Mesures de sécurité intégrées à mon projet

Lorsque mes jambons sont affichés, le slug est composé du titre suivi de 'uniqid()' qui génère un identifiant unique, basé sur la date et heure en microsecondes. Je peux donc créer plusieurs jambons qui portent le même nom sans risquer de générer des slugs identiques.

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

Pour contrer les attaques XSS, je mets en place '|raw' couplé au 'htmlspecialchars()' pour échapper les caractères sensibles HTML et éviter des injections HTML, CSS, JS.

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

        return $this;
    }
```

```twig
{% for article in articles %}
<a href="{{ path('article_show', {slug: article.slug}) }}">
{{article.title|raw}}
</a>
```

Pour contrer les attaques CSRF, je mets en place la génération d'un token CSRF pour éviter que les paramètres de requête soient prévisibles et ainsi éviter de compromettre les données et fonctionnalités de mon application web.

```php
#[Route('/ham/delete/{id}', name: 'ham_delete')]
public function delete(Request $r, EntityManagerInterface $em, Ham $ham)
{
    if($this->isCsrfTokenValid('delete'.$ham->getId(), $r->request->get('csrf'))){
        $em->remove($ham);
        $em->flush();
    }
```

```twig
<input type="hidden" name="csrf" value="{{ csrf_token('delete' ~ ham.id) }}">
```

Pour éviter de faire des requêtes en brut je décommente le code suivant :

```
23
24      //      /**
25      //       * @return Ham[] Returns an array of Ham objects
26      //       */
27      public function findByExampleField($value): array
28      {
29          return $this->createQueryBuilder('h')
30              ->andWhere('h.exampleField = :val')
31              ->setParameter('val', $value)
32              ->orderBy('h.id', 'ASC')
33              ->setMaxResults(10)
34              ->getQuery()
35              ->getResult()
36          ;
37      }
```

Gestion des rôles

- ROLE_USER

J'ai fait en sorte que le formulaire de création d'un jambon ne soit disponible que lorsque le USER est connecté.

```
    public function getRoles(): array
    {
        $roles = $this->roles;
        $roles[] = 'ROLE_USER';

        return array_unique($roles);
    }
```

```
{% if is_granted('ROLE_USER') %}
<h2>Ajouter votre jambon perso !</h2>
{{form(form)  }}
{% else %}
    <div>
    <p>Crée toi un compte pour créer ton jambon !</p>
```

- ROLE_ADMIN

Le rôle admin est attribué lorsque le mot de passe entré dans /register est égal à 'admin1'.

Lorsque l’on se log in en tant qu’admin, nous pouvons supprimer et modifier chaque jambon créé.

```
{% if is_granted('ROLE_ADMIN') %}
  <form action="{{ path('ham_delete', {'id': ham.id}) }}" method="post">
    <input type="hidden" name="csrf" value="{{ csrf_token('delete' ~ ham.id) }}">
    <input type="submit" value='DELETE'>
    </form>

  <a href="{{ path('ham_edit', {'slug': ham.slug}) }}">Edit</a>
  {% endif %}
```

```
if (password_verify("admin1", $hashedPassword)) {
    $user->setRoles(['ROLE_ADMIN']);
}
```