

Guide pour l'authentification.

I) L'entité User.

Afin qu'un utilisateur puisse s'authentifier, il faut d'abord générer l'entité User qui déterminera les attributs de chaque utilisateur inscrit à l'application. Cette entité est représentée par la classe User située dans le fichier `src/Entity/User.php`, et pour permettre à l'utilisateur inscrit de se connecter à l'application elle doit implémenter différentes fonctions de l'interface `UserInterface`.

```

    * @see UserInterface
    */
    public function getRoles(): array
    {
        $roles = $this->roles;
        // guarantee every user at least has ROLE_USER
        $roles[] = 'ROLE_USER';

        return array_unique($roles);
    }

    public function setRoles(array $roles): self
    {
        $this->roles = $roles;

        return $this;
    }

    /**
     * @see UserInterface
     */
    public function getPassword(): string
    {
        return (string) $this->password;
    }

    public function setPassword(string $password): self
    {
        $this->password = $password;

        return $this;
    }

    /**
     * @see UserInterface
     */
    public function getSalt()
    {
        // not needed when using the "bcrypt" algorithm in security.yaml
    }

    /**
     * @see UserInterface
     */
    public function eraseCredentials()
    {
        // If you store any temporary, sensitive data on the user, clear it here
        // $this->plainPassword = null;
    }

```

Vous pourrez retrouver le rôle de chacune de ces méthodes via le lien suivant:

<https://github.com/symfony/symfony/blob/4.0/src/Symfony/Component/Security/Core/User/UserInterface.php>

II) Le fichier de configuration de l'authentification.

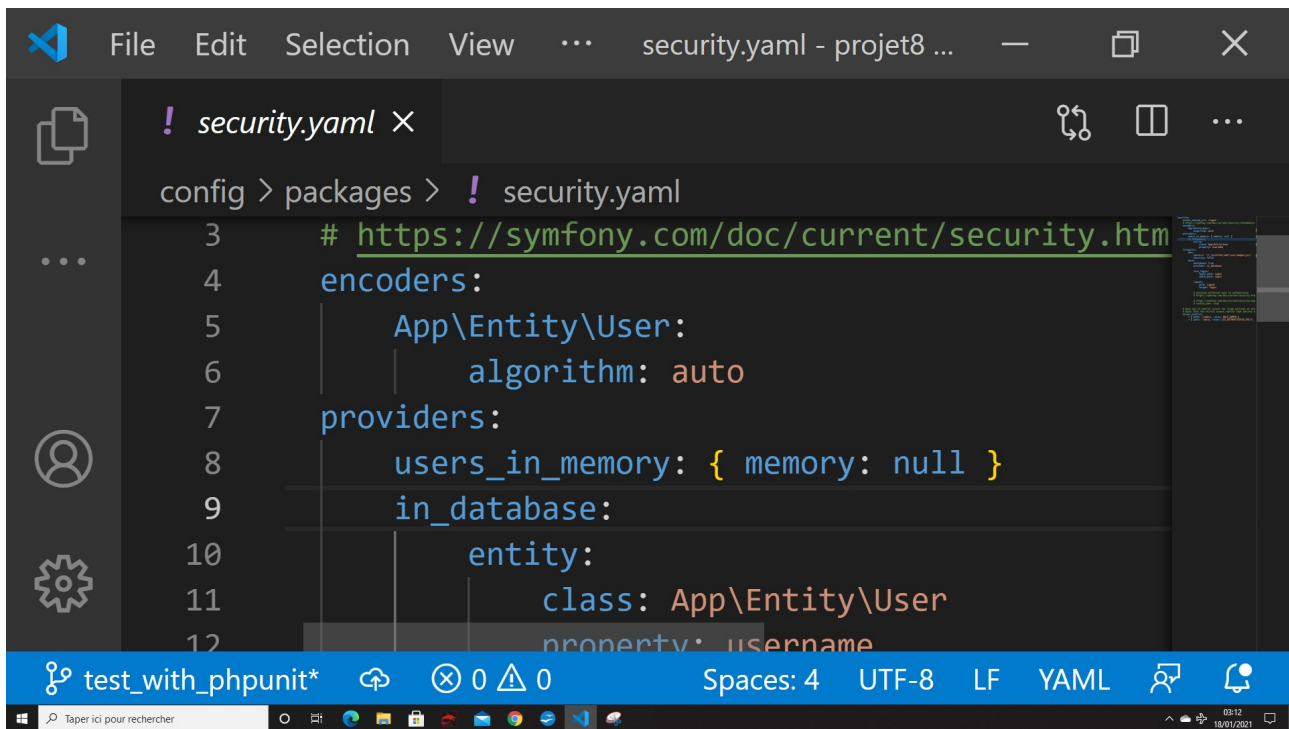
La configuration de l'authentification se fait dans le fichier `config/packages/security.yaml`. Ce fichier comprend plusieurs éléments:

A) Les providers.

