Documentation technique : TODO & CO

Pour OpenClassrooms, par Nicolas JARACH

Tables des Matières

Système d'authentification

Comme tous les projets symfony, il existe des fichiers de configuration stratégiques qui permettent de paramétrer l'application et la faire fonctionner. L'authentification s'appuie sur des stratégies déjà mises en place par des composants symfony installés par défaut et facilement dans le projet.

Le fichier security.yaml

Il contient toutes les informations de configuration du système de sécurité mis en place. Il est impératif de se renseigner via la documentation officielle de symfony avant de réaliser des modifications

```yaml
# Define user providers
providers:
  app_user_provider:
    entity:
      class: App\Entity\User
      property: username
```

C'est ici que vous paramétrez l'Entity symfony qui sera considérée comme un utilisateur lors des différentes évaluations effectuées par le système. Le nom du paramètre 'username' indique que chaque utilisateur est identifié par cette propriété.

```yaml
# Use the newer login form authenticator instead of form_login
custom_authenticator: App\Security\LoginFormAuthenticator
```

Ce paramètre indique l'emplacement du code à exécuter lorsqu'un utilisateur essaie de s'authentifier.

```yaml
# Access control for URL patterns
access_control:
    - { path: ^/login, roles: PUBLIC_ACCESS }
    - { path: ^/register, roles: PUBLIC_ACCESS }
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }
```

Ces derniers paramètres commandent la stratégie globale de permissions de l'application. Les routes vers le formulaire de connexion (login) et d'enregistrement (register) sont d'accès public, le reste est d'accès privé avec une restriction spéciale pour l'administrateur pour toutes les routes qui commencent par */admin*.

```yaml
# Logout configuration
logout:
  path: app_logout
```

```
  target: homepage # Redirect after logout
```

Le nom de la route pour se déconnecter et la route vers laquelle rediriger l'utilisateur le cas échéant.

Le LoginFormAuthenticator

Ce fichier contient paramètres essentiels si l'on souhaite modifier le comportement du formulaire de connexion :

```
use TargetPathTrait;

public const LOGIN_ROUTE = 'app_login';

private UserRepository $userRepository;
private UrlGeneratorInterface $urlGenerator;

public function __construct(
  UserRepository $userRepository,
  UrlGeneratorInterface $urlGenerator
) {
  $this->userRepository = $userRepository;
  $this->urlGenerator = $urlGenerator;
}
```

Le nom de la route vers la page de connexion (ici 'app_login').

```
public function onAuthenticationSuccess(Request $request, TokenInterface
$token, string $firewallName): ?Response
{
  if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
     return new RedirectResponse($targetPath);
  }

  // Change this to your default redirect after successful login
  return new RedirectResponse($this->urlGenerator->generate('homepage'));
}
```

Le comportement à adopter en cas de connexion réussie.

Le SecurityController

```
#[Route('/login', name: 'app_login')]
public function login(AuthenticationUtils $authenticationUtils): Response
{
  // Get the login error if there is one
  $error = $authenticationUtils->getLastAuthenticationError();

  // Last username entered by the user
  $lastUsername = $authenticationUtils->getLastUsername();
```

```
    return $this->render('security/login.html.twig', [
        'last_username' => $lastUsername,
        'error' => $error,
    ]);
}
```

Cette route ‘app_login’ vient prendre en charge la connexion de l’utilisateur et renvoie le message d’erreur le cas échéant. Elle se charge de retourner le template *login.html.twig* qui contient le formulaire et affiche les erreurs le cas échéant

Mise à jour du projet

Pour tous les futurs développements, il est impératif de respecter quelques règles concernant les bonnes pratiques et la structure du projet.

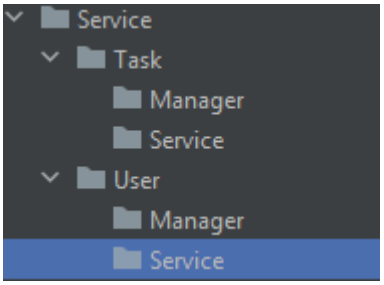
Bonnes pratiques

1. Utiliser l'outil de versionnement git pour garantir une bonne qualité de travail en équipe.
- Avant de commencer un nouveau développement, créer une nouvelle branche avec git checkout <nom de la branche> -b.
- Avant de terminer le développement et intégrer les modifications au projet, résoudre les conflits avec la branche principale en cas de mise à jour effectuée sur cette dernière par d'autres utilisateurs. Pour ce faire, intégrer d'abord les modifications de la branche principale sur votre branche de développement, et résoudre les éventuels conflits.
2. Assurer la couverture du projet par les tests. La couverture attendue est de 70%.
3. Respecter les règles de *peer-review* et les attentes en matière de messages de commit.

Structure du projet

S'agissant d'un projet symfony il est important de respecter la structure MVC définie par le framework. Au vu de la faible complexité de l'application, il est inutile de modifier cette approche. Un semblant de cohérence dans le dossier Service est appréciable.

1. Les nouveaux *Controllers* seront placés dans le dossier Controller, idem pour les Entity, etc.
2. La couche de Service sera placée dans un dossier Service :



- Service -> dossier portant le nom de l'entité 'business' (user, task, ...) -> dossier 'Manager' qui contient des Classes chargées d'appeler les différentes méthodes des Repositories Symfony ; dossier 'Service' qui contient les Classes chargées d'effectuer de la logique métier en appelant les Classes Managers. Ces Classes Services sont appelées dans les Controllers.
  💡Au vu de la faible complexité actuelle du projet, cette structure n'a pas encore été adoptée. Inutile de complexifier 3 méthodes.