé par l'utilisateur, les données sont récupérées par la classe **AppAuthenticator**, qui se charge de l'authentification. Cette classe est générique à la création du système d'authentification de la librairie `Security-Bundle`.

L'authentification est vérifié par la méthode **authenticate(...)**

Redirection de l'utilisateur

Suivant que la personne s'est bien authentifiée ou non, la méthode **onAuthenticationSuccess**, la redirigera vers la dernière page qu'elle a consulté, ou la page de défaut, en cas de succès. A contrario, un message d'erreur sera affiché au-dessus du formulaire de connexion.