Le provider indique l'emplacement des informations utilisées pour authentifier l'utilisateur. Le schéma ci dessous indique que l'authentification se réalise via Doctrine par l'entité User dont l'attribut username servira à identifier l'utilisateur lors de la connexion.

```
# config/packages/security.yaml
providers:
    in_memory: { memory: ~ }
    doctrine:
        entity:
            class: App\User
            property: username
```

B) Les encoders.



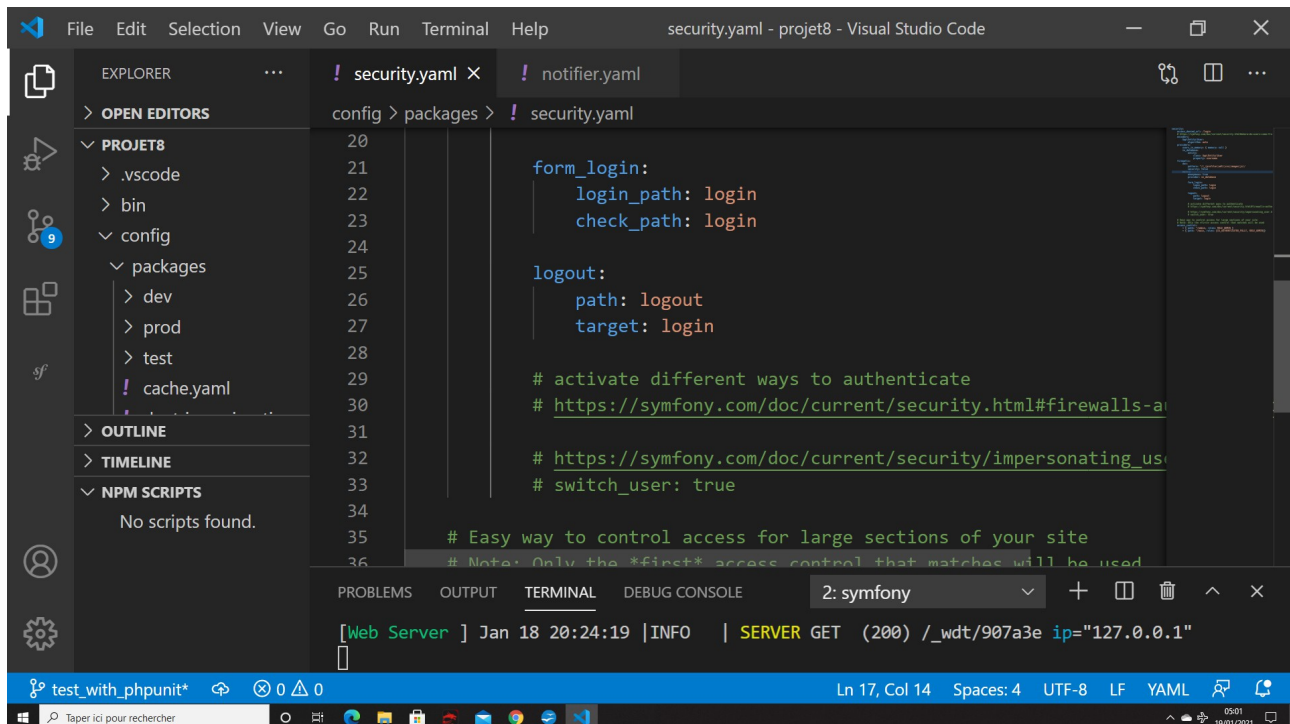
L'encodeur sert à définir l'algorithme que l'on souhaite utiliser pour encoder un attribut dans une entité donnée. Comme le montre l'image ci dessus, dans le cadre de ce projet, nous allons utiliser l'algorithme auto pour encoder le mot de passe, de l'entité password, via UserPasswordEncoderInterface. L'algorithme auto génère lui même l'encodage le mieux adapté pour sécuriser le mot de passe.

