Bucket-List : utilisateurs

Fiche TP du module 8

Solution

- Créer une entité User avec la commande `make:user`

```
λ php bin/console make:user

 The name of the security user class (e.g. User) [User]:
 >

 Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
 >

 Enter a property name that will be the unique "display" name for the user (e.g. email, username,
 uuid) [email]:
 > username

 Will this app need to hash/check user passwords? Choose No if passwords are not needed or will b
e checked/hashed by some other system (e.g. a single sign-on server).

 Does this app need to hash/check user passwords? (yes/no) [yes]:
 >

 created: src/Entity/User.php
 created: src/Repository/UserRepository.php
 updated: src/Entity/User.php
 updated: config/packages/security.yaml


  Success!


 Next Steps:
   - Review your new App\Entity\User class.
   - Use make:entity to add more fields to your User entity and then run make:migration.
   - Create a way to authenticate! See https://symfony.com/doc/current/security.html
```

- Ajouter la propriété `email` (string 180) avec `make:entity`

```
λ php bin/console make:entity User

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > email

 Field type (enter ? to see all types) [string]:
 >


 Field length [255]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/User.php

 Add another property? Enter the property name (or press <return> to stop adding fields):
 >


  Success!


 Next: When you're ready, create a migration with php bin/console make:migration
```

- Modifier la classe User pour ajouter les validateurs et la contrainte0 d'unicité sur email :

```php
<?php

namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
```

```php
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 * @UniqueEntity(fields={"email"}, message="There is already an account with this email")
 * @UniqueEntity(fields={"username"}, message="There is already an account with this username")
 */
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=50, unique=true)
     * @Assert\NotBlank(message="Please provide your username!")
     * @Assert\Length(
     *     min=3,
     *     max=50,
     *     minMessage="Minimum 3 characters please!",
     *     maxMessage="Maximum 50 characters please!"
     * )
     * @Assert\Regex(pattern="/^[a-z0-9_-]+$/i", message="Please use only letters, numbers, underscores and dashes!")
     */
    private $username;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    /**
     * @var string The hashed password
     * @ORM\Column(type="string")
     */
    private $password;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     * @Assert\Email(message="Your email is not valid!")
     */
    private $email;

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

    /**
     * @deprecated since Symfony 5.3, use getUserIdentifier instead
     */
    public function getUsername(): string
    {
        return (string) $this->username;
    }

    public function setUsername(string $username): self
    {
        $this->username = $username;

        return $this;
    }
```

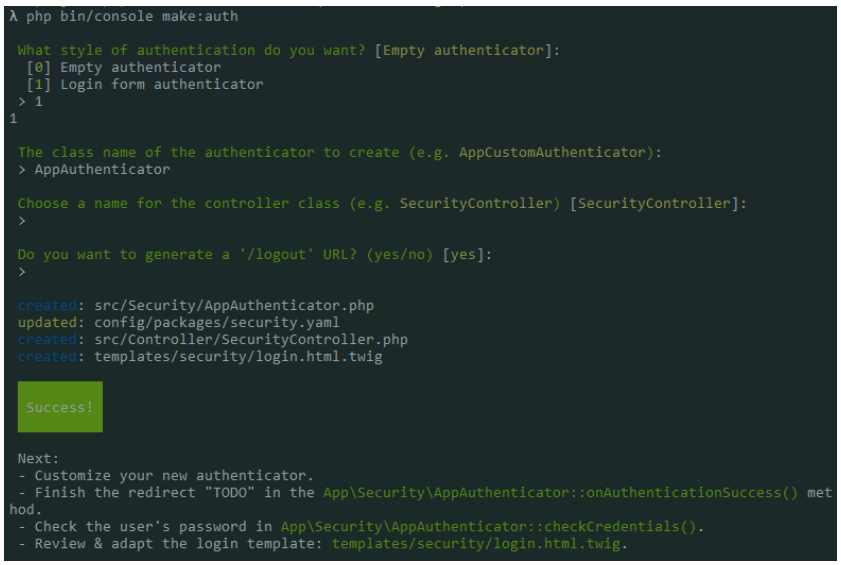
```php
    /**
     * A visual identifier that represents this user.
     *
     * @see UserInterface
     */
    public function getUserIdentifier(): string
    {
        return (string) $this->username;
    }

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

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

        return $this;
    }

    /**
     * Returning a salt is only needed, if you are not using a modern
     * hashing algorithm (e.g. bcrypt or sodium) in your security.yaml.
     *
     * @see UserInterface
     */
    public function getSalt(): ?string
    {
        return null;
    }

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

    public function getEmail(): ?string
    {
```

```
        return $this->email;
    }

    public function setEmail(string $email): self
    {
        $this->email = $email;

        return $this;
    }
}
```

