

OPENCLASSROOMS

TODO LIST

1
2
3



DOCUMENTATION TECHNIQUE

TODO & CO

APPLICATION

13/07/2022

Rédigé par Siaka MANSALY

TABLE DES MATIERES

A.	Introduction	1
B.	Utilisateurs	2
1)	Doctrine.....	2
2)	Entités	2
3)	Méthodes	3
4)	Migrations	3
5)	Configuration.....	3
C.	Authentification.....	5
1)	Formulaire de connexion	5
2)	Controller	5
3)	Vue.....	6
4)	Processus d'authentification	6
D.	Autorisations.....	8
E.	Tests	9
F.	Documentation.....	10
G.	Aller plus loin	10

A. INTRODUCTION

Cette documentation est à destination des développeurs de l'équipe ToDo & Co.

L'application est développée sous le Framework Symfony version 5.4. Ce Framework à l'avantage d'être très bien documenté et dispose d'une grande communauté.

Documentation Symfony : <https://symfony.com/doc/5.4/index.html>

L'application est disponible sur le repository suivant :
<https://github.com/siakamansaly/Audit-and-Improve-Symfony-App>

Vous y trouverez également les instructions d'installation et de contribution au projet ainsi que la documentation.

Structuration des fichiers :

- Les fichiers **PHP** se situent dans le répertoire « **src** »
- Les fichiers de **configurations** se situent dans le répertoire « **config** »
- Les fichiers **Twig** se situent dans le répertoire « **templates** »
- Les fichiers de **tests** se situent dans le répertoire « **tests** »
- Les fichiers **publics** (CSS, JS et autres) se situent dans le répertoire « **public** »
- Les **dépendances** de Symfony installés via Composer se situent dans le répertoire « **vendor** »
- Les fichiers **d'environnement** « .env » sont situés à la racine du projet

Les dépendances de Symfony sont installées via le composant **Composer** qui permet le chargement automatique des dépendances et espaces de noms.

Exemple répertoire « src » situé dans l'espace de nom « App »

```
"autoload": {  
    "psr-4": {  
        "App\\": "src/"  
    }  
},  
"autoload-dev": {  
    "psr-4": {  
        "App\\Tests\\": "tests/"  
    }  
},
```

Retrouvez toute la **documentation PHP** dans le dossier « **PHPDoc** » du répertoire « **docs** ».

<https://github.com/siakamansaly/Audit-and-Improve-Symfony-App/tree/main/docs>

B. UTILISATEURS

1) Doctrine

Le Framework Symfony intègre la dépendance « **Doctrine** ». Doctrine est un ORM (Object–relational mapping) qui a pour rôle de **faire le lien entre les objets et la base de données**.

Les **annotations** Doctrine définissent les différents mappages ou contraintes liées à la **base de données**.

