

Projet ToDo & Co

Documentation technique

04/04/24

1 Sommaire

Table des matières

1 Sommaire.....	2
2 Présentation générale.....	3
2.1 Fonctionnalités.....	3
2.2 Contexte.....	3
3 La console.....	4
4 Authentification.....	4
4.1 Création d'un nouvel utilisateur.....	5
5 Base de données.....	5
5.1 Tables.....	5
5.1.1 User.....	5
5.1.2 Tasks.....	5
5.1.3 Doctrine.....	6
6 Architecture.....	6
6.1 La Config.....	6
6.1.1 config.security.provider.....	6
6.1.2 config.security.firewall.....	7
6.1.3 config.security.access_control.....	7
6.1.4 config.security.role_hierarchy.....	8
6.1.5 config.security.password_hasher.....	9
6.1.6 /package/routes/yaml.....	9
6.2 Les Contrôleurs.....	10
6.2.1 HomeController.....	10
6.2.2 UserController.....	10
6.3 Les entités (Entity).....	11
6.3.1 L'entité User.....	11
6.3.2 L'entité Task.....	11
6.3.3 La relation entre les entités User et Task.....	11
6.4 Services.....	12
6.5 Les Dépôts (Repositories).....	12
6.6 Les Fixtures.....	13
6.7 Les templates.....	13
7 Les tests unitaires.....	14
7.1 Exemple de tests unitaires.....	15
7.1.1 Couverture de tests.....	16
8 Conseils et Recommandations.....	17

2 Présentation générale

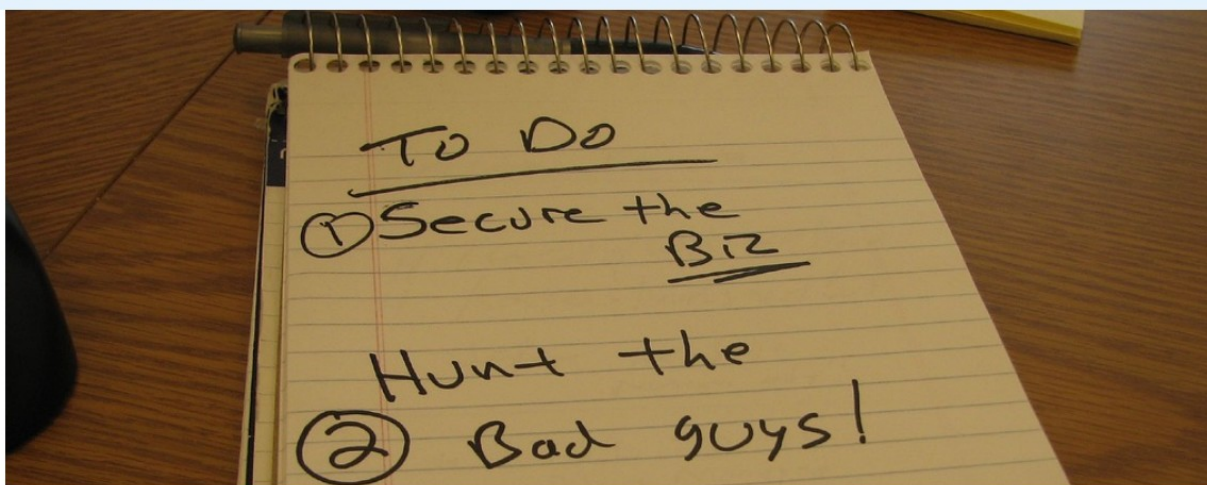
ToDo&Co est un logiciel de travail collaboratif qui permet d'exprimer et de signaler comme terminées des tâches.

Créer un utilisateur

Se connecter

Bienvenue sur Todo List, l'application vous permettant de gérer

To Do List app



Consulter la liste des tâches à faire

Consulter la liste des tâches terminées

2.1 Fonctionnalités

Elles peuvent être créées par un Admin ou par l'utilisateur lui-même, et eux seuls peuvent la supprimer.

La procédure est la suivante :

- Un Admin crée un utilisateur
- L'utilisateur ajoute une tâche
- L'utilisateur ou une autre personne signale cette tâche comme terminée
- L'utilisateur ou l'admin supprime cette tâche.