C) Les firewalls.

Le rôle du firewall est d'indiquer la manière dont les utilisateurs vont être authentifiés sur certaines parties du site. Le firewall dev concerne uniquement la phase de développement, il ne doit pas être changé, de même que le profiler. Le Firewall main enveloppe le site tout entier à partir de la racine désigné via pattern: ^/, il est possible d'accéder au site sans être authentifié, on y indique Doctrine comme provider qui sera utilisé.

Les routes du formulaire de connexion, comme le montre l'image ci dessous, sont définis à partir de form_login. La route qui mène au formulaire est désigné par login_path, tandis que route de vérification de ce formulaire est désigné par check_path, pour les deux cas il s'agit de la route /login. A partir de logout nous définissons la route pour la déconnexion, et le chemin de la redirection suite à la déconnexion. Effectivement, comme l'expose la figure ci

dessous, le chemin pour se déconnecter est /logout, et cela ramène à la page de connexion /login.



D) Les Access Control.

L'access control permet de contrôler l'accès à certaines parties du site aux utilisateurs en fonction de leurs rôles. Grâce à cela, on peut par exemple permettre l'accès à certaines parties du site uniquement aux utilisateurs qui ont le rôle Admin. Dans le cadre de ce projet, nous avons utilisé l'access control pour permettre l'accès aux routes commençant par /admin uniquement aux utilisateurs qui ont le rôle admin, et les routes commençant par /main sont accessibles aux autres utilisateurs connectés, mais aussi à ceux qui ont le rôle admin, c'est ce qu'illustre l'image suivante:

```
32 | # https://symfony.com/doc/current/security/impersonating_user.  
33 | # switch_user: true  
34 |  
35 | # Easy way to control access for large sections of your site  
36 | # Note: Only the *first* access control that matches will be used  
37 | access_control:  
38 | - { path: ^/admin, roles: ROLE_ADMIN }  
39 | - { path: ^/main, roles: [IS_AUTHENTICATED_FULLY, ROLE_ADMIN]}
```

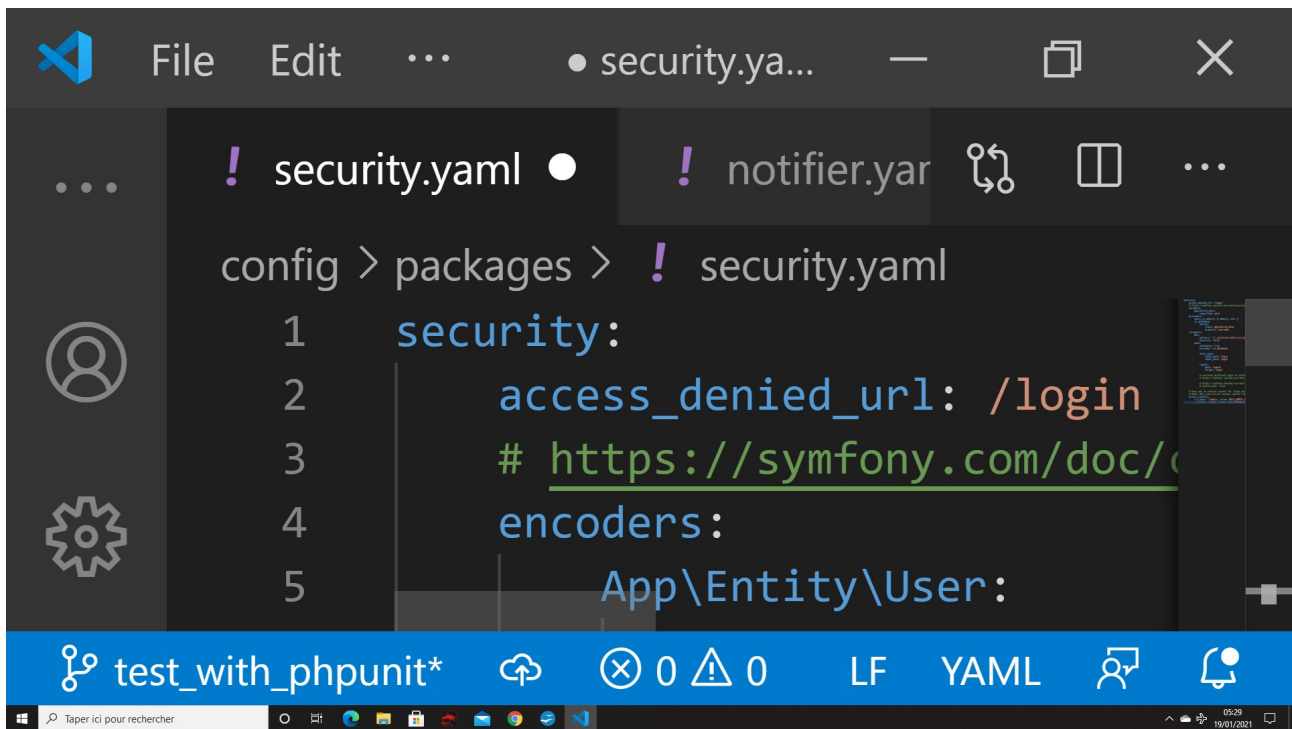
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 2: symfony