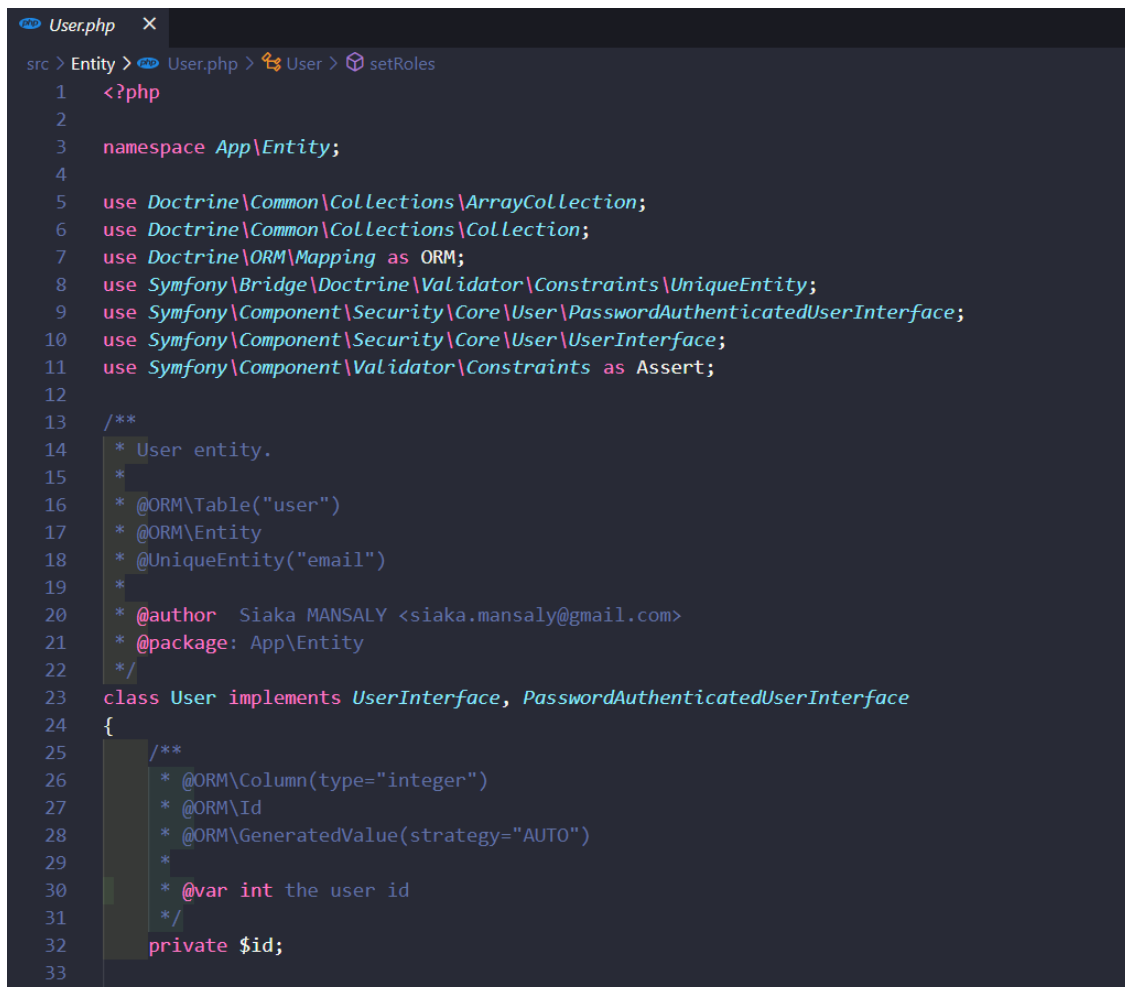
2) Entités

Les différents fichiers représentant les entités sont stockés dans le dossier :
« **src** » -> « **Entity** »

Les utilisateurs de l'application ToDo & Co sont représentés par l'entité « **User** ».

Cette entité implémente les classes « **UserInterface** » et « **PasswordAuthenticatedUserInterface** » qui sont des contrats permettant de sécuriser la connexion des utilisateurs.

Classe « *User* »



```

User.php x
src > Entity > User.php > User > setRoles
1  <?php
2
3  namespace App\Entity;
4
5  use Doctrine\Common\Collections\ArrayCollection;
6  use Doctrine\Common\Collections\Collection;
7  use Doctrine\ORM\Mapping as ORM;
8  use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
9  use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
10 use Symfony\Component\Security\Core\User\UserInterface;
11 use Symfony\Component\Validator\Constraints as Assert;
12
13 /**
14  * User entity.
15  *
16  * @ORM\Table("user")
17  * @ORM\Entity
18  * @UniqueEntity("email")
19  *
20  * @author Siaka MANSALY <siaka.mansaly@gmail.com>
21  * @package: App\Entity
22  */
23 class User implements UserInterface, PasswordAuthenticatedUserInterface
24 {
25     /**
26      * @ORM\Column(type="integer")
27      * @ORM\Id
28      * @ORM\GeneratedValue(strategy="AUTO")
29      *
30      * @var int the user id
31      */
32     private $id;
33
34     /**

```

3) Méthodes

La classe est également composée de **méthodes** permettant **d'interagir** (Getters and Setters) avec les **propriétés** de l'objet User.