2.2 Contexte

Le logiciel présenté est une refonte du projet original, remis au goût du jour en terme de version de langage (Php-5.5.9 ⇒ Php-8.3.4), version de Symfony (3 ⇒ 7), et en terme de bonnes pratiques de développement (voir https://symfony.com/doc/current/best_practices.html).

3 La console

Le CLI (Command Line Interface) est essentiel à toutes les étapes de la conception du logiciel (voir <https://symfony-web.com/demarrer-avec-symfony/prerequis/comment-telecharger-et-installer-symfony-sur-windows/14>).

L'installation des composants (les Bundles) du dossier /Vendor se fait par la commande `symfony install`. Un `symfony update` permettra de mettre à jour les bundles.

La création ou la modification d'une entité se fera par `php bin/console make:entity`.

Symfony est un jeu de Bundles. Pour tout savoir sur les Bundles, suivre : <https://symfony.com/doc/current/bundles/configuration.html#using-the-bundle-extension>.

En terme général, la documentation de Symfony est un allié précieux lors du développement : <https://symfony.com/doc/current/index.html>.

En particulier, Symfony a son propre serveur de développement, qu'il faut initialisé via la commande : `symfony server:start -d`.



```
•
INFO A new Symfony CLI version is available (5.8.14, currently running 5.8.11).

If you installed the Symfony CLI via a package manager, updates are going to be automatic.
If not, upgrade by downloading the new version at https://github.com/symfony-cli/symfony-cli/releases
And replace the current binary (symfony.exe) by the new one.

[WARNING] The local web server is optimized for local development and MUST never be used in a production setup.

[OK] Web server listening
The Web server is using PHP CGI 8.3.4
https://127.0.0.1:8000

Stream the logs via symfony.exe server:log
PS C:\Users\dav\Documents\src\oc8>
```

4 Authentication

Please sign in

Username

Password

Sign in

Les deux utilisateurs servant aux tests, leurs mots de passe et leurs rôles sont :

Utilisateur	Mot de passe	Rôle
u1	d	ROLE_ADMIN
u2	d	ROLE_USER

4.1 Création d'un nouvel utilisateur

Seul un utilisateur ayant les droits `ROLE_ADMIN` peut créer des nouveaux utilisateurs.

5 Base de données

La base de données est spécifiée dans le `.env.local` (et le `.env.test`, qui est spécifique aux tests unitaires).

La variable `DATABASE_URL` renferme le type de bases de données (mysql), le nom de la base (oc8), de l'utilisateur (root) et son mot de passe (dev) :

```
DATABASE_URL="mysql://root:dev@127.0.0.1:3306/oc8"
```

La commande `php bin/console doctrine:database:create` permet d'installer la base une fois pour toutes, ensuite de quoi il ne reste qu'à migrer les entités *User* et *Task* en tant que tables :

```
php bin/console make:migration
```

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

5.1 Tables

5.1.1 User

La table *user* renferme les informations sur les utilisateurs :

- username
- password
- email
- roles

5.1.2 Tasks

La table *task* renferme les données sur les tâches :

- title
- content
- createdAt
- isDone

5.1.3 Doctrine

Doctrine est le moteur de gestion des bases de données de Symfony (voir <https://symfony.com/doc/current/reference/configuration/doctrine.html>).

Plus d'informations sur les tables dans le chapitre sur les entités (4.3).

6 Architecture

La structure des fichiers réponds aux principes [SOLID](https://fr.wikipedia.org/wiki/SOLID_(informatique)) (voir [https://fr.wikipedia.org/wiki/SOLID_\(informatique\)](https://fr.wikipedia.org/wiki/SOLID_(informatique))).

Il consiste en la séparation des classes selon leurs rôles.

Le dossier /test est une réplique du dossier /src où se trouve l'essentiel du logiciel.

Le dossier /Vendor n'est pas à copier sur le serveur, il est généré par la commande *composer install* d'après les indications du *composer.json*.

Pour l'installation, voir le fichier *README.md*.

6.1 La Config

Il n'y a rien à modifier dans ce répertoire hormis qu'il faut en comprendre les principes essentiels.

