Documentation technique

Ce projet a été réalisé à partir du Framework PHP Symfony 5. Les données sont stockées sur une base MySQL.

I. Initialisation du projet

1. Installation du projet via GIT. Installation de GIT sur la machine. Ouvrir le git bash pour cloner le repository du projet : `git clone https://github.com/GATINEAU85/TODOLIST.git`
2. Se déplacer dans une invite de commande sur le dossier du projet : cd TODOLIST
3. Lancer l'installation du projet via composer : `composer install`
4. Initialiser les variables d'environnement présentes à la source du projet (fichier .env). Changer le DATABASE_URL pour le faire correspondre à votre environnement. `DATABASE_URL=mysql://user:password@host:port/database`
5. Installer la base de données test :
    - Pour créer la base de données qui est configurée dans le fichier de configuration de l'environnement : `php bin/console doctrine:database:create`
    - Pour visualiser le traitement SQL qui va servir à créer la structure de la base de données créée au préalable : `php bin/console doctrine:schema:update –-dump-sql`
    - Pour créer la structure de la base de données qui est configurée dans le fichier de configuration de l'environnement : `php bin/console doctrine:schema:update --force`
    - Pour charger le jeu de données test : `php bin/console doctrine:fixtures:load`
6. Lancer le serveur symfony : `symfony server:start`

II. Authentification

L'authentification se base sur le model User (utilisateur). Ce model est lui aussi stocké dans la base de données qui a été configurée au préalable. Le mot de passe y est hashé par l'algorithme automatique présent dans Symfony.

1. Installation

L'authentification des utilisateurs se base sur le bundle Symfony : SecurityBundle. Ce bundle est téléchargé par composer grâce à la commande suivante.

```
composer require symofny/security-bundle
```

Lors de l'installation composer va demander différentes informations pour initialiser l'authentification. La première question va concerner la classe que le bundle va devoir prendre pour s'authentifier. Cette classe pourra être changée par la suite lors de la configuration du bundle. Ici la classe utilisée est 'User'.

La seconde question concerne le stockage de données. Il est demandé si les données de l'utilisateur doivent être stockées en base. Dans notre cas, les données de l'utilisateur seront bien stockées en base oui.

La troisième question concerne quant à elle, le champ unique que doit prendre l'authentification pour identifier un utilisateur. Ce champ est très important puisque c'est celui-ci qui servira de base à l'authentification et qui permettra de ne pas avoir de doublons dans la base de données. Ici, c'est le champ email qui a été configuré.

Enfin la dernière question concerne le hashing du mot de passe. En effet, le mot de passe doit être crypté en base pour une question de sécurité. Il suffit donc de confirmer le cryptage du mot de passe. L'algorithme automatique sera alors paramétré.

2. Configuration

Après avoir installé le bundle SecurityBundle, Symfony nous donne accès à différentes configurations après son installation. Le principal fichier à configurer est le fichier App/config/packages/security.yaml. C'est dans ce fichier que vous allez pouvoir configurer le 'user provider'

Le 'fournisseur d'utilisateur' est important pour de multiples raisons. Il vous permettra par exemple de charger les données utilisateur en sessions, ou bien de simuler un utilisateur pour débugger plus rapidement. Il permet aussi d'utiliser des fonctions comme 'se souvenir de moi'. Le provider sera déjà configuré par les informations que vous avez rentré lors de l'installation.

```
security:
    providers:
        users:
            entity:
                # the class of the entity that represents users
                class: 'App\Entity\User'
                # the property to query by - e.g. username, email, etc
                property: 'username'
```

La seconde configuration concerne l'encodage de l'utilisateur. Ici l'encodage par défaut est resté. C'est l'algorithme qui sera utilisé pour encoder le mot de passe de la classe utilisateur. Ensuite pour l'encoder dans les controllers vous pourrez utiliser le service UserPasswordencoderInterface avant de sauvegarder l'utilisateur en base.

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

La troisième partie de ce fichier à configurer concerne l'authentification et le pare-feu. La sécurité de l'application se concentre essentiellement dans cette partie. C'est ici que va être détaillée la manière dont l'utilisateur va s'authentifier. Le pare-feu de développement n'est pas vraiment important c'est un pare-feu qui est fictif. Il n'a pour seul objectif de ne pas casser certaines autres classes de Symfony dev tools. Tous les URL passeront alors par le pare-feu principal.

C'est ce main firewalls qui va essayer de vous identifier dès votre arrivée sur le site. Si celui-ci ne vous trouve pas, il vous déterminera comme un utilisateur anonyme. Tout d'abord, le form_login vous permet d'identifier la méthode qui sera utilisée pour identifier un utilisateur. Le paramètre logout définira l'inverse de cette méthode puisqu'il sert à identifier la méthode/ le chemin de déconnexion d'un utilisateur. Enfin, l'un des paramètres le plus important est le guard. C'est ce paramètre qui va permettre d'identifier la classe qui sera en charge du processus d'authentification. Ici, cette classe est : App/Security/LoginFormAuthenticator.

Cette classe étend la clase Symfony : AbstractFormLoginAuthenticator et implémente l'interface : PasswordAuthenticatedInterface. C'est cette classe qui va mettre en relation

```
security:
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
            lazy: true
            provider: users
            pattern: ^/
            form_login:
                login_path: login
                always_use_default_target_path:  true
                default_target_path:  /
            logout:
                path: logout
            guard:
                authenticators:
                    - App\Security\LoginFormAuthenticator
```

Enfin, le dernier paramètre intéressant à configurer pour l'utilisation de l'application concerne le control des accès. Effectivement, différents rôles peuvent être attribués à chaque utilisateur. Un utilisateur non connecté sera considéré comme anonyme. Pour ceux qui sont connectés, différents rôles peuvent être attribués à la création du compte (« ROLE_USER », « ROLE_ADMIN » …). Ces rôles peuvent permettent aux utilisateurs d'accéder à différentes méthodes de l'application en fonction de leurs routes. Effectivement, les accès sont paramétrés en fonction du préfixage des routes. Par exemple, un utilisateur qui possède rôle admin pourra avoir accès à toutes les routes qui commencent par « /admin ».

```
security:
    access_control:
        - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/users, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin, roles: ROLE_ADMIN }
```

3. La classe SecurityController

Le SecurityController permet de définir les méthodes Login et Logout qui sont appelées par la configuration du SecurityBundle.