

- ToDoList -

Documentation technique



Table des matières

I. Installation.....	3
I.1. Cloner le dépôt.....	3
I.2. Installer les dépendances.....	3
I.3. Créer la base de données.....	3
II. Les fixtures.....	4
III. Le dossier SRC.....	5
III.1. Les entités.....	5
III.2. Les contrôleurs.....	5
III.3. Les formulaires.....	5
III.4. Les repositories.....	5
IV. L'authentification.....	6

I. Installation

Partons du principe que vous utilisez un ordinateur sur lequel l'application ToDoList n'est pas installée. La première étape va donc consister à installer le projet.

I.1. Cloner le dépôt

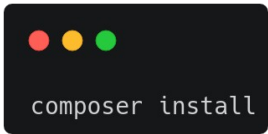
Nous allons cloner le dépôt GitHub. Ouvrez un terminal, placez-vous dans le dossier désiré et entrez la commande suivante :

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `git clone https://github.com/devperez/ToDoList.git` is displayed in a light green monospace font.

```
git clone https://github.com/devperez/ToDoList.git
```

I.2. Installer les dépendances

Une fois le clonage terminé, installons les dépendances. Une fois dans le répertoire du projet, entrez cette commande dans le terminal :

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `composer install` is displayed in a light green monospace font.

```
composer install
```

I.3. Créer la base de données

Passons à présent à la base de données. La première chose à faire est de renseigner le fichier `.env` qui se situe à la racine du projet. Modifiez la ligne

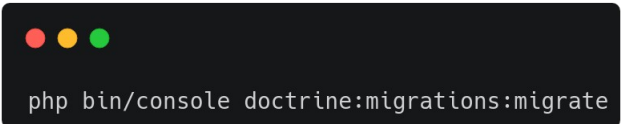
DATABASE_URL qui convient en ajoutant vos informations de connexion à la base de données.

Créez ensuite la base de données de l'application :



```
php bin/console doctrine:database:create
```

Enfin, exécutez les migrations afin d'avoir les tables nécessaires au bon fonctionnement de l'application :



```
php bin/console doctrine:migrations:migrate
```

II. Les fixtures

Pour l'instant, l'application est installée et la base de données est prête mais elle est vide. Dans le dossier src, vous trouverez le dossier DataFixtures. Ce dossier contient un fichier du même nom, DataFixtures.php.

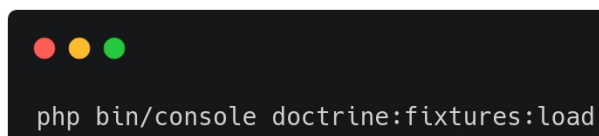
C'est ce fichier qui va vous permettre de peupler la base de données avec quelques utilisateurs et quelques tâches. Pour l'instant, ce fichier crée automatiquement 3 utilisateurs :

- testAdmin avec un rôle admin,
- testUser avec un rôle user,
- userAnonymous avec un rôle user.

Remarque : il faut savoir que le userAnonymous est important dans la logique de l'application car toutes les tâches créées sans connexion préalable sont rattachées à cet utilisateur.

Le fichier DataFixtures.php crée également 10 tâches qui seront automatiquement liées à l'utilisateur testUser et qui seront marquées comme non faites.

Lançons à présent ces fixtures :

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The command `php bin/console doctrine:fixtures:load` is entered in the terminal.

```
php bin/console doctrine:fixtures:load
```

Si vous vérifiez votre base de données, vous devriez à présent avoir 3 utilisateurs dans la table users et 10 tâches dans la table tasks.

III. Le dossier SRC

III.1. Les entités

La majeure partie du code qui nous intéresse se trouve dans le dossier src. Le premier dossier sur lequel nous allons nous arrêter est le dossier Entity. Ce dossier renferme deux fichiers, Task.php et User.php. Ces fichiers sont des classes php qui représentent les données dans la base de données. User.php représente par exemple une instance de la table users. Dans ces classes, vous retrouverez les propriétés, les méthodes et les relations propres à chaque entité.