Les fichiers .yaml sont interprétés en Python et possèdent une hiérarchie qui découle des tabulations. Elles sont donc significatives, les modifier provoque des erreurs.

Le fichier le plus important est dans */config/packages/security.yaml*. On va l'examiner :

6.1.1 config.security.provider

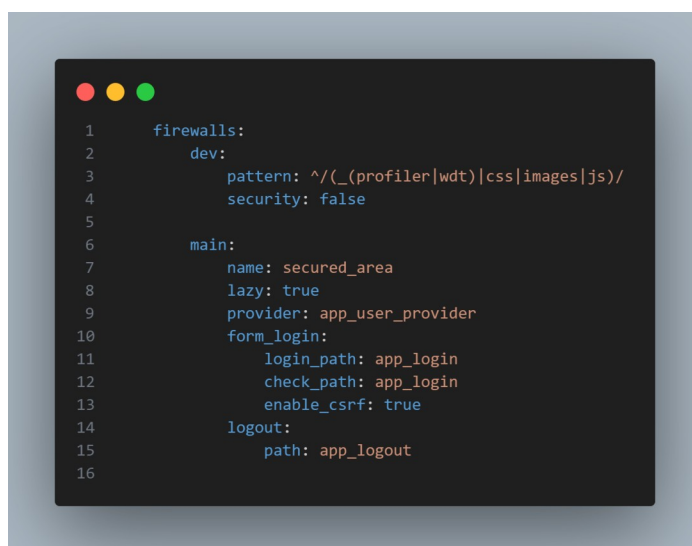
```
1 security:
2     # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
3     password_hashers:
4         Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
5     # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
6     providers:
7         # used to reload user from session & other features (e.g. switch_user)
8         app_user_provider:
9             entity:
10                 class: App\Entity\User
11                 property: username
```

Le *app_user_provider* est le moyen dont dispose Symfony pour retrouver un utilisateur logué dans la base de données (voir https://symfony.com/doc/current/security/user_providers.html).

Il est demandé de spécifier l'entité qui est utilisée pour stocker les utilisateurs, ici *\App\Entity\User*.

La propriété *username* est l'identifiant principal qui est demandé dans le formulaire de connexion. Cela aurait pu être l'email, comme cela se fait le plus souvent, mais on est resté fidèles au projet d'origine. Pour le changer il faut aussi le spécifier dans l'entité *User*.

6.1.2 config.security.firewall



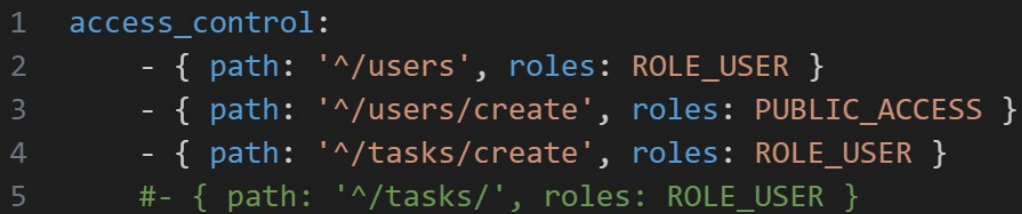
Les chemins des contrôleurs désignés pour identifier les formulaires de login et de logout est spécifié ici *app_login* et *app_logout*, qu'on trouve dans */src/Controller/securityController.php*.

Ce sont les moyens classiques de login de Symfony, c'est à dire que l'appel de ces contrôleurs fait implicitement référence au firewall, sans qu'il n'y ait besoin de rédiger de code au niveau des contrôleurs.

On aurait aussi bien pu utiliser un moyen personnalisé de s'authentifier, auquel cas il faut commenter ces lignes et décommenter le *custom_authenticator* et l'*entry_point*.

Dans ce cas il vaut mieux recommencer un *php bin/console make:auth* comme nous l'avons fait pour obtenir un processus de login normalisé et fonctionnel.

