Documentation Technique

To&Co

Les Fichiers à modifier avec les explications

Nous utilisons le Framework Symfony (3.1). Cette version a été migrée vers la version 3.4

Principalement les dossiers qui nous concernent sont App, Src et Tests.

Dossier App

Pour commencer, quand il s'agit d'installer des bundles, à la racine de ce dossier, il y'a le fichier *AppKernel.php* qui nous intéresse car c'est dans ce fichier qu'on ajoute l'instance du bundle après l'installation.

```
if (in_array($this->getEnvironment(), ['dev', 'test'], true)) {
    $bundles[] = new Symfony\Bundle\DebugBundle\DebugBundle();
    $bundles[] = new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle();
    $bundles[] = new Sensio\Bundle\DistributionBundle\SensioDistributionBundle();
    $bundles[] = new Sensio\Bundle\GeneratorBundle\SensioGeneratorBundle();
    $bundles[] = new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle();
```

On a le dossier config pour la partie sécurité principalement Le fichier *security.yml* nous concerne pour l'authentification et limiter les rôles. C'est dans ce fichier qu'on donne l'accès à l'admin et user comme pour avoir accès a la gestion membre pour l'admin .

```
access_control:
    - { path: ^/users/create, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/tasks, roles: ROLE_USER }
    - { path: ^/, roles: IS_AUTHENTICATED_ANONYMOUSLY }
```

Et pour finir dans ce dossier il faut aussi toucher le dossier Ressources/views pour la partie vue afin de changer et ajouter les variables twig en fonction de l'appel du contrôleur.

```
        <div>
            <form action="{{ path('task_toggle', {'id' : task.id }) }}">
                <button class="btn btn-success btn-sm pull-right">
                    {% if not task.isDone %}Marquer comme faite{% else %}Marquer non terminée{% endif %}
                </button>
            </form>
            {% if task.user == app.user or (task.user is null and is_granted('ROLE_ADMIN')) %}
                <form action="{{ path('task_delete', {'id' : task.id }) }}">
                    <button class="btn btn-danger btn-sm pull-right">Supprimer</button>
                </form>
            {% endif %}
        </div>
    </div>
</div>
```

Dossier Src

C'est dans ce dossier qu'on trouvera l'essentiel du projet avec les classes, *controller, Form Repository , Entity* . Ce sont des fichiers qui représentent le moteur du projet,

Entity : pour ajouter la relation entre task et user, ajouter des propriétés en plus.

```php
/**
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\User", inversedBy="tasks")
 * @ORM\JoinColumn(nullable=true)
 */
private $user;
```

Controlleur : faire appel à la route de chaque fonction du contrôleur, construire une action puis le faire appel à la vue, permettre de modifier la fonction delete pour que seulement à la possibilité de supprimer les taches anonymes .d'exécuter les actions en fonction de l'utilisateur.

```php
/**
 * @Route("/tasks/{id}/delete", name="task_delete")
 */
public function deleteTaskAction(Task $task)
{
    if (($task->getUser() !== null && $task->getUser() !== $this->getUser()) ||
        ($task->getUser() === null && !$this->get('security.authorization_checker')->isGranted('ROLE_ADMIN'))) {
        throw $this->createAccessDeniedException('You cannot access this page!');
    }
    $entityManager = $this->getDoctrine()->getManager();
    if ($task->getUser()) {
        $this->getUser()->removeTask($task);
    }
    $entityManager->remove($task);
    $entityManager->flush();
    $this->addFlash('success', 'La tâche a bien été supprimée.');
    return $this->redirectToRoute('task_list');
}
}
```

Form : ajouté les propriétés de l'entité pour les faire correspondre aux champs à remplir.

Pour finir le dossier Test

Afin de mettre en place des tests comme unitaire et fonctionnelle pour tester les fichiers présentes dans le dossier src avec *entity, form ,controller* , c'est dans ce dossier que tout ce passe avec notamment les mêmes noms des fichiers en ajoutant test afin de simplifier et rendre compréhensif les tests.

```php
class DefaultControllerTest extends WebTestCase
{
    <?php
    public function setUp()
    {@inheritDoc}
    public function setUp()
    {
        parent::setUp();

        $this->loadFixturesForTests();
    }

    public function testIndex()
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/');

        $this->assertEquals(302, $client->getResponse()->getStatusCode());
        $this->assertContains('login', $client->getResponse()->getContent());

    }
```