Exemple Fonction « getRoles » de la classe « User »

```
/**
 * Get the user roles.
 *
 * @return array<mixed> the user roles
 */
public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';

    return array_unique($roles);
}
```

4) Migrations

Après la modification d'annotations Doctrine d'une entité ou l'ajout de nouvelles propriétés, vous devez exécuter sur votre terminal les commandes ci-dessous :

- `php bin/console make:migration`
Cette commande génère un fichier de migration avec les modifications SQL de la base de données
- `php bin/console doctrine:migrations:migrate`
Cette commande exécute le script de migration généré dans la base de données.

Ces opérations sont nécessaires afin que la base de données soit toujours synchronisé avec les entités.

5) Configuration

Pour **mettre en place le système d'authentification** et donc **sécuriser des parties de l'application**, on a défini les règles de connexion du pare-feu de Symfony dans le fichier de configuration « **security.yaml** » situé dans le répertoire « **packages** » du dossier de configuration.

Fichier « security.yaml » de l'application

```
config > packages > security.yaml > {} security > {} firewalls
1 security:
2   enable_authenticator_manager: true
3   # https://symfony.com/doc/current/security.html#registering-the-user-hashing
4   password_hashers:
5     #Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface
6     App\Entity\User:
7       algorithm: bcrypt
8
9   # https://symfony.com/doc/current/security.html#loading-the-user-the-user-pr
10  providers:
11    users_in_memory: { memory: null }
12    app_user_provider:
13      entity:
14        class: App\Entity\User
15        property: username
16  firewalls:
17    dev:
18      pattern: ^/(_(profiler|wdt)|css|images|js)/
19      security: false
20    main:
21      lazy: true
22      provider: app_user_provider
23      pattern: ^/
24      custom_authenticator: App\Security\SecurityControllerAuthenticator
25      logout:
26        path: app_logout
```

Description du fichier de configuration « security.yaml » :

- Définition des règles de cryptage du mode de passe de l'utilisateur (Ligne 4 à 7)
 - Algorithme de cryptage : « bcrypt »
- Définition de la classe et de la propriété de l'utilisateur permettant la connexion (Ligne 12 à 15)
 - Classe : « User »
 - Propriété : « username »
- Définition du mécanisme d'authentification et des contraintes associés (Ligne 20 à 24)
 - Utilisateur anonyme autorisé
 - Sélection du provider défini
 - Définition du périmètre d'action du firewall (pattern)
 - Sélection du fichier de contrôle de l'authentification
- Définition du mécanisme de déconnexion (Ligne 25 à 26)
 - Sélection du chemin de la méthode de déconnexion

Les **utilisateurs non-authentifiés** sont automatiquement **redirigés** vers la page de connexion lorsqu'ils souhaitent accéder à une partie sécurisée de l'application.

Il est possible de modifier le **contrôle d'accès des routes** directement dans ce fichier sous le paramètre « **access_control** ».

A savoir, les accès peuvent également être gérés directement dans les contrôleurs.

*Fichier « security.yaml »
seule la route « /login » est accessible aux utilisateurs non-authentifiés*

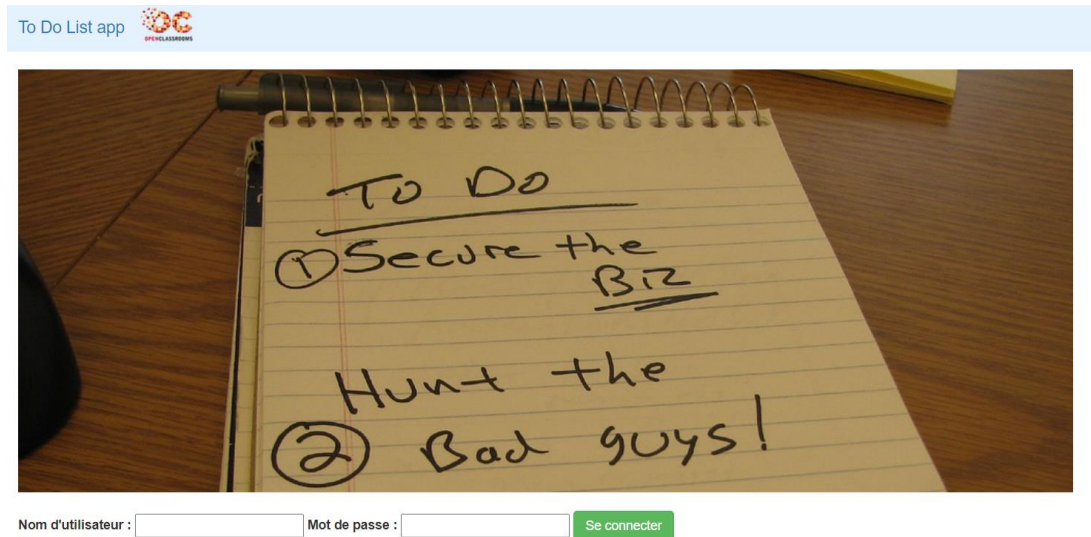
```
access_control:
  - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/users, roles: ROLE_ADMIN }
  - { path: ^/, roles: ROLE_USER }
```

