Support de cours : Les Voters en Symfony

1. Introduction aux Voters

Les **voters** en Symfony sont des composants utilisés pour gérer les autorisations avancées. Ils permettent de contrôler l'accès à des actions spécifiques en fonction de la logique métier.

Les voters s'intègrent avec le **système de sécurité** et interviennent lorsque vous utilisez la méthode :

```
$this->isGranted('ACTION', $subject);
```

Exemple d'utilisation :

- Empêcher un utilisateur de modifier un contenu s'il n'en est pas l'auteur.
- Autoriser uniquement les administrateurs à supprimer un élément.

2. Création d'un Voter en Symfony

Étape 1 : Créer un Voter

Générer un voter avec la commande suivante :

```
php bin/console make:voter ContentVoter
```

Cela crée un fichier dans `src/Security/ContentVoter.php`.

Étape 2 : Exemple de Voter

```
namespace App\Security;

use App\Entity\Content;
use Symfony\Bundle\SecurityBundle\Security;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
use Symfony\Component\Security\Core\Authorization\Voter\Voter;

class ContentVoter extends Voter
{
    public const VIEW = 'CONTENT_VIEW';
    public const EDIT = 'CONTENT_EDIT';

    public function __construct(private Security $security) {}

    protected function supports(string $attribute, mixed $subject): bool
    {
        return in_array($attribute, [self::VIEW, self::EDIT]) && $subject
instanceof Content;
    }

    protected function voteOnAttribute(string $attribute, mixed $subject,
TokenInterface $token): bool
    {
        $user = $token->getUser();

        if (!$user) {
            return false; // L'utilisateur doit être connecté
        }

        /** @var Content $content */
        $content = $subject;
```

```
        return match ($attribute) {
            self::VIEW => $this->canView($content, $user),
            self::EDIT => $this->canEdit($content, $user),
            default => false,
        };
    }

    private function canView(Content $content, $user): bool
    {
        return $content->isPublic() || $user === $content->getAuthor();
    }

    private function canEdit(Content $content, $user): bool
    {
        return $user === $content->getAuthor() ||
$this->security->isGranted('ROLE_ADMIN');
    }
}
```

Étape 3 : Utiliser le Voter

Dans un contrôleur :

```
use App\Entity\Content;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Security\Http\Attribute\IsGranted;

class ContentController extends AbstractController
{
    #[IsGranted('CONTENT_VIEW', 'content')]
    public function show(Content $content): Response
    {
        return $this->render('content/show.html.twig', ['content' => $content]);
    }

    public function edit(Content $content): Response
    {
        $this->denyAccessUnlessGranted('CONTENT_EDIT', $content);

        return $this->render('content/edit.html.twig', ['content' => $content]);
    }
}
```

3. Tester un Voter

Avec PHPUnit :

Créer un test unitaire pour vérifier le bon fonctionnement de votre voter.

```
namespace App\Tests\Security;

use App\Entity\Content;
use App\Security\ContentVoter;
use PHPUnit\Framework\TestCase;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;

class ContentVoterTest extends TestCase
{
    public function testUserCanViewPublicContent(): void
    {
        $voter = new ContentVoter();
```

```
        $content = (new Content())->setIsPublic(true);
        $token = $this->createMock(TokenInterface::class);

        $this->assertTrue($voter->vote($token, $content, [ContentVoter::VIEW]));
    }
}
```

4. Meilleures Pratiques

- **Réutiliser les constantes** : Définir des constantes pour vos actions.
- **Limiter les responsabilités** : Chaque voter doit gérer un seul sujet.
- **Cohérence** : Utilisez `denyAccessUnlessGranted()` pour déclencher les vérifications d'accès.

5. Conclusion

Les **voters** sont un mécanisme puissant pour gérer des permissions complexes en Symfony. Ils vous permettent de centraliser la logique d'autorisation et d'améliorer la sécurité et la lisibilité de votre application.