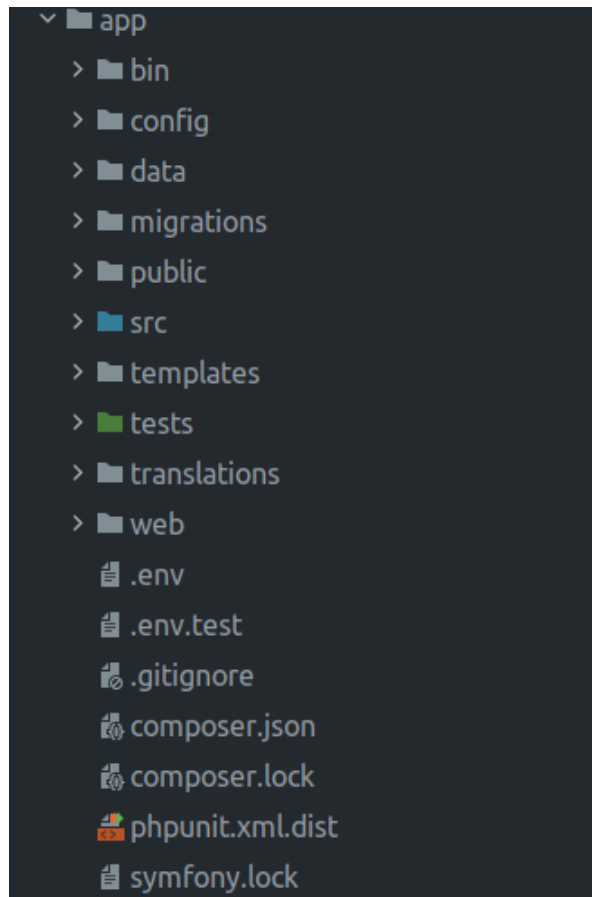


Documentation technique

To-do List

Explication de l'architecture.....	2
Authentification	4
1) Gestion des paramètres.....	4
a) Provider	4
b) Firewall	5
c) Access Control.....	6
2) Gestion des droits	7

Explication de l'architecture



Il y a plusieurs dossiers dans l'architecture de Symfony et chacun à sa fonction.

- Bin, contient les exécutables de notre projet (actuellement, console et phpunit), il est modifié automatiquement si nécessaire et ne nécessite donc pas de manipulation de notre part.
- Config, contient tous les fichiers de paramétrage de Symfony et ses dépendances nous serons amenés à le modifier afin de personnaliser certain réglage si nécessaire (authentification).
- Migrations, est un dossier dont les fichiers sont modifiés automatiquement lors de la mise en place et des mises à jour de la base de données. (Voir documentation Doctrine)
- Public, contient les fichiers de style (les assets c'est-à-dire le CSS, les polices, les images, le JS, etc..) c'est un dossier qui sera modifié lors de changement du frontend du site.
- Src, contient la logique métier de notre application et sera le dossier qui sera modifié à chaque mise à jour des fonctionnalités.