C. AUTHENTIFICATION

1) Formulaire de connexion

Pour s'authentifier à l'application, la route est la suivante : **/login**

Page de connexion



2) Controller

La méthode « **login** » du fichier « **SecurityController.php** » permet de contrôler l'affichage de la **page de connexion** « **/login** ».

Si un **utilisateur déjà connecté** souhaite accéder à cette route, il est automatiquement **redirigé** vers la page d'accueil.

Ce Controller renvoie également les éventuelles erreurs et informations de connexion (lastUsername) à la vue.

Classe « SecurityController »

```
class SecurityController extends AbstractController
{
    /**
     * Login page.
     */
    /**
     * @Route("/login", name="login")
     */
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        if ($this->getUser()) {
            return $this->redirectToRoute('homepage');
        }

        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security/login.html.twig', ['last_username' => $lastUsername, 'error' => $error]);
    }
}
```

3) Vue

Ce formulaire a été généré par la vue Twig « **login.html.twig** » se situant dans le répertoire « **security** » des templates.

Fichier « *login.html.twig* »

```

powerShell  login.html.twig x
templates > security > login.html.twig > form > input#password
1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4  {% if error %}
5  <div class="alert alert-danger" role="alert">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
6  {% endif %}
7
8  <form method="post">
9      <label for="username">Nom d'utilisateur :</label>
10     <input type="text" id="username" name="_username" value="{{ last_username }}" />
11
12     <label for="password">Mot de passe :</label>
13     <input type="password" id="password" name="_password" />
14     <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}" />
15
16     <button class="btn btn-success" type="submit">Se connecter</button>
17 </form>
18 {% endblock %}

```

Comme on peut le voir en ligne 1, la vue étend le modèle de base de l'application nommé « **base.html.twig** ».

Dans le corps (« **block body** » ligne 3 à 18), la vue génère un **formulaire de connexion**.

Ce formulaire est protégé par un **token CSRF** permettant de contrer les attaques des pirates.

Toutes les **erreurs** liées à ce formulaire seront affichées au-dessus du formulaire (ligne 5).

4) Processus d'authentification

Une fois le formulaire soumis, le **pare-feu** intercepte l'évènement et exécute le provider personnalisé situé dans le fichier « **SecurityControllerAuthenticator.php** » du répertoire « **Security** » des dossiers PHP.

Fichier « *security.yaml* »

Provider personnalisé paramétré en ligne 24

```

16  firewalls:
17      dev:
18          pattern: ^/(_(profiler|wdt)|css|images|js)/
19          security: false
20      main:
21          lazy: true
22          provider: app_user_provider
23          pattern: ^/
24          custom_authenticator: App\Security\SecurityControllerAuthenticator
25          logout:
26              path: app_logout

```


Pour information, la classe « **SecurityControllerAuthenticator** » étend la classe « **AbstractLoginFormAuthenticator** » qui permet de faciliter l'authentification.

Classe « *SecurityControllerAuthenticator* »

```

28 class SecurityControllerAuthenticator extends AbstractLoginFormAuthenticator
29 {
30     use TargetPathTrait;
31
32     public const LOGIN_ROUTE = 'login';
33
34     private UrlGeneratorInterface $urlGenerator;
35
36     /** ...
37     public function __construct(UrlGeneratorInterface $urlGenerator)
38     {
39         ...
40     }
41
42     /** ...
43     public function authenticate(Request $request): Passport
44     {
45         ...
46     }
47
48     /** ...
49     public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?
50     Response
51     {
52         ...
53     }
54
55     /** ...
56     protected function getLoginUrl(Request $request): string
57     {
58         ...
59     }
60 }

```

Ensuite, la méthode « **authenticate** » récupère les **données** (username, password et csrf_token) de la requête http du formulaire soumis puis effectue les **vérifications**.