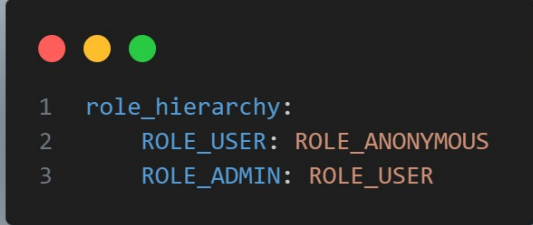
6.1.3 config.security.access_control



```
1 access_control:
2   - { path: '^/users', roles: ROLE_USER }
3   - { path: '^/users/create', roles: PUBLIC_ACCESS }
4   - { path: '^/tasks/create', roles: ROLE_USER }
5   #- { path: '^/tasks/', roles: ROLE_USER }
```

Le contrôle des accès permet d'allouer des routes à des rôles, et donc d'en interdire l'accès en cas de violation.

6.1.4 config.security.role_hierarchy



```
1 role_hierarchy:
2   ROLE_USER: ROLE_ANONYMOUS
3   ROLE_ADMIN: ROLE_USER
```

C'est ici que sont désignés les rôles supportés. Ils doivent correspondre aux rôles rendus possibles par le formulaire d'ajout d'utilisateur, situé dans `/src/Form/UserFormType.php`. L'entité *User* renvoie par défaut, si aucun rôle n'est assigné à un ancien utilisateur, le rôle *Anonymous*.

Pour lire ce code : `ROLE_ADMIN > ROLE_USER > ROLE_ANONYMOUS`.

6.1.5 config.security.password_hasher

```
1  when@test:
2      security:
3          password_hashers:
4              # By default, password hashers are resource intensive and take time. This is
5              # important to generate secure password hashes. In tests however, secure hashes
6              # are not important, waste resources and increase test times. The following
7              # reduces the work factor to the lowest possible values.
8              Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
9                  algorithm: auto
10                 cost: 4 # Lowest possible value for bcrypt
11                 time_cost: 3 # Lowest possible value for argon
12                 memory_cost: 10 # Lowest possible value for argon
```

Ici l'algorithme cryptographique utilisé pour hasher les mots de passes est laissé par défaut à celui qui est proposé par Symfony, actuellement *Bcrypt* (voir https://symfony.com/doc/current/best_practices.html#use-the-auto-password-hasher).

6.1.6 /package/routes/yaml

```
1  controllers:
2      resource:
3          path: ../src/Controller/
4          namespace: App\Controller
5      type: attribute
```

Le fichier routes.yaml sert à faire coïncider l'espaces de noms (namespace) des contrôleurs avec un répertoire spécifique. Les contrôleurs sont le point de départ de l'exploitation du logiciel.

6.2 Les Contrôleurs

6.2.1 HomeController

```
1 class HomeController extends AbstractController
2 {
3     #[Route('/', name: 'app_home')]
4     public function index(): Response
5     {
6         return $this->render('home/index.html.twig', [
7             'controller_name' => 'HomeController',
8         ]);
9     }
10 }
```

Un contrôleur typique, ici le *HomeController*, spécifie obligatoirement une route dans un attribut qui est présenté comme un commentaire de code, en réalité rendu opérationnel. C'est là qu'est spécifié le chemin d'accès mais aussi le nom de la route, afin de facilement la désigner dans les templates, et de ne pas avoir à les modifier si on change d'idée pour le chemin.

6.2.2 UserController

```
1 #[Route('/users/create', name: 'user_create')]
2 public function create(Request $request): Response
3 {
4     $user = new User();
5     $formUser = $this->createForm(UserFormType::class, $user);
6     $formUser->handleRequest($request);
7     if ($formUser->isSubmitted() && $formUser->isValid()) {
8         $this->userService->saveUser(
9             $user,
10            $formUser->get('_password')->getData(),
11            $formUser->get('roles')->getData()
12        );
13         $this->addFlash('success', 'L'utilisateur a bien été ajouté.');
```

Dans ce autre exemple on voit comment est traité un formulaire, qui appelle le contrôleur dont il est issu. Les éléments du formulaire appartiennent à une classe spécifique, *UserFormType*, et le traitement des données reçues du *Request* est délocalisé dans un service dédié.

Avant cela on peut vérifier la validité du formulaire ou si l'utilisateur a les droits, et après on peut le renvoyer vers une autre route.

6.3 Les entités (Entity)

Il n'y a que deux entités dans le logiciel ToDo&Co, et cela suffit déjà à engendrer toute la somme de dispositifs qui y sont associés.

