DOCUMENTATION TECHNIQUE

Projet 8 : ToDo List

Richard PETIT

Sommaire

I/ Description du projet

1.1 Contexte

1.2 Technologies

II/ Installation du projet et contribution

2.1 Consulter les issues déjà existantes

2.2 Créer une issue

2.3 Cloner le projet

2.4 Créer une branche

2.5 Développer

2.6 La mise en place de tests

2.7 Soumettre la pull request

III/ Le fonctionnement de l’authentification

3 .1 L’authentification

3.1.1 Configurer l’authentification

3.1.2 Configurer le chargement des utilisateurs

3.1.3 Encoder le mot de passe utilisateur

3 .2 Autorisations et rôles des utilisateurs

I/ Description du projet

1.1 Contexte

Notre rôle ici est d'améliorer la qualité de l'application. La qualité est un concept qui englobe bon nombre de sujets : on parle souvent de qualité de code, mais également la qualité perçue par l'utilisateur de l'application ou encore la qualité perçue par les collaborateurs de l'entreprise, et enfin la qualité que perçue lorsqu'il nous faut travailler sur le projet.

Ainsi, pour ce projet , nous sommes en charge des tâches suivantes :

- l'implémentation de nouvelles fonctionnalités ;
- la correction de quelques anomalies ;
- et l'implémentation de tests automatisés.

1.2 Technologie

Le projet ToDoList utilise à ce jour Symfony 6.0.0 et PHP 8.0.13 qui sont, au moment de la création de cette documentation, les versions stables les plus récentes.

II/ Contribuer au projet

Pour contribuer au projet Il est demandé de respecter les normes de qualités telles que les PSR-4 (PHP Standards Recommendations).

Le projet est hébergé sur Github à l'adresse suivante : https://github.com/RichardPetit/TodoList .

Il est donc nécessaire de savoir comment utiliser Git pour contribuer au projet.

Pour en apprendre plus sur Git, vous pouvez vous référer à la documentation officielle :

https://docs.github.com/en/get-started

Le workflow (cheminement à suivre) pour contribuer au projet est le suivant.

2.1 Consulter les issues déjà existantes

Si vous avez constaté un problème sur l'application ou si vous avez une idée de nouvelle fonctionnalité pour l'améliorer,vérifiez que celui-ci n'a pas déjà été relevé et/ou résolu.

2.2 Créer une issue

Si le problème n'a pas été relevé ou si votre idée d'amélioration n'est pas présente, créez un nouvelle issue en précisant de manière claire et concise le but de cette issue (bug, nouvelle fonctionnalité etc.).

2.3 Cloner le projet

Il vous faudra ensuite cloner le projet en local grâce à la commande suivante :

git clone https://github.com/RichardPetit/TodoList.git

Rendez-vous dans le dossier contenant le projet cloné avec un terminal et lancez un composer install afin d'installer les différentes librairies utilisées dans le projet.

Configurez le fichier .env pour y paramétrer votre base de données.

```
DATABASE_URL="mysql://root:root@127.0.0.1:3306/db_name?serverVersion=5.7"
```

Puis exécutez les commandes suivante afin de créer la base de données :

- php bin/console doctrine:database:create .

- php bin/console doctrine:schema:update .

2.4 Créer une branche

Créez une nouvelle branche avec un nom explicite.

Exemple : Fix-problème-remarqué en cas de bug à réparer ;

Feature-nom-nouvelle-fonction pour une amélioration du projet.

Pour cela, assurez-vous d'être sur la branche master du projet, et utiliser la commande git checkout -b <Feature-branch-name> .

2.5 Développer

Développez ce que vous souhaitez mettre en place.

Sachez que votre travail ne sera pas accepté s'il ne respecte pas les règles de bonne pratiques en vigueurs.

- Les différentes PSR (1, 2, 4, 12) :
https://symfony.com/doc/current/contributing/code/standards.html

-Les conventions de code de Symfony :
https://symfony.com/doc/current/contributing/code/conventions.html

- Les bonnes pratiques de symony : https://symfony.com/doc/current/best_practices.html .

- La structure des fichiers mis en place et la logique de l'application.

