Authentication documentation

Created for the project « To Do List » on Symfony 5.1

Authentication of the To Do List project

What is the objective of this documentation

This documentation has for objective to understand how the authentication of the project «To Do List » works.

What do we need to know

The authentication of this project is linked to the version 5,1 of Symfony. The current version used for this project.

It's about the same method used since the version 4 of Symfony.

Summary

1. Symfony Security bundle

First off, check if the bundle is correctly installed.

1- How to check if the bundle is installed :

Run the next command-line "symfony console debug:autowiring security"

If you find some component of security, so he is well installed. However, if it is not you have to install it.

2.1-How to install the bundle with composer

Run the command-line "composer require symfony/security-bundle"

Then you will find it in the « composer.json » file. The line "symfony/security-bundle " with the current version.

2.2- Make sure that the bundle is enabled

- Open the « config/bundle.php » file.

- Insert the next line: Symfony\Bundle\SecurityBundle\SecurityBundle::class => ['all' => true],

2. User entity

1. Create your User class

At first, we must create a simple User class without worrying about authentication configuration.
For now, its unique job is to manage and stock some informations in the database about the User like every entity and not more.

We insert the next property : « id », « name », « email », « password »

And theirs methods « getter » and « setter »

2. Update our User class for authentication

This time we will adapt our User class for the authentification. After upgrading it, this will allow to use this same class to be authenticated with Symfony authentication

1- Implement the UserInterface in our User class.

2- User class have to got « username » and « password » properties

So let's change « name » to « username » for the property and the methods

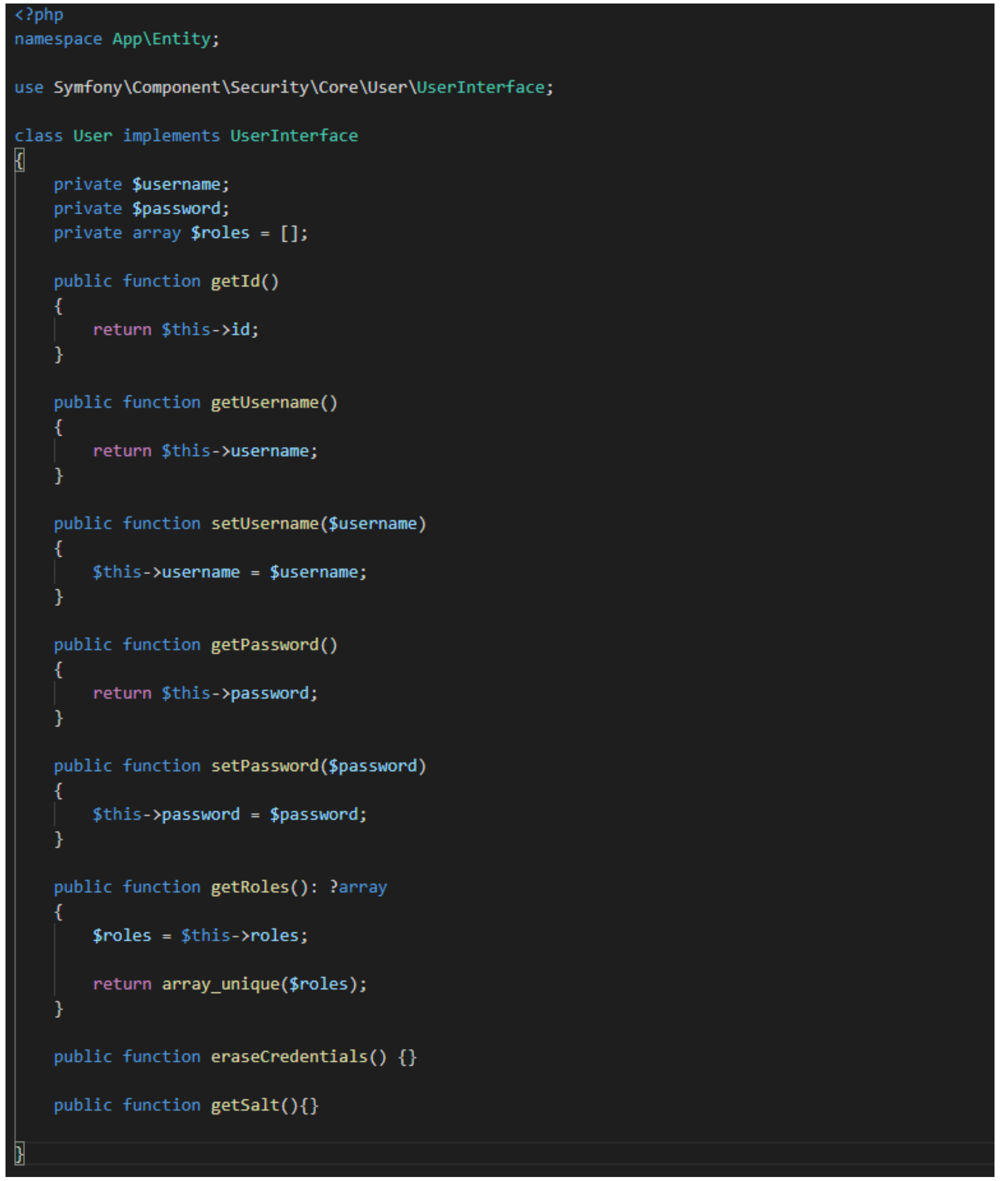
3- Add the next method « getSalt », « eraseCredentials » Let's them return null.