L'entité utilisée par le Php Data Objet (PDO) des requêtes MySQL, (voir <https://www.php.net/manual/fr/book.pdo.php>) contient les colonnes des tables associées à ces objets (du même nom) et les getter/setters de ces données.

6.3.1 L'entité User

```
1 #[ORM\Entity(repositoryClass: UserRepository::class)]
2 #[ORM\Table(name: '`user`')]
3 #[ORM\UniqueConstraint(name: 'UNIQ_IDENTIFIER_USERNAME', fields: ['username'])]
4 #[UniqueEntity(fields: ['username'], message: 'There is already an account with this username')]
5 class User implements UserInterface, PasswordAuthenticatedUserInterface
6 {
```

L'entité *User* étant désignée par le *config.security* comme associée au login, elle spécifie le champ unique utilisé comme identifiant : le *username*.

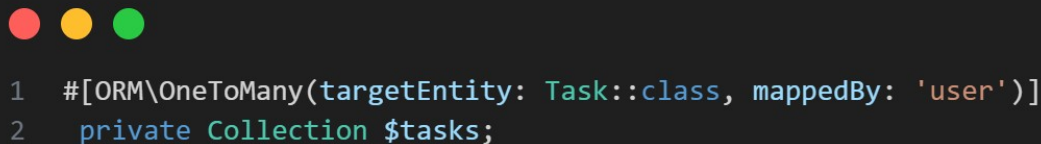
6.3.2 L'entité Task

L'entité *Task* contient les champs associés aux colonnes de la table *task*.

C'est ici qu'il faut en ajouter si on veut faire progresser le logiciel, avant de relancer une migration par les commandes *php bin/console doctrine:migrations:diff* et *php bin/console doctrine:fixtures:load*.

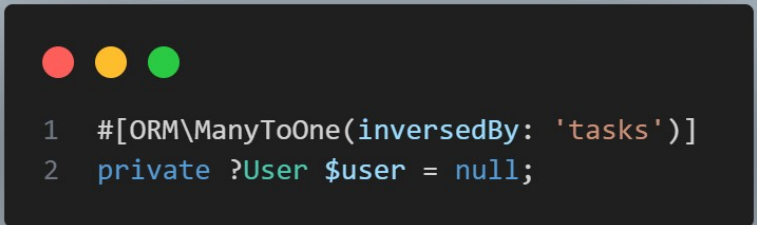
6.3.3 La relation entre les entités User et Task

Les tables *user* et *task* sont relationnées, dans la base de données par une clef étrangère (*foreign_key*), qui est retranscrite en dur dans les entités, comme suit :



```
1  #[ORM\OneToMany(targetEntity: Task::class, mappedBy: 'user')]  
2  private Collection $tasks;
```

Figure 1: Dans l'entité User, on désigne une Collection d'entités \$tasks associée à \$user dans User.



```
1  #[ORM\ManyToOne(inversedBy: 'tasks')]  
2  private ?User $user = null;
```

Figure 2: Dans l'entité Task, on désigne la variable \$user associée à la Collection \$tasks dans Task.

Le relation se fait sur le mode *ManyToOne*, depuis la perspective de *Task*.

6.4 Services

Les services sont destinés à être réutilisables dans n'importe quel autre service.

Ce sont des fonctions souvent préfixées par « get » qui servent à rallier les dépôts (Repository).