III.2. Les contrôleurs

Dans le dossier Controller se trouvent les différents contrôleurs de l'application. Deux sont directement liés aux actions relatives aux tâches et aux utilisateurs (TaskController et UserController), DefaultController s'occupe quant à lui de retourner la page index et SecurityController s'occupe de l'authentification. Nous reviendrons sur ce point plus tard.

III.3. Les formulaires

Vous trouverez également dans le dossier src un dossier nommé Form. Ce dossier contient deux fichiers. Le premier, TaskType est le formulaire de création d'une tâche. Le second, UserType, est le formulaire de création d'un nouvel utilisateur.

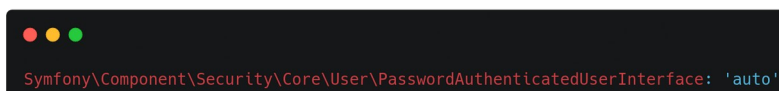
III.4. Les repositories

Le dossier Repository est le dernier du dossier src. Il comprend deux fichiers : TaskRepository.php et UserRepository.php. Les repositories permettent de gérer la persistance des données. Les repositories fournissent les méthodes pour créer, lire, mettre à jour et supprimer des entités. Toute cette logique est centralisée ici.

IV. L'authentification

Le fichier config/packages/security.yaml est essentiel au processus d'authentification. Il configure plusieurs aspects de la sécurité de l'application comme la gestion des mots de passe, les fournisseurs d'utilisateurs, les pare-feux et les contrôles d'accès.

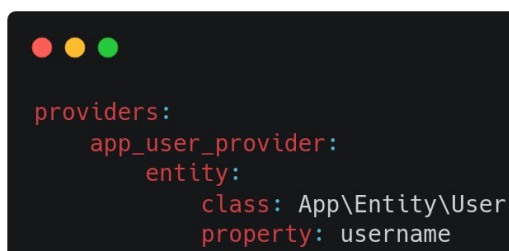
La section password_hashers définit comment les mots de passe utilisateurs sont hashés. Les utilisateurs qui implémentent l'interface PasswordAuthenticatedUserInterface verront leur mot de passe automatiquement hashé.



```
Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

La section providers est utilisée pour charger les utilisateurs dans le système de sécurité de symfony.

On utilise l'entité User comme source des utilisateurs. Property : username indique que nous utilisons cette propriété pour identifier les utilisateurs.



```
providers:
  app_user_provider:
    entity:
      class: App\Entity\User
      property: username
```

La section firewalls définit les niveaux de sécurité pour différentes parties de l'application.

Dev est un pare-feu pour l'environnement de développement, il est désactivé pour les routes spécifiées.

Main est le pare-feu principal de l'application. Le paramètre `lazy:true` permet de charger les utilisateurs en différé ce qui permet d'améliorer les performances. Avec `provider` on utilise le fournisseur défini précédemment. `Form_login` configure la connexion par le biais du formulaire. On a ensuite les deux urls (dans notre cas, il s'agit de deux fois la même), une pour la page de connexion et l'autre pour la vérification des tentatives de connexion.

Logout suit la même logique pour la déconnexion.

```
firewalls:
  dev:
    pattern: ^/(_(profiler|wdt)|css|images|js)/
    security: false
  main:
    lazy: true
    provider: app_user_provider
    form_login:
      login_path: app_login
      check_path: app_login
    logout:
      path: app_logout
```

La section `access_control` définit les règles d'accès aux différentes parties de l'application. Seuls les utilisateurs avec le rôle `ROLE_ADMIN` peuvent accéder aux routes commençant par `/admin`. Seuls les utilisateurs avec le rôle `ROLE_USER` peuvent accéder aux routes commençant par `/profile`.

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

Les requêtes d'authentification passent par la fonction `loginAction` du `SecurityController`. C'est cette fonction qui va afficher le formulaire de connexion et qui va également vérifier les données entrées par l'utilisateur et les comparer avec celles stockées en base de données.