Add the last method « getRoles » who have to return an array type with some role like ['ROLE_USER']

Now, this configuration returns « ROLE_USER » everytime.

But for this project, we have a « ROLE_USER » and « ROLE_ADMIN » role.

We so decided to return the roles property value directly from our database.
This role will be managed from the « user form » when an admin navigate directly on the website.



```php
<?php
namespace App\Entity;

use Symfony\Component\Security\Core\User\UserInterface;

class User implements UserInterface
{
    private $username;
    private $password;
    private array $roles = [];

    public function getId()
    {
        return $this->id;
    }

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

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

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

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

    public function getRoles(): ?array
    {
        $roles = $this->roles;

        return array_unique($roles);
    }

    public function eraseCredentials() {}

    public function getSalt(){}

}
```

3. Security Component configuration

Let's manage all configurations « config/packages/security.yaml » file

1 . Encoding password

It means, when a password come from User entity, call the Password Encoder of symfony and use the « **bcrypt** » algorythm.

2. User provider

The main job of the User provider is to manage user data and save session.

We choose the name « in_database ». It's doesn't matter.
Then ask it to give us the user from the User entity.
For this, use the « username » property.

*You can choose the property you want. You could for example pick « **email** » Make sure this property is unique when you save an user.*

3. Firewall

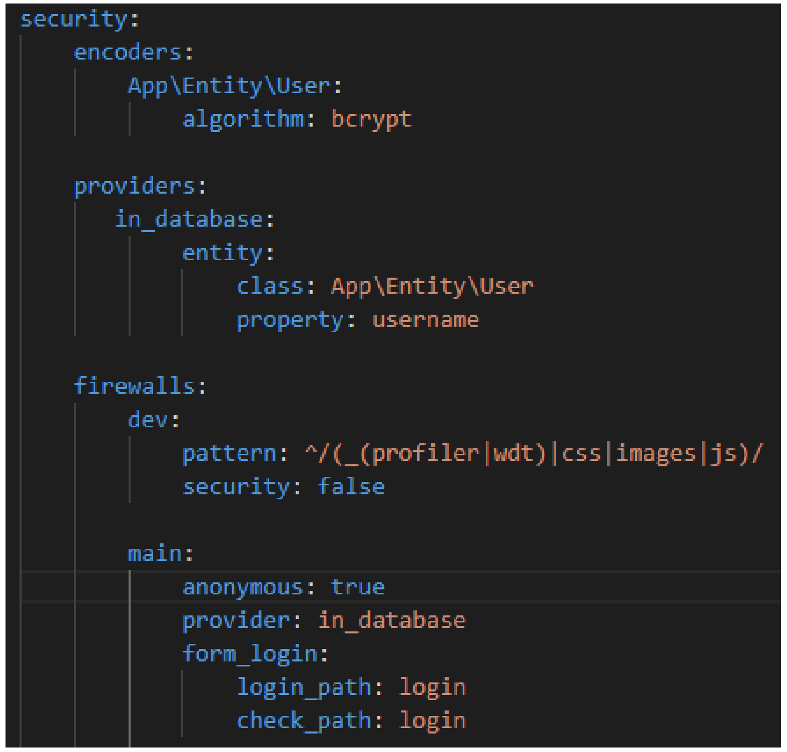
We can create multiple firewalls as needed