[Web Server] Jan 18 20:24:19 | INFO | SERVER GET (200) /_wdt/907a3e ip="127.0.0.1"

Ln 39, Col 71 Spaces: 4 UTF-8 LF YAML

E) Le paramètre security

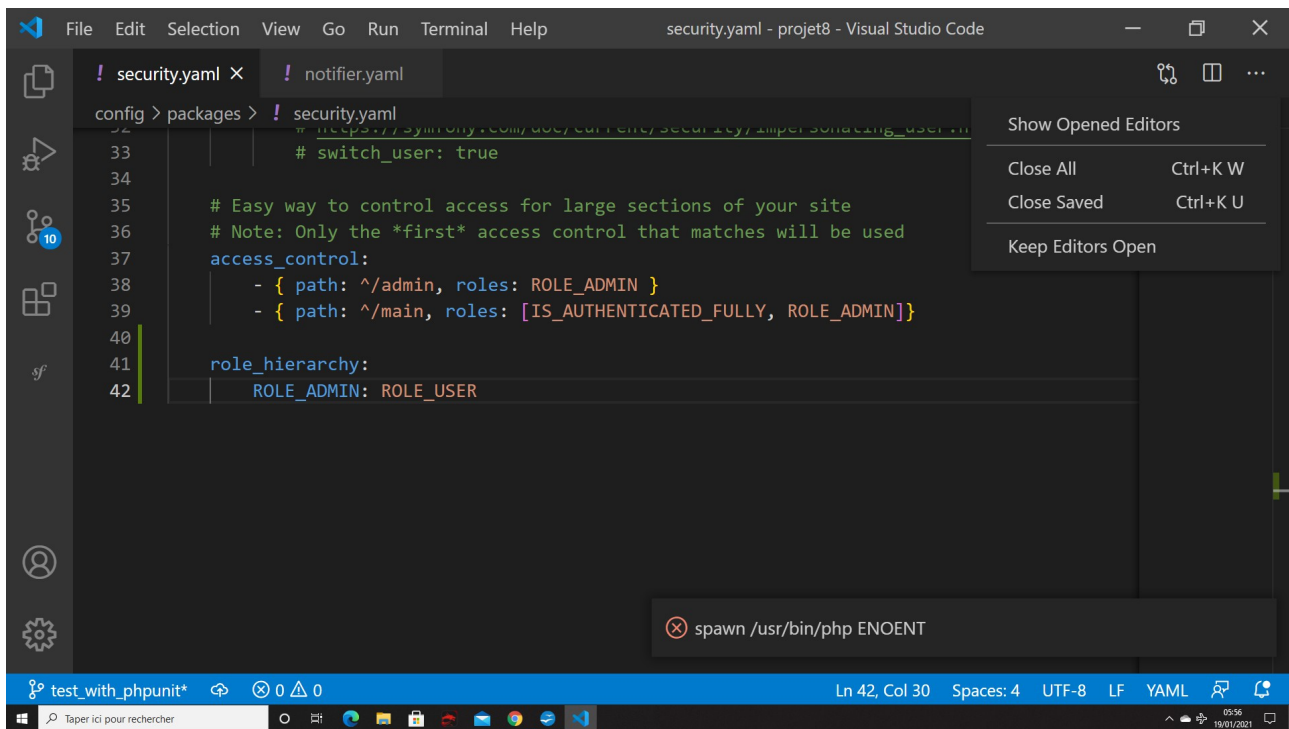
Ci dessous, dans le paramètre sécurité nous avons défini l'`access_denied_url` vers le chemin `/login`. Cela signifie que si un utilisateur tente d'accéder à une route à laquelle son rôle ne lui donne pas accès, il sera redirigé vers la route login, donc sur la page de connexion. Cela nous permet donc de désigner la route vers laquelle doivent être redirigés ceux qui tentent d'accéder à des parties du site auxquelles leur rôle ne leur donne pas le droit d'accès.



```
config > packages > security.yaml
1  security:
2      access_denied_url: /login
3      # https://symfony.com/doc/...
4      encoders:
5          App\Entity\User:
```

