Documentation Technique du Site ToDo & Co

Introduction

La présente documentation technique détaille l'architecture, les composants clés, et le fonctionnement du site ToDo & Co développé en utilisant le framework PHP Symfony.

Sommaire

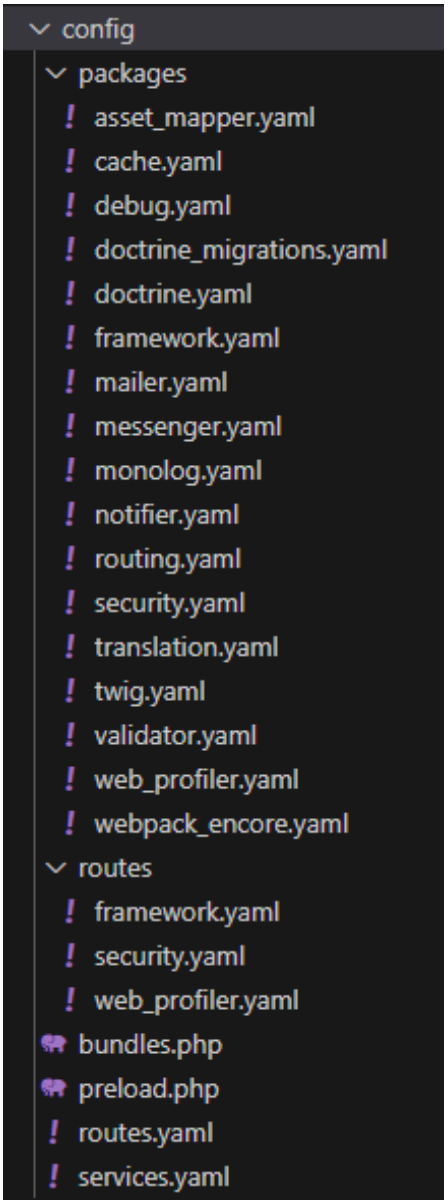
I. Vue d'ensemble

A. Structure du Projet

Le projet suit la structure standard de Symfony, avec les répertoires principaux suivants :

Config

Configuration de l'application



Ici nous avons les fichiers de configuration.

Symfony utilise le langage YAML pour ses fichiers de configuration.

Le seul fichier non trivial ici est : config/packages/security.yaml

Ce fichier contient les configurations de sécurité de l'application.

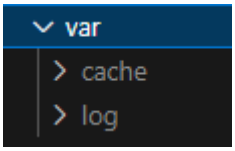
Il permet de faire deux choses. il gère l'authentification et l'access control (restriction de routes à certains utilisateurs)

```yaml
access_control:
    - { route: task_create, roles: ROLE_USER }
    - { route: task_edit, roles: ROLE_USER }
```

Exemple d'access control restreignant l'accès à la création d'une tâche aux utilisateurs non connectés

Var

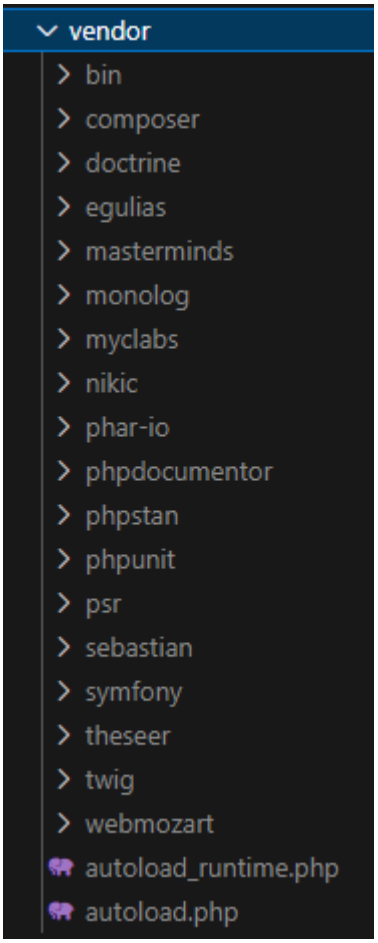
Fichiers temporaires



Le dossier var contient les fichiers temporaires tels que les logs.