- The **« dev »** firewall is a fake. it makes sure that you don't accidentally block Symfony's dev tools which live under URLS see in the picture.

- For the others firewall like **« main »**.They have all their own authentication system.

The main firewall say :

- At first on the website, be logged as anonymous.
- Use our provider « in_database » created earlier.
- Use the path « login » to manage and check the login reference to method path.



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

    providers:
      in_database:
            entity:
                class: App\Entity\User
                property: username

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

        main:
            anonymous: true
            provider: in_database
            form_login:
                login_path: login
                check_path: login
```

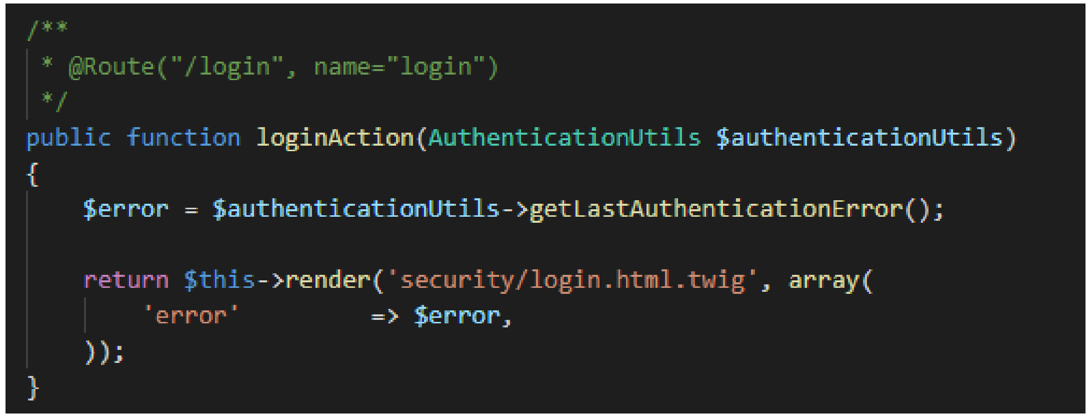
4. Method and form login

Create the login method inside a controller

- The path « login », is the path indicated in the « security.yaml » file

- He returns the page where the login form must be.

- And returns an error message if the submit form fails



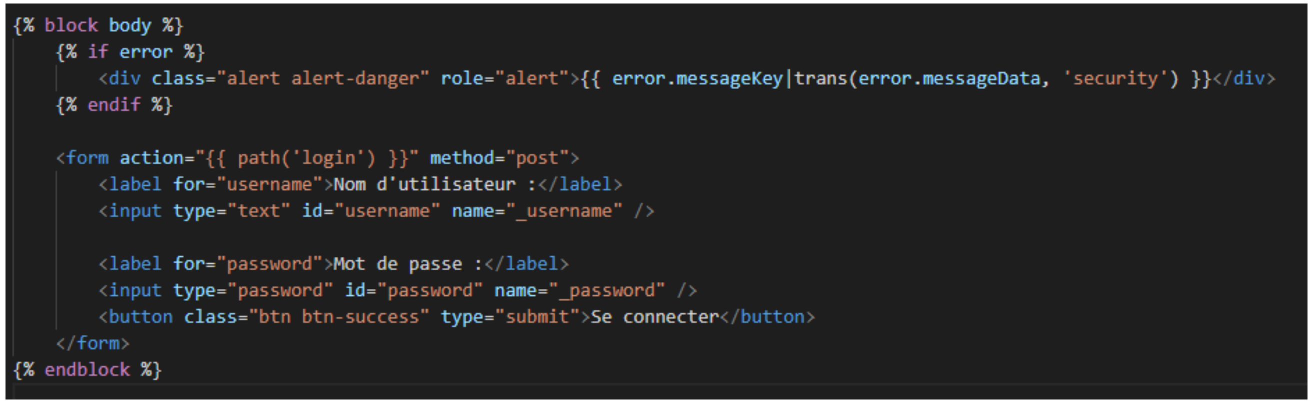
```php
/**
 * @Route("/login", name="login")
 */
public function loginAction(AuthenticationUtils $authenticationUtils)
{
    $error = $authenticationUtils->getLastAuthenticationError();