Méthode « *authenticate* » de la classe « *SecurityControllerAuthenticator* »

```

56 public function authenticate(Request $request): Passport
57 {
58     /** @var string $username */
59     $username = $request->request->get('_username', '');
60
61     $request->getSession()->set(Security::LAST_USERNAME, $username);
62
63     /** @var string $password */
64     $password = $request->request->get('_password', '');
65
66     /** @var string $csrf */
67     $csrf = $request->request->get('_csrf_token', '');
68
69     return new Passport(
70         new UserBadge($username),
71         new PasswordCredentials($password),
72         [
73             new CsrfTokenBadge('authenticate', $csrf),
74         ]
75     );
76 }

```

En cas de **succès** de connexion, la méthode « **onAuthenticationSuccess** » est exécutée et redirige l'utilisateur vers la page demandée ou vers la page d'accueil.

Méthode « *onAuthenticationSuccess* » de la classe « *SecurityControllerAuthenticator* »

```

86 public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?
87 Response
88 {
89     if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
90         return new RedirectResponse($targetPath);
91     }
92
93     return new RedirectResponse($this->urlGenerator->generate('homepage'));
94 }

```

En cas **d'échec** de connexion, l'utilisateur est alors redirigé à la page de connexion avec un récapitulatif des **erreurs**.

D. AUTORISATIONS

Les **autorisations** sont implémentées dans des **Voters**. Ils permettent de centraliser toute la logique des autorisations, puis de les réutiliser à de nombreux endroits.

Ces Voters sont situés dans le répertoire « **Security** » des dossiers PHP.

Classe « TaskVoter »

```

20 class TaskVoter extends Voter
21 {
22     public const DELETE = 'TASK_DELETE';
23     private Security $security;
24
25     /** ...
28     public function __construct(Security $security)
29     { ...
31     }
32
33     /** ...
36     protected function supports(string $attribute, $subject): bool
37     { ...
39     }
40
41     /** ...
46     protected function voteOnAttribute(string $attribute, $subject, TokenInterface $token): bool
47     { ...
49     }
50
51     /** ...
54     public function canDelete(Task $task, UserInterface $user): bool
55     { ...
57     }
58 }

```

La classe « **TaskVoter** » dans cet exemple, détermine qui a le droit ou non de supprimer une tâche.

On a la possibilité d'utiliser un Voter soit via des **annotations**, soit la commande « **denyAccessUnlessGranted** » (ligne 140).

Exemple d'utilisation de la commande « denyAccessUnlessGranted »

```

131 /**
132  * Task deletion page.
133  *
134  * @Route("/tasks/{id}/delete", name="task_delete")
135  *
136  * @throws AccessDeniedException "TASK_DELETE" Voter is not granted.
137  */
138 public function deleteTaskAction(Task $task): RedirectResponse
139 {
140     $this->denyAccessUnlessGranted('TASK_DELETE', $task);
141     $this->doctrine->getManager()->remove($task);
142     $this->doctrine->getManager()->flush();
143
144     $this->addFlash('success', 'La tâche a bien été supprimée.');
```

Dans l'exemple ci-dessus, le Voter est appelé avant chaque suppression de tâche.

Dans le cas d'un retour positif (true), la tâche sera supprimée.

Dans le cas où le retour est négatif (false), une page d'erreur 403 (Accès refusé) est générée.

E. TESTS

Nous avons également mis en place de **tests automatisés** avec **PHPUnit**. La documentation PHPUnit est disponible sur le lien suivant : <https://phpunit.readthedocs.io/fr/latest/>

Chaque **entité** dispose de **tests unitaires** dans le dossier « **Unit** » du répertoire de tests.

De plus, Chaque entité dispose de **tests fonctionnels** dans le dossier « **Functional** » du répertoire de tests.