Vendor

Librairies Composer



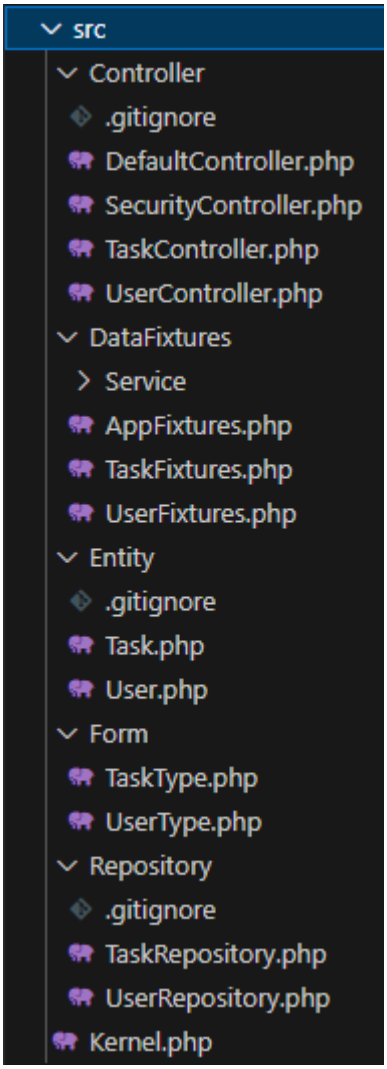
Le dossier contient les fichiers des librairies installées avec Composer.

Ce dossier n'est en pratique jamais versionné et modifié.

Le fichier autoload.php, généré avec Composer permet au projet de charger les librairies simplement.

Src

code source de l'application



Ce dossier contient le code source de l'application.

C'est ici que la plupart des modifications vont êtres effectuées.

Symfony suit le modèle MVC (Modèle-Vue-Contrôleur) pour assurer une séparation claire des préoccupations. Les fichiers de contrôleurs se trouvent dans le répertoire src/Controller, les entités dans src/Entity, et les vues dans le répertoire templates.

Les entités sont des classes particulièrement importantes. il s'agit de classes reliées à la base de données par l'intermédiaire de Doctrine, l'orm de symfony

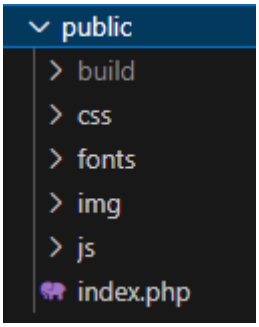
Le dossier Controller contient les contrôleurs, qui sont les classes responsables de lier les entités (Modèle) et les templates (Vue)

Le dossier form contient les fonctions permettant de créer un formulaire pour chaque entité.

Enfin le dossier DataFixtures contient des classes permettant de créer rapidement un jeu de fausses données.

Public

fichiers publics

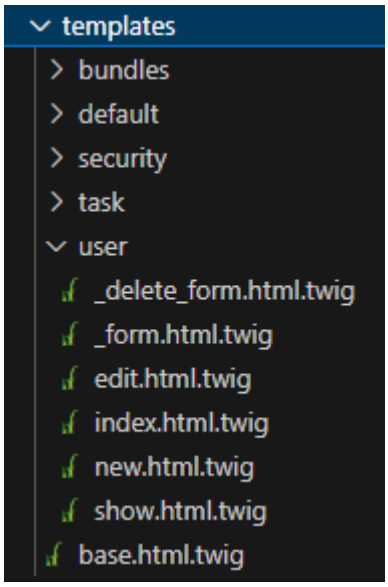


Ce dossier contient les fichiers accessibles à tous.

Notamment, le sous dossier build contient les fichiers CSS et JS minifiés par Webpack encore

Templates

html avec twig



Ce dossier contient tout l'HTML de notre application.

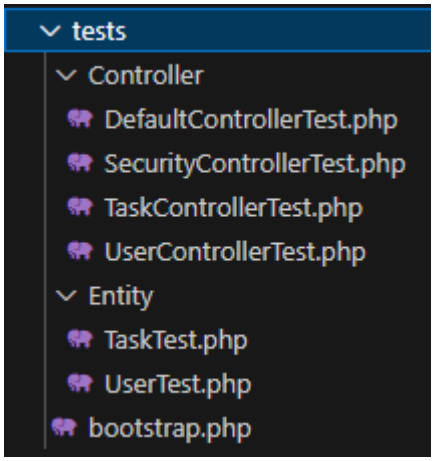
Il contient des fichiers .twig qui utilisent du code twig, permettant de facilement créer des templates de fichiers HTML

Exemple de syntaxe :

```
{% for user in users %}
    <tr>
        <td>{{ user.id }}</td>
```

Test

tests unitaires et fonctionnels



Ce dossier contient les fichiers de test.