6.5 Les Dépôts (Repositories)

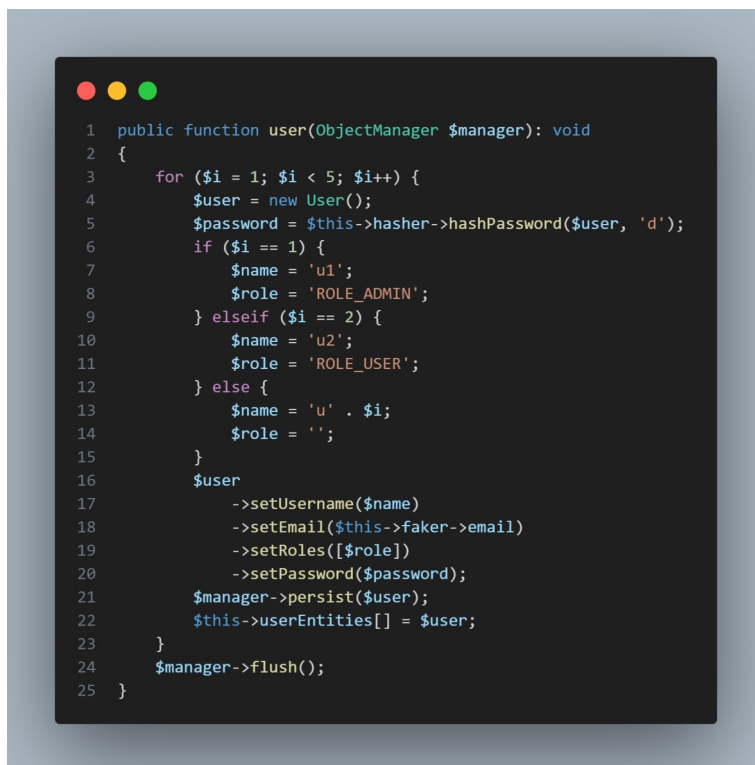
C'est ici qu'on lieu des actions de récupération de données, ici depuis MySQL.

En étendant la classe *SerciveEntityRepository*, le Repository obtient une logique de noms de classes, souvent préfixées par « find », dont un certain nombre qui sont proposées par défaut, ou d'autres qui sont « magiques » dans la mesure où elles comprennent ce qui est demandé, comme par exemple « *findTaskByUser* », qui n'a pas besoin d'être rédigée explicitement (voir

https://symfony.com/doc/current/introduction/from_flat_php_to_symfony.html#the-sample-application-in-symfony).

6.6 Les Fixtures

Les Fixtures servent à remplir les tables avec des données aléatoires qui servent pour les tests.

A screenshot of a code editor with a dark background and light-colored text. The code is in PHP and defines a public function named 'user' that takes an 'ObjectManager \$manager' as an argument. The function is designed to create five users in a loop. For each user, it generates a password using a hasher, assigns a role (either 'ROLE_ADMIN' for the first user or 'ROLE_USER' for the others), and sets a username. The username for the first user is 'u1', for the second is 'u2', and for the others, it's 'u' followed by the loop index. The user object is then persisted using the manager's 'persist' method, and the 'userEntities' array is updated. Finally, the manager's 'flush' method is called to save the data.

```
1 public function user(ObjectManager $manager): void
2 {
3     for ($i = 1; $i < 5; $i++) {
4         $user = new User();
5         $password = $this->hasher->hashPassword($user, 'd');
6         if ($i == 1) {
7             $name = 'u1';
8             $role = 'ROLE_ADMIN';
9         } elseif ($i == 2) {
10            $name = 'u2';
11            $role = 'ROLE_USER';
12        } else {
13            $name = 'u' . $i;
14            $role = '';
15        }
16        $user
17            ->setUsername($name)
18            ->setEmail($this->faker->email)
19            ->setRoles([$role])
20            ->setPassword($password);
21        $manager->persist($user);
22        $this->userEntities[] = $user;
23    }
24    $manager->flush();
25 }
```

Lors de l'exécution d'un `php bin/console doctrine:fixtures:load`, l'app `DataFixtures/AppFixtures` ira remplir les tables `User` et `Task` avec des données aléatoires.

Nous avons fait en sorte de simuler la situation initiale, où les tâches ne sont pas reliées à des utilisateurs, et où ceux-ci n'ont pas encore de rôles. Dans ce cas seul l'Admin pourra les effacer.

6.7 Les templates

Le moteur de templates est confié à Twig et il n'a pas changé depuis la V1.

Les variables envoyées à Twig sont des objets qui sont ensuite automatiquement sérialisés.

Si on veut filtrer des données en fonction du rôle utilisateur, dans le template cela se transcrit en comparant le rôle de l'objet parsé avec le rôle de l'utilisateur en cours, comme ici dans le template `userlist` :

```
1 {% if is_granted('ROLE_ADMIN') or (user.username == currentUser) %}  
2     <a href="{{ path('user_edit', {'id' : user.id}) }}" class="btn btn-success btn-sm">Edit</a>  
3 {% endif %}
```

La variable `currentUser` aura été définie au préalable dans le template d'après le `user.username`.

