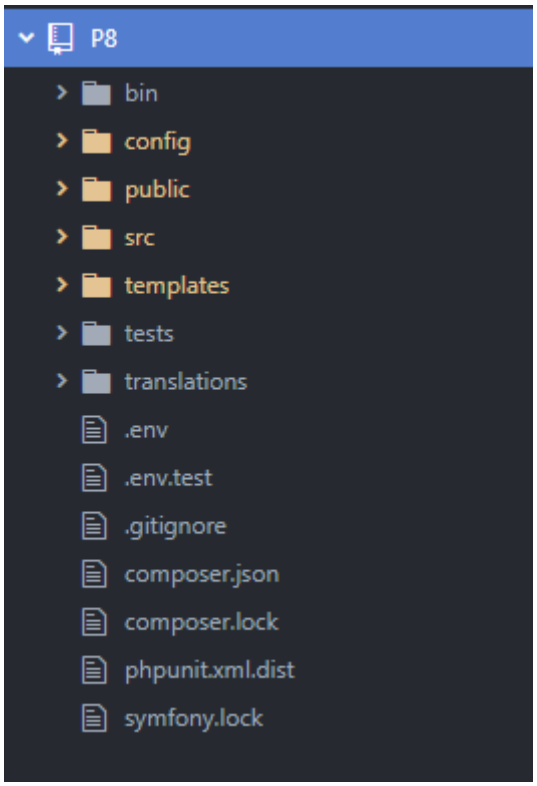
Documentation technique

Le framework Symfony 5 :

Architecture :

L'architecture du framework Symfony se compose de plusieurs dossiers et fichiers.



Les éléments les plus importants :

- Le dossier « config » contient les paramètres de Symfony et des bundles installés sur le projet.
- Le dossier « public » contient les différents assets des pages (css/js/fonts/etc..)
- Le dossier « src », l'un des plus importants, contient le code du projet. (Les controlleurs, les entitys, les form)
- Le dossier « templates » contient les vues et les templates Twig du projet.
- Le dossier « tests » contient les tests unitaires et fonctionnels du projet.
- Le fichier « .env » contient les variables d'environnement (database, mailer, …)
- Le fichier « composer.json » contient la liste de l'ensemble des fonctionnalitées de symfony et des bundles installés, ainsi que leur numéro de version.

Authentification

1. Base de donnée

La base de donnée du projet a été modifiée afin d'y intégrer les tables relatives à l'authentification.
Une table appelée « user » contient les informations des utilisateurs inscrit.
Elle comprend donc :

- Un champ « id »
- Un champ « username »
- Un champ « password »
- Un champ « email »
- Et un champ « roles »

2. \config\packages\security.yaml :

La section encoders concerne le chiffrement du mot de passe des utilisateurs.
Pour ce projet, l'encoder bcrypt est utilisé .

```
encoders:
    App\Entity\User: bcrypt
```

La section providers défini quelle entity et quel champs serviront à l'authentification.
Pour ce projet, le champs username de l'entity User est utilisé.

```
providers:
    doctrine:
        entity:
            class: App:User
            property: username
```

La section firewalls défini les différents path pour le système d'authentification.

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false

    main:
        anonymous: ~
        pattern: ^/
        form_login:
            login_path: login
            check_path: login_check
            always_use_default_target_path:  true
            default_target_path:  /
        logout: ~
```

La section access_control sers à définir les accès des rôles aux différentes parties du projet.
Sur ce projet, on constate que les path login et users/create sont accessibles aux anonymes, le reste du path /users est seulement accessible aux administrateurs, et le reste du projet est accessible à toutes les personnes connectées (avec le rôle ROLE_USER).

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

3. \src\Entity\User.php

Le fichier User.php contient les getters et les setters de notre entité user.
Un champs rôle à donc été ajouté dans la table user de la base de donnée afin de définir le rôle (User ou Admin) de chaque utilisateur.

Les fonctions importantes relatives à l'authentification :

- getUsername et setUsername : « get et set » pour le champs username.
- getPassword et setPassword : « get et set » pour le champs password.
- getRoles et setRoles : « get et set » pour le champs role.

Ce sont ces getters et setters qui font le lien avec la base de donnée.