Ces fichiers sont écrits avec PHPUnit.

Le sous dossier Entity contient les tests **unitaires** relatifs aux entités. (chaque méthode de l'entité est testée)

Quant au sous-dossier Controller, il contient les tests **fonctionnels** relatifs aux contrôleurs (chaque route est testée)

II. Authentification

A. Le back-end

l'authentification commence dans l'entité User.

```php
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\Table(name: `user`)]
77 references | 2 implementations
class User implements UserInterface, PasswordAuthenticatedUserInterface
```

l'entité User implémente deux interfaces. l'interface UserInterface utilise des méthodes pour l'authentification avec Symfony.

Il y a ensuite un contrôleur : SecurityController qui contient deux méthodes : login et logout.

ces méthodes sont presque vides et servent principalement à indiquer la ROUTE

```php
#[Route(path: '/login', name: 'login')]
```

Ainsi que la Vue (dans le cas de LOGIN)

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

Enfin ces informations sont données à Symfony a l'aide du fichier config/packages/Security.yaml

on indique d'abord le PROVIDER entity

```yaml
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

on indique ensuite à Symfony comment ce provider doit se connecter

(se connecter : utiliser la route login, se déconnecter utiliser la route logout)

```yaml
main:
    lazy: true
    provider: app_user_provider
    form_login:
        login_path: login
        check_path: login
        enable_csrf: false
    logout:
        path: logout
```

B. Le front-end

Ce code permet donc à l'utilisateur de se connecter à l'aide du formulaire situé dans templates/security/login.html.twig. Ce formulaire contient obligatoirement deux champs appelés _username et _password.

le champ _username demande ici l'adresse mail de l'utilisateur (comme indiqué dans le fichier security.yaml)

```yaml
entity:
    class: App\Entity\User
    property: email
```

Pour déconnecter l'utilisateur, il suffit de le renvoyer sur la route logout.

```twig
{% if app.user %}
<a href="{{ path('logout') }}" class="pull-right btn btn-danger">Se déconnecter</a>
{% endif %}
```

III. Les données

Les données, telles que les informations sur les utilisateurs sont stockées dans la base de données.

Pour y accéder il faut tout d'abord la créer. Pour cela il faut ajouter la variable d'environnement dans le fichier .env.local :

```
DATABASE_URL="mysql://USERNAME:PASSWORD@SERVEUR:PORT/BASE_DE_DONNEE?serverVersion=8.0.32&charset=utf8mb4"
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
```

ensuite il faut effectuer une migration à l'aide des commandes :

Ensuite, pour créer la base de données et … il faut exécuter les commandes suivantes :

"php bin/console doctrine:database:create"

Le système symfony migration versione les changements à travers les fichiers de migrations. ceux-ci sont créés à l'aide de la commande :

"php bin/console make:migration"

Les fichiers de migration contiennent deux fonctions : up() et down()

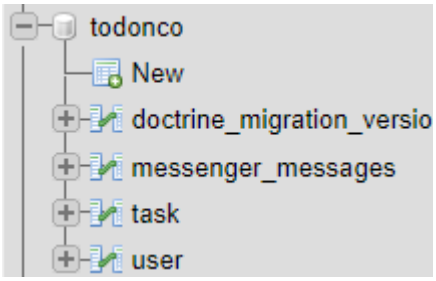
```
public function up(Schema $schema): void
{
    // this up() migration is auto-generated, please modify it to your
    $this->addSql('CREATE TABLE task (id INT AUTO_INCREMENT NOT NULL, c
    $this->addSql('CREATE TABLE `user` (id INT AUTO_INCREMENT NOT NULL,
    $this->addSql('CREATE TABLE messenger_messages (id BIGINT AUTO_INCR
}