F) Les rôles Hierarchy

On aurait aussi pu mettre en place un rôle hierarchy. Il s'agit d'un paramétrage permettant à un utilisateur ayant certain rôle d'en hériter un ou plusieurs autres. Par exemple, un utilisateur ayant rôle admin pourrait, par ce moyen, hériter aussi du rôle user, et par conséquent aussi tous les accès de ce dernier, comme le montre l'image ci dessous à partir de la ligne 41:



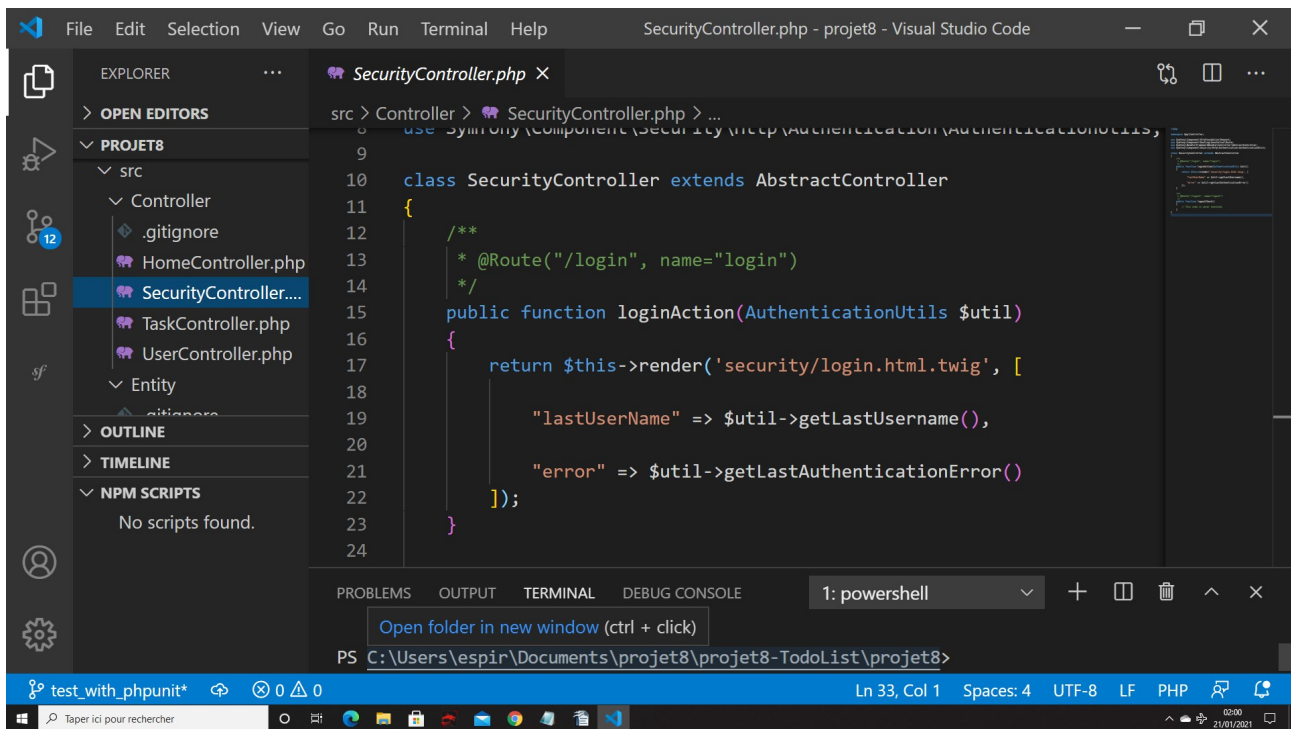
```
config > packages > ! security.yaml
33 # switch_user: true
34
35 # Easy way to control access for large sections of your site
36 # Note: Only the *first* access control that matches will be used
37 access_control:
38   - { path: ^/admin, roles: ROLE_ADMIN }
39   - { path: ^/main, roles: [IS_AUTHENTICATED_FULLY, ROLE_ADMIN]}
40
41 role_hierarchy:
42   ROLE_ADMIN: ROLE_USER
```

spawn /usr/bin/php ENOENT

III) La méthode pour l'authentification.

