Guide de Sécurité pour le Développement Symfony

Introduction

Ce guide fournit une vue d'ensemble des mesures de sécurité implémentées et recommandées pour notre application Symfony. Il vise à assurer une base solide pour la sécurité et à guider les développeurs dans les bonnes pratiques à adopter.

Mesures de Sécurité Implémentées

Sécurisation des Routes

Les routes sont définies avec précision via des attributs, clarifiant les intentions et limitant les accès non autorisés.

Exemples :

ArticleController.php

```php
/**…
#[Route('/article/{slug}', name: 'article_show')]
public function show(Article $article): Response
{…
}

/**…
#[Route('/article/edit/{slug}', name: 'article_edit')]
public function edit(Request $request, Article $article, EntityManagerInterface $entityManager): Response
{…
}
```

Validation et Assainissement des Entrées

Les types de formulaire Symfony (ArticleType) sont utilisés pour la validation, et SluggerInterface pour la création sécurisée des slugs, assurant l'assainissement des données.

Exemple :

ArticleController.php

```php
#[Route('/article/new', name: 'article_new')]
public function new(Request $request, EntityManagerInterface $entityManager, SluggerInterface $slugger):
Response
{
    if (!$this->isGranted('ROLE_ADMIN')) {
        throw new AccessDeniedException('Seuls les administrateurs peuvent créer des articles.');
    }

    $article = new Article();
    $form = $this->createForm(ArticleType::class, $article);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $slug = $slugger->slug($article->getTitle()) . '-' . uniqid();
        $article->setSlug($slug);

        $entityManager->persist($article);
        $entityManager->flush();

        return $this->redirectToRoute('article_show', ['slug' => $article->getSlug()]);
    }

    return $this->render('article/new.html.twig', [
        'articleForm' => $form->createView(),
    ]);
}
```

Protection CSRF

Les tokens CSRF sont utilisés pour sécuriser les formulaires contre les attaques par Cross-Site Request Forgery, particulièrement pour les actions de suppression.

Exemple :

ArticleController.php

```php
#[Route('/article/delete/{slug}', name: 'article_delete')]
public function delete(Request $request, Article $article, EntityManagerInterface $entityManager): Response
{
    if (!$this->isGranted('ROLE_ADMIN')) {
        throw new AccessDeniedException('Seuls les administrateurs peuvent supprimer les articles.');
    }

    if ($this->isCsrfTokenValid('delete' . $article->getId(), $request->request->get('_token'))) {
        $entityManager->remove($article);
        $entityManager->flush();
    }

    return $this->redirectToRoute('article_index');
}
```

show.html.twig

```twig
<form action="{{ path('article_delete', {'slug': article.slug}) }}" method="post" onsubmit="return
confirm('Êtes-vous sûr?');">
    <input type="hidden" name="_token" value="{{ csrf_token('delete' ~ article.id) }}">
    <button type="submit">Supprimer</button>
</form>
```

Gestion Sécurisée du Contenu HTML

L'utilisation de |raw dans Twig est limitée aux contenus sûrs et nettoyés préalablement grâce à htmlspecialchars (Article entity) pour éviter les vulnérabilités XSS.

Exemple :

Article.php

```php
public function setTitle(string $title): static
{
    $this->title = htmlspecialchars($title);

    return $this;
}

public function getContent(): ?string
{...
}

public function setContent(string $content): static
{
    $this->content = htmlspecialchars($content);

    return $this;
}
```

index.html.twig

```twig
{% for article in articles %}
    <h2>
        <a href="{{ path('article_show', {'slug': article.slug}) }}">{{ article.title|raw }}</a>
    </h2>
    <div>{{ article.content|raw }}</div>
{% else %}
    <p>Aucun article trouvé.</p>
{% endfor %}
```

Recommandations pour la Sécurité

Utilisation Prudente de |raw Employez |raw uniquement avec du contenu dont la sécurité est garantie, et préférez l'échappement automatique de Twig pour le reste.