2.6 La mise en place de tests

Une fois le développent terminé, vérifiez que votre travail n'ait pas cassé l'application.

Pour cela, il faut utiliser PHPUnit avec la commande : php bin/phpunit .

Si les résultats des tests relèvent des erreurs, corrigez les sans quoi votre contribution ne pourra être mise en place dans le projet.

2.7 Soumettre la pull request

Si vos tests ont été réalisé avec succès, vous pouvez pousser votre travail sur le repository.

Faites ensuite une demande de pull request afin de fusionner (merge) votre branche sur develop.

Seul le responsable du projet peut valider le merge.

III/ Le fonctionnement de l’authentification

3.1. L’authentification

Pour l’authentification, le fichier security.yml (chemin du fichier : app\config\packages\ security.yml) doit être configuré.

La partie la plus importante du fichier security.yml est la clé firewalls. On peut configurer autant de firewall que désiré.

3.1.1 Configurer l’authentification

L'application utilise le firewalls main.

Détails des clés utilisées :

- **pattern** : Une regex définissant les URL filtrées. Ici toutes les URL sont filtrés.
- **form_login** :
    - **login_path** : Le nom de la route utilisée pour se connecter.
    - **check_path** : Le nom de la route utilisée pour vérifier le couple utilisateur/mot de passe.
    - **enable_csrf** :
- **logout** : Autorise la déconnexion.

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        lazy: true
        provider: app_user_provider
        form_login:
            # "login" is the name of the route created previously
            login_path: login
            check_path: login
            enable_csrf: true
        logout:
            path: app_logout
```

3.1.2 Configurer le chargement des utilisateurs

Dans l'application les utilisateurs sont stokés en base de données,et nous utilisons Doctrine pour les récupérer. Nos utilistateurs sont représentés par l'entité USER, et nous retrouverons en BDD grâce au champ email.

Plusieurs providers peuvent-être configurés.

3.1.3 Encoder le mot de passe utilisateur

La clé password_hashers est utilisé pour définir l'encoder utilisé. Il a été définit pour l'algorithm la valeur « auto ». Cette valeur sélectionne automatiquement le meilleur algorithme de hachage possible, il ne fait donc pas référence à un algorithme spécifique et il changera à l'avenir pour l'entité User.

```
password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    App\Entity\User:
        algorithm: auto
```

Différents encoders peuvent-être utilisé sur différentes classes.

3.2 Autorisations et rôles des utilisateurs

Les autorisations sont définies dans la partie access_control.

Cela permet de décider si un utilisateur peut accéder à un URL (exemple restreindre l'accès des Url contenant « admin » aux utilisateurs ayant le ROLE_ADMIN).

```
access_control:
     - { path: ^/admin, roles: ROLE_ADMIN }
```

Les utilisateurs anonyme (non connectés) peuvent accéder à la page de login.

En revanche, toutes les autres URL, nécessitent au minimum le ROLE_USER.

Il est également possible d'afficher certaine données dans un fichier Twig seulement si l'utilisateur connecté est admin. Pour cela il faut utiliser la fonction is_granted() .

Exemple :

```
{% if (is_granted('ROLE_ADMIN')) %}
    <a href="{{ path('nom_de_la_route') }}" class="btn">Créer un utilisateur</a>
{% endif %}
```

Le processus d'autorisation nécessite deux étapes différentes :

- L'utilisateur reçoit un ensemble spécifique de rôles lors de la connexion (par exemple ROLE_ADMIN).

- Vous ajoutez du code pour qu'une ressource (par exemple une URL, un contrôleur) nécessite un "attribut" spécifique (le plus souvent un rôle comme ROLE_ADMIN) afin d'être accessible.

Lorsqu'un utilisateur se connecte, Symfony appelle la méthode getRoles() sur l'entité utilisateur pour en déterminer ses rôles. Dans notre classe User, les rôles sont un tableau qui est stocké en base de données, et chaque utilisateur se voit toujours attribuer au moins un rôle : ROLE_USER.

Deux rôles sont définis :

- ROLE_USER

- ROLE_ADMIN

Le rôle admin possède les droits du rôle User.