Rendu sécuriser un site web

Thibaut François A2 - CDI

Injection SQL

- Pour contrer les injection SQL nous utilisons plusieurs rôles pour empêcher l'accès de l'entièreté de la base de données à n'importe quel utilisateur.

id	username	roles
1	tibo	["ROLE_ADMIN"]
2	alpha	[]

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

Attaques XSS

- Pour réduire les risques d'attaques XSS nous avons choisis htmlspecialchars et la methode « |raw ». Ces outils nous permettent d'échapper les balise HTML et donc éviter qu'un utilisateur malveillant est accès au contenu de la page et puisse me modifier.

```
public function setName(string $name): static
{
    $this->name = htmlspecialchars($name);

    return $this;
}
```

```
<h3>Ship name: {{ship.name|raw}}</h3>
<p>Owner name: {{ship.owner|raw}}</p>
<p>Ship Brand: {{ship.brand|raw}}</p>
<p>Ship Registration nb: {{ship.registrationNumber|raw}}</p>
```

Attaques CSRF

- Contre les attaques CSRF deux méthodes sont utilisés. La première, est les jetons CSRF, ils sont générés aléatoirement coté serveur donc non falsifiable pour un utilisateur. La seconde, est l'attribut SameSite dans les cookies, cela permet de s'assurer que les informations envoyées par l'utilisateur proviennent bien de notre site web.

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

```twig
<form action="{{path('app_ship_delete', {id: ship.id})}}" method="POST">
    <input type="hidden" name="csrf" value="{{csrf_token('delete'~ship.id)}}">
    <input type="submit" value="Delete">
</form>
```

Autres bonnes pratiques

- Le mot de passe de l'utilisateur et toujours haché en base de données pour éviter les fuites d'informations.

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

id	username	roles	password
1	tibo	["ROLE_ADMIN"]	$2y$13$pA4GjlWDZkq884j09JFJ3u3WroOiLXfbf12kz0aFjHBvo...
2	alpha	[]	$2y$13$D35xqArJEWlgDWUr3WJDleTVv9sanOk/FXsLBLIL9qq...

- Pour accéder au vaisseaux (ship) individuellement nous utilisons un slug qui contient le nom du vaisseau et un uniqueID. Si le nom de vaisseau change alors le slug en fait autant.

```php
if ($form->isSubmitted() && $form->isValid()) {
    $slug = $slugger->slug($ship->getName()).'-'.uniqid();
    $ship->setSlug($slug);
    $em->persist($ship);
    $em->flush();
```

- Dans le fichier ShipRepository on peut activer des requêtes SQL préparé pour réduire encore le risque d'injection SQL. Dans ce projet cette fonctionnalité n'est pas nécessaire.

```php
29    //        return $this->createQueryBuilder('s')
30    //            ->andWhere('s.exampleField = :val')
31    //            ->setParameter('val', $value)
32    //            ->orderBy('s.id', 'ASC')
33    //            ->setMaxResults(10)
34    //            ->getQuery()
35    //            ->getResult()
36    //        ;
```