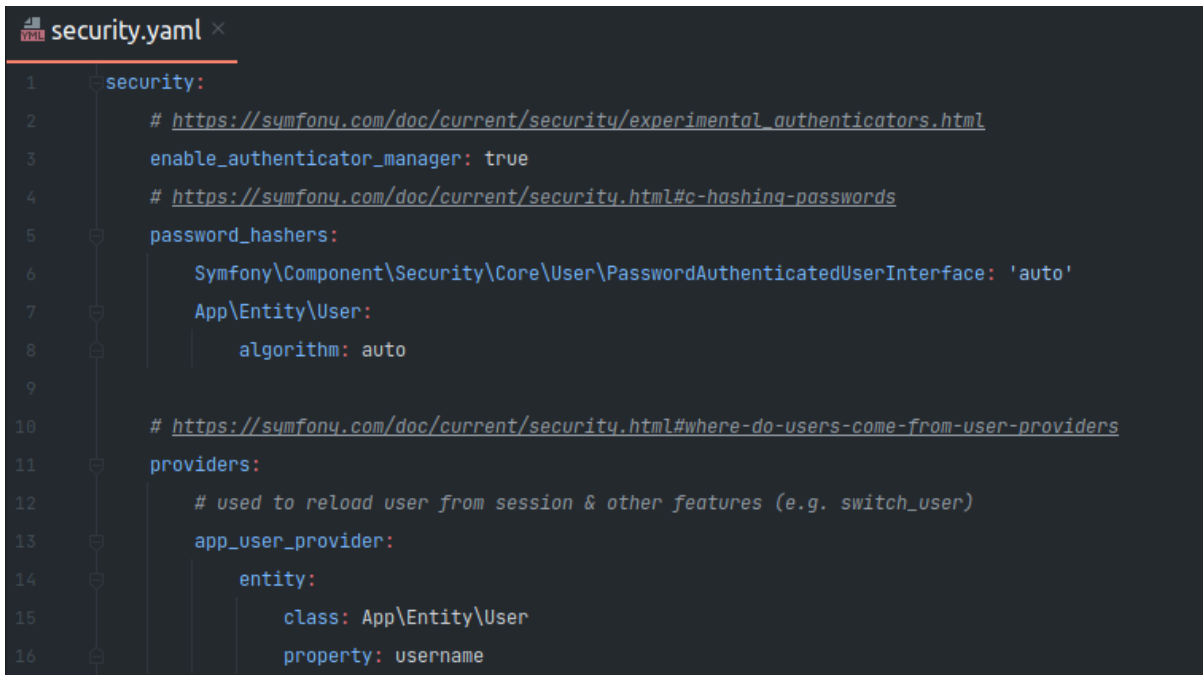
- Templates, contient le html de nos pages sous format “twig” et sera aussi modifié lors de changement du frontend
- Test, contient les tests unitaire et fonctionnel de l’application
- Translations, contient les fichiers permettant de traduire le contenu du site sur plusieurs langages (possible amélioration)
- Var, contient les données temporaires de l’application et n'est modifié principalement que pour effacer le cache, il est modifié automatiquement si nécessaire et ne nécessite donc pas de manipulation de notre part.
- Vendor, contient les dépendances du projet, il est modifié automatiquement si nécessaire et ne nécessite donc pas de manipulation de notre part.
- Web, contient le rapport de couverture du code de notre application, il est généré automatiquement si nécessaire et ne nécessite donc pas de manipulation de notre part.

Authentification

1) Gestion des paramètres

Le document contenant les paramètres de l'authentification est config/packages/security.yaml

a) Provider



```
1 security:
2     # https://symfony.com/doc/current/security/experimental_authenticators.html
3     enable_authenticator_manager: true
4     # https://symfony.com/doc/current/security.html#c-hashing-passwords
5     password_hashers:
6         Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
7         App\Entity\User:
8             algorithm: auto
9
10    # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
11    providers:
12        # used to reload user from session & other features (e.g. switch_user)
13        app_user_provider:
14            entity:
15                class: App\Entity\User
16                property: username
```

Le provider est le paramètre qui précise où sont stockés les utilisateurs et avec quelles informations un utilisateur peut être authentifié.

Dans notre cas, notre user provider est l'entité User et la propriété avec laquelle l'utilisateur peut se connecter est l'identifiant. Cela signifie qu'une fois notre utilisateur créé, il est enregistré dans notre base de données et lors de l'authentification cette table sera parcourue afin de trouver l'utilisateur correspondant.

b) Firewall

```
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

    entry_point: form_login
    logout:
      path: app_logout
```

Le pare-feu est le principal composant de Security, c'est lui qui va automatiquement s'assurer à chaque requête que l'utilisateur soit identifié et soit autorisé à accéder à la page demandée. S'il n'est pas identifié il redirige automatiquement vers la méthode de connexion spécifiée, s'il n'est pas autorisé il refuse l'accès. Il assure aussi la déconnexion.