- Mettre à jour la base de données

php bin/console doctrine:schema:update --force

- Générer le système d'authentification avec `make:auth`



- Adapter la classe de l'Authenticator
    - Modifier la fonction onAuthenticationSuccess() :

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    return new RedirectResponse($this->urlGenerator->generate('main_home'));
}
```

    - Surcharger la fonction supports() : nécessaire si on utilise Wamp sans passer par un Virtual Host

```
public function supports(Request $request) : bool
{
    return self::LOGIN_ROUTE === $request->attributes->get('_route')
        && $request->isMethod('POST');
}
```

- Adapter le fichier « templates/security/login.html.twig » : remplacer app.user.username qui est déprécié par app.user.userIdentifier

```
{% if app.user %}
    <div class="mb-3">
        You are logged in as {{ app.user.userIdentifier }}, <a href="{{ path('app_logout') }}">Logout</a>
    </div>
{% endif %}
```

- Générer la page d'inscription avec `make:registration-form`

```
λ php bin/console make:registration-form

 Creating a registration form for App\Entity\User

 Do you want to add a @UniqueEntity validation annotation on your User class to make sure d
ounts aren't created? (yes/no) [yes]:
 >

 Do you want to send an email to verify the user's email address after registration? (yes/n
 > no

 Do you want to automatically authenticate the user after registration? (yes/no) [yes]:
 >

 updated: src/Entity/User.php
 created: src/Form/RegistrationFormType.php
 created: src/Controller/RegistrationController.php
 created: templates/registration/register.html.twig
```

- Adapter la classe RegistrationFormType :

```
$builder
    ->add('username', TextType::class)
    ->add('email', EmailType::class)
    ->add('plainPassword', PasswordType::class, [
        // instead of being set onto the object directly,
        // this is read and encoded in the controller
        'mapped' => false,
        'attr' => ['autocomplete' => 'new-password'],
        'constraints' => [
            new NotBlank([
                'message' => 'Please enter a password',
            ]),
            new Length([
                'min' => 6,
                'minMessage' => 'Your password should be at least {{ limit }} characters',
                // max length allowed by Symfony for security reasons
                'max' => 4096,
            ]),
        ],
    ])
;
```

- Adapter « register.html.twig » en conséquence :

```
{% extends 'base.html.twig' %}

{% block title %}Register{% endblock %}

{% block body %}
    <h1>Register</h1>

    {{ form_start(registrationForm) }}
        {{ form_row(registrationForm.username) }}
        {{ form_row(registrationForm.email) }}
        {{ form_row(registrationForm.plainPassword, {
            label: 'Password'
        }) }}

        <button type="submit" class="btn">Register</button>
    {{ form_end(registrationForm) }}
{% endblock %}
```

- Pour protéger la page de création d'idée, configurer `access_control` dans security.yaml

```
security:
    ...
    access_control:
        - { path: ^/wishes/create$, roles: ROLE_USER }
```

- Pour afficher les liens conditionnellement dans le menu, on va modifier

« base.html.twig » en utilisant la variable `app.user` :

```
...
<header>
    <div class="container">
        <div class="header-top">
            <h1 class="header-logo">
                <a href="{{ path('main_home') }}" title="Go back to homepage">
                    <img src="{{ asset('img/logo.svg') }}" alt="Bucket-List Logo">Bucket-List
                </a>
            </h1>
            <nav class="user-nav">
                {% if app.user %}
                    <a href="{{ path('app_logout') }}" title="Lougout">Logout (Hello {{ app.user.username }})</a>
                {% else %}
                    <a href="{{ path('app_register') }}" title="Create your account">Register</a>
                    <a href="{{ path('app_login') }}" title="Login">Login</a>
                {% endif %}
            </nav>
        </div>
        <nav class="header-nav">
            <a href="{{ path('main_home') }}" title="Go back to homepage">Home</a>
            <a href="{{ path('wish_list') }}" title="All things to do">All wishes</a>
            {% if app.user %}
                <a href="{{ path('wish_create') }}" title="Add your own ideas!">Add yours!</a>
            {% endif %}
            <a href="{{ path('main_about_us') }}" title="About us">About us</a>
        </nav>
    </div>
</header>
...
```

- Modifier la fonction create() dans WishController pour pré-remplir le champ auteur par le pseudo de l'utilisateur connecté :

```
public function create(Request $request, EntityManagerInterface $entityManager): Response
{
    // notre entité vide
    $wish = new Wish();

    //pour préremplir le pseudo dans le formulaire...
    $currentUserUsername = $this->getUser()->getUserIdentifier();
    $wish->setAuthor($currentUserUsername);

    ...
}
```

- Modifier le style dans « app.css » :

```
...

.header-top {
    display: flex;
    justify-content: space-between;
}

.user-nav a {
    margin-left: 1rem;
}

header h1 {
    margin: 0 0 10px 0;
    flex: 1;
}
```