0 references
public function down(Schema $schema): void
{
    // this down() migration is auto-generated, please modify it to your
    $this->addSql('DROP TABLE task');
    $this->addSql('DROP TABLE `user`');
    $this->addSql('DROP TABLE messenger_messages');
}
```

enfin la commande :

“php bin/console doctrine:migration:migrate”

Exécute les requêtes SQL contenues dans la fonction up().



la base de données est désormais créée et peuplée des tables nécessaires au fonctionnement de l’application (user et task ici)

C’est ici que se stockent les données de l’application.

En effet c’est Doctrine, l’orm qui lie les Entités avec cette base de données.

Par exemple, lorsque le formulaire sur la route user_create dans le UserController est rempli, ces actions sont effectuées :

```
$entityManager->persist($user);
$entityManager->flush();
```

Ces deux lignes indiquent la Doctrine d’ajouter l’utilisateur à la base de données.

enfin, avant d’être ajoutés, les mots de passes sont hachés à l’aide de la commande :

“ $user->setPassword($passwordHasher->hashPassword($user, $form->getData()->getPassword()));”

Cela utilise le passwordHasher indiqué dans le fichier security.yaml

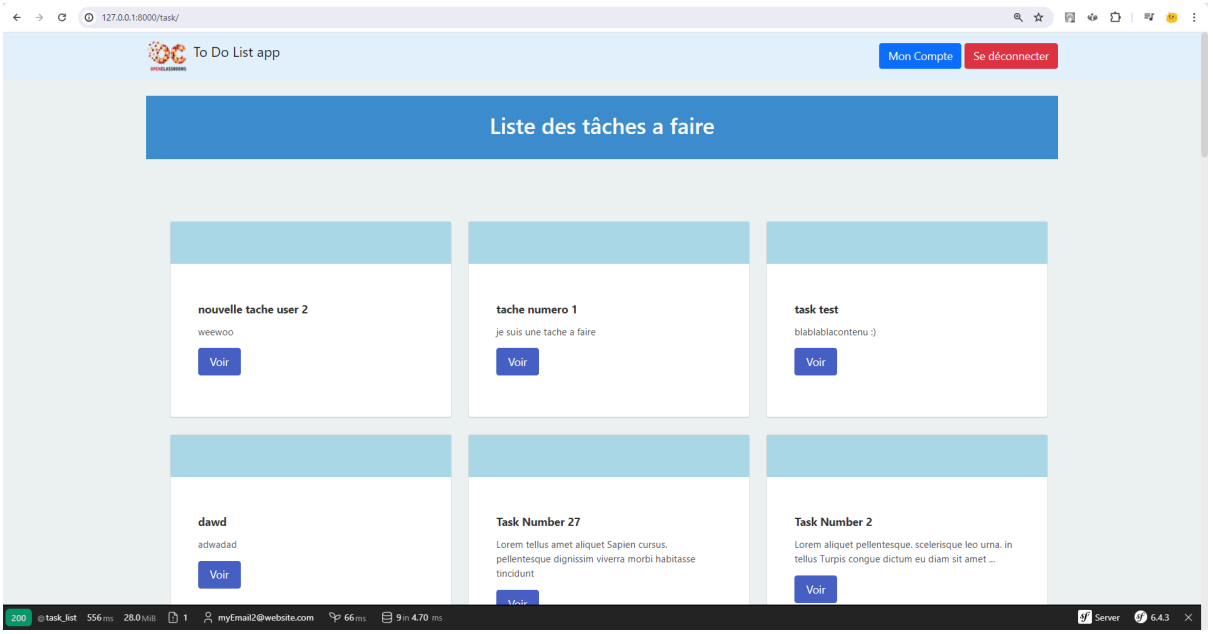
```
password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

IV. Performance

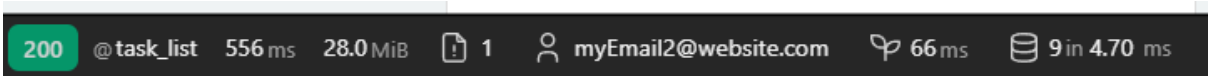
A. Rapport de performance

La performance de l'application est globalement bonne,

Cela peut être constaté à l'aide d'un profiler, comme par exemple celui de Symfony



sur la page concernant la liste des tâches à faire, avec le profiler activé, on observe des bonnes performances



On voit ici :

- Un chargement de la page en ½ seconde
- 28 MiB d'utilisation de mémoire
- 4.7ms de temps de requête vers la base de données