Document technique des mesures mis en place pour sécuriser le site

Contrer les Attaques XSS

Vérifier les formulaires coté serveur pour échapper les caractères sensibles HTML. Dans notre cas en PHP, on va utiliser *htmlspecialchars.* On l'affecte soit dans les setters soit dans les getters mais pas les

```php
public function setPrice(float $price): static
{
    $this->price = htmlspecialchars($price);

    return $this;
}
```

Ne pas oublier pas de mettre des */raw* si on veut visualiser les données côté navigateur pour désactiver l'échappement automatique des caractères spéciaux On couple toujours les *htmlspecialchars* et les */raw.*

```twig
<div class="main">
    <h1>Hello {{ cheese.name|raw }}!</h1>
    <p>Hello {{ cheese.description|raw }}!</p>

    <a href="{{ path('app_cheese') }}">Back to the list</a>
</div>
```

Attaques CSRF

Consiste à compromettre les données et fonctionnalités d'une application web.
Pour contrer cela, on des jetons CSRF (valeurs uniques générées aléatoirement côté serveur et envoyées au client).

Dans le CheeseController, rajouter la méthode isCsrfTokenValid() pour générer un token

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

    return $this->redirectToRoute( route: 'app_cheese' );

}
```

Ensuite dans tous les formulaires on ajoute un <input type :'hidden'> avec les informations suivantes.

```
<form action="{{ path('app_cheese_delete', {'id': cheese.id}) }}" method="post">
    <input type="hidden" name="csrf" value="{{ csrf_token('delete' ~ cheese.id) }}"
    <button type="submit">Supprimer</button>
</form>
```

Vérifier bien que dans l'inspecteur d'éléments apparaisse le token dans la value.

```
<input type="hidden" name="csrf" value="2a76fd6d564b8841.ry
9jkI9H77yi53G_-4YeqLHCJMPrbYaW9LRCSTom81M.1R8G3b4vjPv9iiWMi
v990IWXVKC8K7fXpe4EfXVXmzTOYVLAvgSjxJGGHg">
```

Injections SQL

Consiste à envoyer une requête SQL non autorisée.
Pour contrer cela, on définit des droits limités aux utilisateurs.

Dans notre cas, on définit un rôle admin qui peut ajouter dans *packages/security.yaml*, supprimer ou ajouter des fromages.

```
access_control:
     - { path: ^/admin, roles: ROLE_ADMIN }
     - { path: ^/, roles: PUBLIC_ACCESS}
```

On peut aussi utiliser des requêtes préparées. Étant donné que les valeurs des paramètres sont traitées séparément de la structure de la requête, il est beaucoup plus difficile pour un attaquant d'injecter du code SQL malveillant dans une requête préparée.

On décommente donc le script suivant dans notre CheeseRepository

```
//    /**
//     * @return Cheese[] Returns an array of Cheese objects
//     */
//    public function findByExampleField($value): array
//    {
//        return $this->createQueryBuilder('c')
//            ->andWhere('c.exampleField = :val')
//            ->setParameter('val', $value)
//            ->orderBy('c.id', 'ASC')
//            ->setMaxResults(10)
//            ->getQuery()
//            ->getResult()
//        ;
//    }
```

```
/**
 * @return Cheese[] Returns an array of Cheese objects
 */
public function findByExampleField($value): array
{
    return $this->createQueryBuilder( alias: 'c')
        ->andWhere( ...where: 'c.exampleField = :val')
        ->setParameter( key: 'val', $value)
        ->orderBy( sort: 'c.id', order: 'ASC')
        ->setMaxResults( maxResults: 10)
        ->getQuery()
        ->getResult()
    ;
}
```

Autre pratiques

Slug

Mettre des slug au lieu des id dans les routes pour réduire le risque d'attaque par force brute ou encore limiter les attaques par manipulation d'URL

Exemple dans notre CheeseController :

```php
#[Route('/show/{slug}', name: 'cheese_show')]
public function show(Cheese $cheese)
{
    return $this->render( view: 'cheese/show.html.twig', [
        'cheese' => $cheese
    ]);
}
```

On ajoute donc bien évidemment le getter et le setter dans notre entité Cheese.

```php
public function getSlug(): ?string
{
    return $this->slug;
}

public function setSlug(string $slug): static
{
    $this->slug = $slug;

    return $this;
}
```

Hasher le mot de passe

Dès que le formulaire est validé on hashe le mot de passe.

```php
if ($form->isSubmitted() && $form->isValid()) {
    // encode the plain password
    $user->setPassword(
        $userPasswordHasher->hashPassword(
            $user,
            $form->get('plainPassword')->getData()
        )
    );
```