7 Les tests unitaires

Le répertoire à la racine `/tests` renferme la même structure de dossiers et de fichiers que dans le répertoire `/src`.

Les tests unitaires permettent d'avoir un suivi sur les retours attendus de chacune de ces fonctions dans l'ensemble des cas de figures rendus possibles, ou non, par le logiciel.

- `/tests`
 - `/Controller`
 - `/HomeControllerTest.php`
 - `/TaskControllerTest.php`
 - `/UserControllerTest.php`
 - `/Entity`
 - `/TaskTest.php`
 - `/UserTest.php`
 - `/Repository`
 - `/TaskRepositoryTest.php`
 - `/UserRepositoryTest.php`
 - `/Services`
 - `/UserRepositoryTest.php`
 - `/UserServiceTest.php`

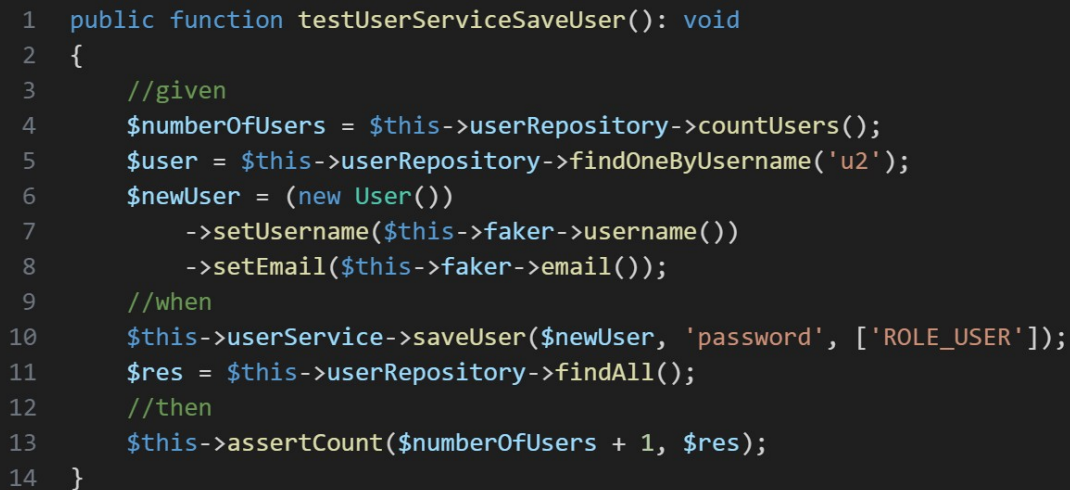
7.1 Exemple de tests unitaires

Les tests sont rédigés selon la forme *Given/When/Then*, où on annonce les données, les conditions, et les résultats attendus.

Les assertions sont les résultats attendus, elles peuvent donc être positives ou négatives, s'il y a une erreur. (voir catalogue des types d'assertions possibles :

<https://docs.phpunit.de/en/10.5/assertions.html#assertisobject>).

Dans cet exemple, on crée un utilisateur, on l'enregistre dans la base de données de tests (oc8_test, fournie par le `.env.test`), et on s'attend à ce que le nombre total d'utilisateurs ait été incrémenté.



```
1 public function testUserServiceSaveUser(): void
2 {
3     //given
4     $numberOfUsers = $this->userRepository->countUsers();
5     $user = $this->userRepository->findOneByUsername('u2');
6     $newUser = (new User())
7         ->setUsername($this->faker->username())
8         ->setEmail($this->faker->email());
9     //when
10    $this->userService->saveUser($newUser, 'password', ['ROLE_USER']);
11    $res = $this->userRepository->findAll();
12    //then
13    $this->assertCount($numberOfUsers + 1, $res);
14 }
```

Dans ce autre exemple, plus complexe, on déplace le curseur du Client, initialement fixé sur la page où on efface une tâche, et ensuite amené à suivre la redirection attendue par *followRedirect()*.

On vérifie que le nombre de tâches existantes + 1 est égal au nombre de tâches initialement découvertes.