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

Create the login form

- Show errors

- Show the login form



```twig
{% block body %}
    {% if error %}
        <div class="alert alert-danger" role="alert">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
    {% endif %}

    <form action="{{ path('login') }}" method="post">
        <label for="username">Nom d'utilisateur :</label>
        <input type="text" id="username" name="_username" />

        <label for="password">Mot de passe :</label>
        <input type="password" id="password" name="_password" />
        <button class="btn btn-success" type="submit">Se connecter</button>
    </form>
{% endblock %}
```

You must know, that the fields must have the following « name » attributes .

- « _username » for the login *(which ever field used inside the « security.yaml » file)*
- « _password » for the password

You can configure and choose the « name » attributes inside the « security.yaml » file

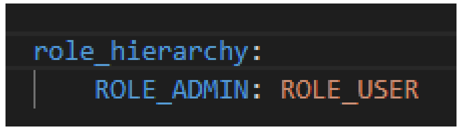
5. Roles

By default, the users are connected as anonymous. Once connected, they will use their own role registered in the database as seen before.
We configured it inside the « security.yaml » file.

1. Hierarchical roles

We managed our own hierarchical roles.

Users with the « ROLE_ADMIN » role will also have the « ROLE_USER » role.

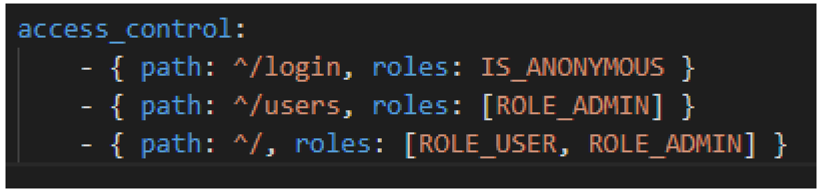


```yaml
role_hierarchy:
    ROLE_ADMIN: ROLE_USER
```

2. Access control

« Acces_control » allows depending on the role to authorize access to different paths

- Anonymous role can access to the login page
- Admin role can access all page starting by /users
- Admin and User can access all pages



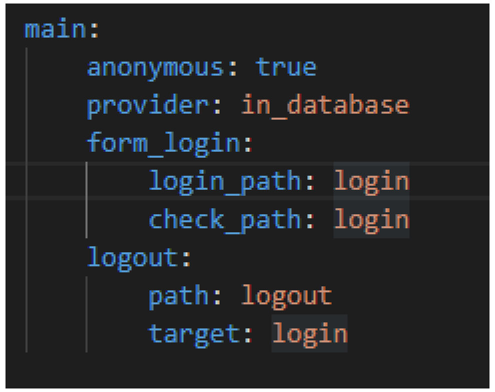
```yaml
access_control:
    - { path: ^/login, roles: IS_ANONYMOUS }
    - { path: ^/users, roles: [ROLE_ADMIN] }
    - { path: ^/, roles: [ROLE_USER, ROLE_ADMIN] }
```

6. Logging out

How to configure logging out

Inside our « main » firewall in the « security.yaml » file.

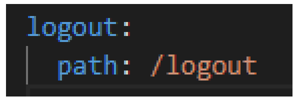
- Define which path used to disconnect and destroy the session
- Define redirection of the user after logout.



```yaml
main:
    anonymous: true
    provider: in_database
    form_login:
        login_path: login
        check_path: login
    logout:
        path: logout
        target: login
```

Create logout path

We need to create only the logout path. For this, create it inside the « config/routes.yaml » file.



```yaml
logout:
  path: /logout
```