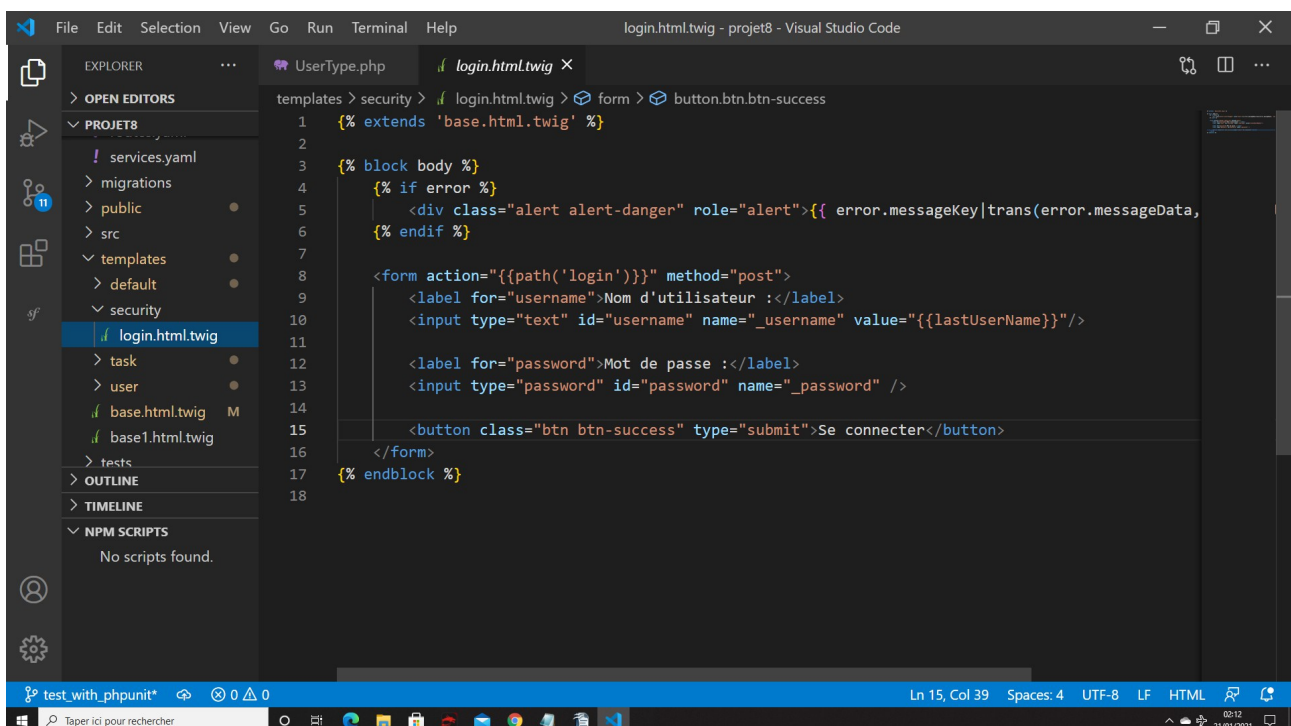
Une méthode pour vérifier les informations entrées par l'utilisateur lors de la connexion est nécessaire, cette dernière se retrouve dans le fichier

App/controller/SecurityController. Dans le cadre de ce projet il s'agit de la méthode loginAction. Elle renverra un message d'erreur sur la page du formulaire de connexion si les identifiants renseignés par l'utilisateur qui souhaite se connecter sont erronés.



IV) La vue.

Le formulaire de connexion est accessible dans le dossier templates/security, il s'agit du fichier login.html.twig illustré par la figure ci dessous :



C'est ce dernier fichier que vous devez modifier si vous

souhaitez modifier l'aspect du formulaire de connexion.