On aura vérifié que l'utilisateur faisant cette opération a bien le rôle `ROLE_ADMIN` (utilisateur « u1 »).

On aura assuré qu'il est bien identifié grâce à la fonction *loginUser*.

```

1  public function testTaskControllerDeleteTaskByAdmin(): void
2  {
3      $numberOfTasks = $this->taskRepository->countTasks();
4      $user = $this->userRepository->findOneByUsername('u1');
5      $user2 = $this->userRepository->findOneByUsername('u2');
6      $taskEntity = $this->taskRepository->findOneByUser($user2);
7      $this->client->loginUser($user, 'secured_area');
8      $crawler = $this->client->request('GET', '/tasks/' . $taskEntity->getId() . '/delete');
9      $this->assertResponseRedirects('/tasks');
10     $crawler = $this->client->followRedirect();
11     $this->assertAnySelectorTextSame('div', 'list_tasks');
12     $newNumberOfTasks = $this->taskRepository->countTasks();
13     $this->assertEquals($numberOfTasks, $newNumberOfTasks + 1);
14 }

```

De cette manière, on procède, pas à pas, en testant toutes les éventualités, aux tests unitaires de l'ensemble du logiciel.






















Les fonctions précédentes peuvent être testées via les commandes :

`vendor/bin/phpunit --filter=testUserServiceSaveUser`

`vendor/bin/phpunit --filter=testTaskControllerDeleteTaskByAdmin.`

7.1.1 Couverture de tests

La commande `vendor/bin/phpunit --coverage-html public/test-coverage` permet d'engendrer le rapport de couverture des tests, qui se situe dans le répertoire `/public/test-coverage` :

	Code Coverage							
	Lines		Functions and Methods			Classes and Traits		
Total		75.87%	283 / 373		78.87%	56 / 71		35.71% 5 / 14
■ Controller		57.29%	55 / 96		64.29%	9 / 14		40.00% 2 / 5
■ DataFixtures		0.00%	0 / 33		0.00%	0 / 4		0.00% 0 / 1
■ Entity		88.37%	38 / 43		92.31%	24 / 26		50.00% 1 / 2
■ Form		99.29%	139 / 140		83.33%	5 / 6		50.00% 1 / 2
■ Repository		85.71%	30 / 35		90.91%	10 / 11		50.00% 1 / 2
■ Service		80.77%	21 / 26		80.00%	8 / 10		0.00% 0 / 2
Kernel.php		n/a	0 / 0		n/a	0 / 0		n/a 0 / 0

Ici, 75 % du logiciel est couvert par les tests, soit sa principale partie.

8 Conseils et Recommandations

N'importe qui peut signaler une tâche comme terminée, autre que son auteur ou l'admin.

L'organisation interne étant laissée à l'utilisateur (le client), il devrait y avoir le choix des modes de fonctionnement sur ce point, dans un tableau de paramétrages.

De même, les tâches peuvent avoir à être résolues partiellement, ou de manière collaborative. Dans ce cas plusieurs personnes seraient assignées à une même tâche.

Il n'est pas possible de s'enregistrer comme utilisateur, seul l'Admin peut ajouter un nouvel utilisateur. Peut-être faudra-t-il permettre à l'utilisateur de le faire.

La mise en page est laissée « dans son jus ». Elle gagnera à être refaite également, de façon à ce que :

- le bouton « Ajouter une tâche » ne vienne pas désaligner les blocs.
- les icônes de statut ☒ et ☐ devraient être placés de façon à ne pas perturber la lecture du titre, qui peut être long.

Sur la page d'accueil, un bouton inutile ne portant vers aucun lien stipule « Voir les tâches accomplies ». Il vaudrait mieux supprimer ce bouton et permettre de faire un tri sur la page de la liste des tâches elles-mêmes.

Les tâches devraient avoir une date de création et une date de résolution. Elles pourraient également être situées sur un calendrier avec une deadline.

Enfin, aucun lien ne permet de tomber sur la page de gestion des utilisateurs (/users), qu'il faut taper soi-même dans la barre d'adresse. On retombe sur cette page seulement après la création d'un nouvel utilisateur.

Toutes ces modifications ont été rendues possibles à faire facilement par l'upgrade qui est proposé.

