PACHEU Quentin

Document technique de sécurité

Document technique détaillant les mesures de sécurité implémentées sur la plateforme de gestion CRUD pour les joueurs de tennis. Une partie des fonctionnalités de sécurité est intégrée nativement par Symfony, tandis que le reste des dispositifs de protection a été ajouté de manière personnalisée.

Gestion des permissions :

Les rôles des utilisateurs sont administrés de manière native par Symfony. L'attribution des autorisations d'accès aux différentes routes s'effectue via la configuration du fichier ‘config/packages/security.yaml’, notamment dans la section ‘access_control’.

```yaml
access_control:
    - { path: ^/edit, roles: ROLE_ADMIN }
    - { path: ^/delete, roles: ROLE_ADMIN }
    - { path: ^/register, roles: PUBLIC_ACCESS }
    - { path: ^/login, roles: PUBLIC_ACCESS }
    - { path: ^/, roles: ROLE_USER }
```

Attaque CSRF :

Pour les formulaires susceptibles de modifier la base de données (comme le formulaire de suppression), l'intégration d'un jeton CSRF est indispensable pour authentifier l'origine du formulaire.

```twig
<form action="{{ path('app_joueur_delete', {'slug': joueur.slug}) }}" method="post" style="display: inline;">
    <input type="hidden" name="_method" value="DELETE">
    <button type="submit" onclick="return confirm('Êtes-vous sûr de vouloir supprimer ce joueur ?')">Supprimer</button>
</form>
```

Injections SQL :

La sécurité du site web contre les injections SQL est prise en charge nativement par Symfony, grâce au fichier Repository/JoueurRepository.php. Tant que ce fichier n'est pas altéré avec des requêtes personnalisées, aucun risque n'est à craindre. Cependant, en cas de personnalisation, il est impératif d'utiliser la méthode ->setParameter() pour assurer une protection adéquate contre les injections SQL.

Protection des IDs :

Par mesure de sécurité, évitez d'afficher les ID en clair dans le code ou dans les URL du site. Pour remédier à cela, nous utilisons un slug, un identifiant unique qui remplace l'ID. Le slug est généré à partir du nom du joueur, sans espaces, accompagné d'un nombre unique pour garantir l'unicité. Cette approche permet d'éviter les conflits entre des titres similaires. Le slug est ensuite utilisable à tout moment dans le code de l'application.

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

    $em->persist($joueur);
    $em->flush();
}
```

```twig
<a href="{{ path('app_joueur_show', {'slug': joueur.slug}) }}">Voir</a>
```

```twig
<a href="{{ path('app_joueur_edit', {'slug': joueur.slug}) }}">Modifier</a>
```

Attaques XSS :

Afin de prévenir les attaques XSS, il est crucial d'incorporer la fonction htmlspecialchars() dans les fonctions Set des pages du fichier Entity. Cette méthode assure l'échappement de tous les caractères HTML potentiellement dangereux. Pour conserver le formatage correct du texte tout en assurant la sécurité, il convient d'ajouter |raw lors de l'affichage du contenu.

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

    return $this;
}
```

```twig
<h2>{{ joueur.name|raw }}</h2>
<h3>{{ joueur.prenom|raw }}</h3>
<p>Nationalité : {{ joueur.nationality|raw }}</p>
<p>Classement : {{ joueur.classement|raw }}</p>
```