Tests unitaires de Classe «User»

```
<?php

namespace App\Tests\Unit\Entity;

use App\Entity\Task;
use App\Entity\User;
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase
{
    public function testIsTrue(): void
    {
        $user = new User();
        $user->setEmail('true@test.com');
        $user->setUsername('username');
        $user->setPassword('password');
        $user->setRoles(['ROLE_USER']);

        $this->assertTrue('true@test.com' === $user->getEmail());
        $this->assertTrue('username' === $user->getUsername());
        $this->assertTrue('password' === $user->getPassword());
        $this->assertTrue($user->getRoles() === ['ROLE_USER']);
    }
}
```

Tests fonctionnels de Classe «SecurityController»

```
1 <?php
2
3 namespace App\Tests\Functional\Controller;
4
5 use App\Tests\Functional\AbstractWebTestCase;
6 use Symfony\Component\HttpFoundation\Response;
7
8 class SecurityControllerTest extends AbstractWebTestCase
9 {
10     public function testAccessPageLoginWithBadCredentials(): void
11     {
12         $this->client->followRedirects();
13         $crawler = $this->client->request('GET', '/login');
14         $form = $crawler->selectButton('Se connecter')->form([
15             'username' => 'anonymous',
16             'password' => 'badpassword', []]);
17         $crawler = $this->client->submit($form);
18
19         $this->assertSelectorExists('div.alert.alert-danger:contains("Invalid credentials")');
20     }
21 }
```

En cas de modification de l'entité, il est nécessaire de rajouter les **tests unitaires** et **fonctionnels** correspondants puis **d'exécuter** l'ensemble des **tests** afin **vérifier** le **bon fonctionnement** de l'application (Commande « `php bin/phpunit` »).

Exécution des tests grâce à la commande « `php bin/phpunit` »

```
PS D:\GitProjects\VP8_TodoAndCo> php bin/phpunit
PHPUnit 9.5.21 #StandWithUkraine

Testing
.....                                     38 / 38 (100%)

Time: 00:14.852, Memory: 78.00 MB

OK (38 tests, 92 assertions)

Generating code coverage report in HTML format ... done [00:00.220]
PS D:\GitProjects\VP8_TodoAndCo>
```

F. DOCUMENTATION

L'application est **documentée** avec des annotations **PHPDoc**. Le site de PHPDocumentor (<https://www.phpdoc.org/>) pourra vous aider si besoin.

En cas d'ajout ou de modifications de fonctionnalités dans l'application, pensez à bien **documenter** le code.

Exemple de documentation PHPDoc

```

13  /**
14   * User entity.
15   *
16   * @ORM\Table("user")
17   * @ORM\Entity
18   * @UniqueEntity("email")
19   *
20   * @author Siaka MANSALY <siaka.mansaly@gmail.com>
21   * @package: App\Entity
22   */
23  class User implements UserInterface, PasswordAuthenticatedUserInterface
24  {
25      /**
26       * @ORM\Column(type="integer")
27       * @ORM\Id
28       * @ORM\GeneratedValue(strategy="AUTO")
29       *
30       * @var int the user id
31       */
32      private $id;

```

La commande « `php phpDocumentor.phar` » vous permettra de **regénérer** la **documentation** afin que vos modifications (ou ajouts) apparaissent.

Exemple exécution commande « `php phpDocumentor.phar` »

```

PS D:\GitProjects\P8_ToDoAndCo> php phpDocumentor.phar
phpDocumentor v3.3.1

Parsing files
 14/14 [=====] 100%
Applying transformations (can take a while)
 21/21 [=====] 100%
All done in 1 seconds!

```

G. ALLER PLUS LOIN

Cette documentation n'est pas exhaustive mais vous aura, je l'espère, fournie les bases pour développer au sein de ce projet.

En cas de modification au niveau des entités, n'hésitez pas à effectuer les **migrations**.

Ecrivez les **tests unitaires et fonctionnels** des nouvelles fonctionnalités ou modifications.

N'hésitez pas à consulter la **documentation PHP** de l'application qui se situe dans le répertoire « **PHPDoc** » du dossier « **docs** » et peut être consultée en locale.

Dernière chose, pensez à bien **documenter votre code** pour moi et surtout vos futurs collègues.