```php
public function getUsername()
{
    return $this->username;
}

public function setUsername($username)
{
    $this->username = $username;
}
```

```php
public function getPassword()
{
    return $this->password;
}

public function setPassword($password)
{
    $this->password = $password;
}
```

```php
public function setRoles($roles)
{
    $this->roles = $roles;
}

public function getRoles()
{
    $roles = $this->roles;
    $roles[] = 'ROLE_USER';

    return array_unique($roles);
}
```

4. \config\routes.yaml

Ce fichier nous permet de créer les différentes routes du projet.
Concernant l'authentification, 3 routes sont importantes dans ce fichier :

```yaml
login:
  path: /login
  controller: App\Controller\SecurityController::loginAction

login_check:
  path: /login_check

logout:
  path: /logout
```

- login : cette route redirige vers la fonction loginAction du SecurityController.
- login_check : cette route permet à Symfony de procéder à l'authentification et de la vérifier.
- logout : cette route permet à Symfony de procéder à la déconnexion de l'utilisateur.

5. \src\Controller\SecurityController.php

Le fichier SecurityController contient la fonction liée à la route login.

Elle permet de rediriger l'utilisateur sur le formulaire de connexion avec différentes options.

L'option last_username permet de préremplir le formulaire avec le dernier username de l'utilisateur.

L'option error se charge d'afficher les différents messages d'erreur en cas de problème de connexion.

```php
class SecurityController extends AbstractController
{

    public function loginAction(AuthenticationUtils $authenticationUtils)
    {

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

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

Comment gérer les accès au projet ?

Il existe différentes manières de gérer les accès aux différents rôles sur le projet :

1. \config\packages\security.yaml :

Dans la section acces_control de ce fichier, il suffit de définir les différents path que vous voulez sécuriser, et spécifier pour quel rôle celui ci reste accessible.

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

IS_AUTHENTICATED_ANONYMOUSLY : Définit que le path est accessible aux utilisateurs anonymes.

ROLE_ADMIN : Le path est accessible seulement pour les utilisateurs ayant le rôle Admin.

ROLE_USER : Le path est accessible seulement pour les utilisateurs connectés au projet.

2. Dans le code des controllers :

```php
$user = $this->getUser();
$roles = $user->getRoles();
$repository = $this->getDoctrine()->getRepository(Task::class);
$taskvalid = $repository->findOneBy(['id' => $taskid, 'userCreate' => $userid]);

if (!empty($taskvalid) or $roles[0] == 'ROLE_ADMIN') {
  $form = $this->createForm(TaskType::class, $task);
```

Directement dans le code de vos fonctions, vous pouvez faire des vérifications de rôle utilisateur.

Vous pouvez récupérer les informations de l'utilisateur grâce à la ligne « $user = $this->getUser() ; »
Ensuite, récuperez le rôle de cet utilisateur avec la fonction getRoles() ;
Enfin, réaliser la vérification, comme sur le deuxième encadré de la capture ci-dessus.

3. Dans les templates Twig :

Vous pouvez également gérer les vues en fonction de l'authentification de l'utilisateur.

Les vues se trouvent dans le dossier templates du projet.

```
<div class="">
1. {% if not app.user %}
   <a href="{{ path('user_create') }}" class="btn btn-primary">Inscription</a>
   {% endif %}

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

3. {% if app.user and is_granted('ROLE_ADMIN') %}
   <a href="{{ path('user_create') }}" class="btn btn-primary">Créer un utilisateur</a>
   <a href="{{ path('user_list') }}" class="btn btn-primary">Liste des utilisateurs</a>
   {% endif %}

</div>
```

Sur cette exemple, différentes vérifications sont réalisées :

1. On affiche le lien si l'utilisateur n'est pas connecté.
2. On affiche le lien si l'utilisateur est connecté.
3. On affiche les liens si l'utilisateur est connecté et si il possède le rôle Admin.

Important :

Des tests unitaires et fonctionnels ont été implémentés avec PhpUnit.
Veillez a réaliser ces tests à la fin des modifications afin de vérifier que le projet fonctionne toujours correctement.

Vous pouvez réaliser ces tests en ouvrant votre cmd et en tapant la commande suivante :
« php bin/phpunit --coverage-html tests/Rapports »

Si la réponse obtenue contient un message écrit en vert, vos tests sont validés et vos modifications non pas impacté le bon fonctionnement de l'application.

De plus, veillez également à respecter les normes PSR 1 et 12 en vigueur sur l'ensemble de votre code.