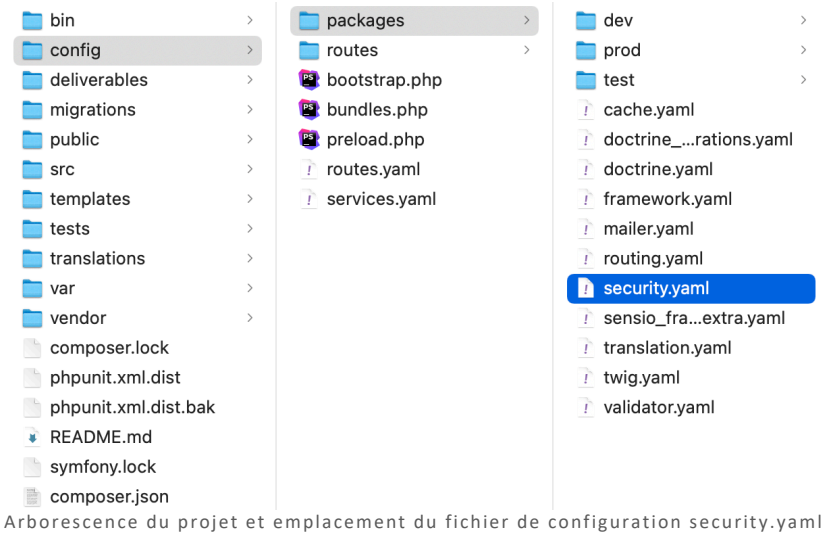
ToDo & Co .

Comprendre et utiliser le composant Security de Symfony pour le projet ToDo & Co.

Table des matières

I- Introduction.

La configuration de l'authentification se trouve dans le dossier config du projet Symfony : *./config/packages/security.yaml*. Pour plus d'informations quant à son installation, veuillez-vous référer à la documentation officielle concernant le composant Security de Symfony.



Arborescence du projet et emplacement du fichier de configuration security.yaml

Ci-après, le descriptif de chaque section de configuration de l'authentification pour ce projet-ci.

II- Configuration.

1- Provider.

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

Lors de l'authentification, le firewall devra identifier la personne qui souhaite se connecter et ira le chercher dans un fournisseur d'utilisateur, ou **Provider**.

Ici, grâce aux informations soumises par l'utilisateur, on indique donc que le firewall devra comparer et authentifier celui-ci par l'entité **User**, qui représente une table 'user' en base de données (Voir doc Symfony sur Doctrine et les Entités). Il est possible de paramétrer plusieurs providers, de différents types (Voir la documentation).

La propriété `property` correspond à identifiant visuel unique permettant d'identifier et comparer un utilisateur lors de l'authentification. Par exemple, cela peut être soit un **username, un uuid, ou un email**.

L'Entité qui servira à une authentification doit forcément étendre **UserInterface** qui possède des méthodes nécessaires à l'authentification.

2- Encoder.

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

L'option `encoders` permet de choisir l'algorithme qui permettra de crypter les mots de passes de l'application.

On indique ici que le champ **'password'** du provider utilisant l'entité **'User'** aura comme algorithme **'bcrypt'**.

3- Role Hierachy.

```
role_hierarchy:
    ROLE_ADMIN: ROLE_USER
```

On peut définir des rôles qui conditionneront les actions d'un utilisateur sur l'application. On parlera donc d'autorisations.

Ici, dans l'application nous avons défini deux rôles : un **ROLE_ADMIN** et un **ROLE_USER**.

La propriété `role_hierarchy` définira une hiérarchie de rôles dans notre projet : **ROLE_ADMIN** héritera de tous les droits des utilisateurs possédant le rôle **ROLE_USER**, mais l'inverse n'est pas vrai.

4- Firewalls.

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        provider: app_user_provider
        anonymous: ~
        pattern: ^/
        form_login:
            login_path: login
            check_path: login_check
            always_use_default_target_path: true
            default_target_path: /
        logout: ~
```

Le firewall définit le mécanisme d'authentification utilisé pour chaque requête. C'est-à-dire qu'à chaque requête effectuée par un utilisateur, l'application vérifiera dans cette section si l'utilisateur peut accéder ou non à une zone définie.

La section `dev` n'est pas à prendre en compte ici : elle s'assure juste que dans un environnement de développement, l'accès au profiler de Symfony (Voir la documentation) n'est pas bloqué par aucune logique métier d'accès à une ressource.

La section `main` représente le fonctionnement du firewall de l'application. C'est ici que va être déterminé le champ d'action de l'utilisateur selon **qui il est** : S'il est « authentifié » en tant qu'anonyme, en qu'utilisateur pleinement authentifié avec un rôle ou non … le tout, selon la requête et le niveau d'accès de l'application.

On va décortiquer chaque paramètre de la section :

- `provider` : Sélection du provider sur lequel le firewall va authentifier l'utilisateur. Ici, on choisira App\Entity\User.
- `anonymous` : Cela défini l'utilisateur non authentifier en tant qu'anonyme. Cela permet définir des routes accessibles aux utilisateurs anonyme malgré certaines restrictions.
- `pattern` : Définition du chemin d'accès à partir duquel le firewall vérifiera toujours la nature de l'utilisateur, et s'il est autorisé à y accéder. Ici, avec le pattern « `^/` » on vérifiera toujours que chaque utilisateur possède ce qu'il faut pour accéder à la ressource.
- `form_login` : Prend en charge automatiquement une requête HTTP de type POST (via le formulaire de connexion associé) :
    - `login_path` : Détermine le chemin d'accès du formulaire de connexion associé.
    - `login_check` : Sur cette route, le firewall va intercepter la fameuse requête HTTP de type POST pour authentifier ou non, l'utilisateur.
    - `always_use_default_target_path` : Ce paramètre indique au firewall d'ignorer la précédente requête lors de l'authentification, si l'option est sur le booléen « true », et de le rediriger vers un chemin par défaut.
      Exemple : Je veux les utilisateurs > redirection vers le formulaire de login > authentification > redirection vers le chemin par défaut.
    - `default_target_path` : Chemin par défaut après chaque authentification. Ignore l'URL précédemment entré et filtré par le firewall.
- `logout` : Détermine comment détruire la session de l'utilisateur complètement authentifié et sur quel chemin (ici « /logout »).

5- Access Control.

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

Dans cette partie, on détermine les accès selon le rôle de chaque utilisateur qui utilise l'application.
Par exemple, pour tous les chemins commençant par :

- « **^/login** » : Les utilisateurs anonymes peuvent accéder à la ressource.
- « **^/users** » : Seuls les utilisateurs authentifiés ayant le rôle **ROLE_ADMIN** peuvent accéder à la ressource.
- « **^/** » : Seuls les utilisateurs authentifiés ayant le rôle **ROLE_USER (et donc ROLE_ADMIN par héritage des rôles)** peuvent accéder à la ressource.