Dans notre application, nous avons choisi de nous authentifier avec un formulaire (form_login), vers lequel l'utilisateur est redirigé dès qu'il cherche à accéder à une page à accès restreint.

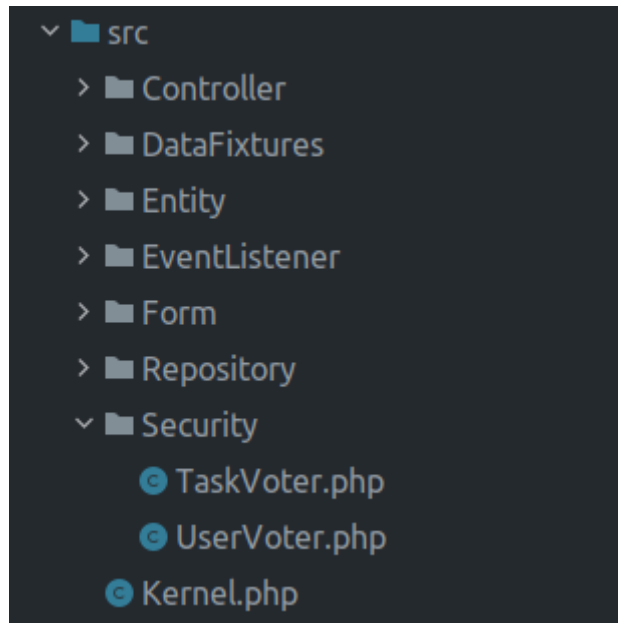
c) Access Control

```
# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    # - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }
```

Le Framework Symfony introduit un concept de rôle, chaque utilisateur a un rôle et ce rôle lui confère des droits. Cette section des paramètres permet au développeur s'il le souhaite de rendre l'accès à certaine partie de son site interdit en fonction du rôle de l'utilisateur.

Dans notre application, vu que la majorité de nous avons choisi de ne pas utiliser l'Access control, mais de faire la sécurisation dans les controllers.

2) Gestion des droits



Le choix a été fait de gérer les droits d'accès avec des Voter. Pour cela, nous en avons créé deux :

- Gestion des droits sur les tâches
- Gestion des droits de modification des utilisateurs

```

<?php

namespace App\Security;

use ...

class TaskVoter extends Voter
{
    const OWN = 'own';

    private Security $security;

    public function __construct(Security $security)
    {
        $this->security = $security;
    }

    protected function supports(string $attribute, $subject): bool
    {
        //if the attribute isn't one we support, return false
        if ($attribute !== self::OWN) {
            return false;
        }

        // only vote on 'Post' objects
        if (!$subject instanceof Task) {
            return false;
        }

        return true;
    }
}

```

Ces voter sont des instances du VoterInterface et permettent de conférer à certains utilisateurs des droits s'ils correspondent à des critères précis.


```

protected function voteOnAttribute(string $attribute, $subject, TokenInterface $token)
{
    $user = $token->getUser();

    if (!$user instanceof User) {
        // the user must be logged in; if not, deny access
        return false;
    }

    if ($this->security->isGranted('ROLE_ADMIN')) {
        return true;
    }

    $task = $subject;

    switch ($attribute){
        case self::OWN:
            return $this->isOwned($task, $user);
        }
    }

    private function isOwned(Task $task, UserInterface|User $user): bool
    {
        return $user == $task->getAuthor();
    }
}

```

Dans le cas de notre TaskVoter par exemple une tâche ne peut être modifiée que par l'utilisateur qui l'a créé ou par un administrateur. Si l'utilisateur répond à l'une de ces contraintes il lui est conféré l'attribut OWN qui lui donne l'autorisation de modifier la tâche.