RAPPORT PROJET CHUTESHOOT

Objectifs :
L'entreprise Chuteshoot vous sollicite pour développer un site web.Ce dernier comprendra un système de connexion sécurisé donnant accès à une boîte à idées.

Hassan Boucherit BTS SIO 24.1

Sommaire :

Inscription :

```php
<?php
session_start();
```

Démarre une nouvelle session PHP ou reprend une session existante.

Connexion à la base de données

```php
$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);
```

Cette ligne crée un nouvel objet mysqli qui tente de se connecter à la base de données MySQL avec les informations disponibles

```php
$db_host = 'localhost';
$db_user = 'root';
$db_pass = 'root';
$db_name = 'Chuteshoot';
```

Vérification de la connexion à la base de données

```php
if ($conn->connect_error) {
    die("Échec de la connexion : " . $conn->connect_error);
}
```

Cette portion du code vérifie si la connexion à la base de données a été établie avec succès. Si la connexion échoue, le script s'arrête et affiche un message d'erreur.

Fonction anti-XSS

```php
function antixss($input) {
    $new = strip_tags($input);
    $new = htmlspecialchars($new, ENT_QUOTES);
    return $new;
}
```

La fonction `antixss` est définie pour nettoyer les données reçues pour prévenir contre les attaques XSS (Cross-Site Scripting). Elle utilise les fonctions `strip_tags` et `htmlspecialchars` pour supprimer ou encoder les caractères spéciaux dangereux.

Récupération et nettoyage des données du formulaire

```php
$username = antixss($_POST['username']);
$email = antixss($_POST['email']);
$password = antixss($_POST['password']);
$confirm_password = antixss($_POST['confirm_password']);
```

Ici, les données envoyées par l'utilisateur via le formulaire (`username`, `email`, `password`, etc.) sont récupérées et nettoyées en utilisant la fonction `antixss` précédemment définie.

Vérification des mots de passe

```php
if ($password !== $confirm_password) {
    header("Location: inscription.php");
    exit;
}
```

Ce bloc de code compare le mot de passe et la confirmation du mot de passe. S'ils ne correspondent pas, l'utilisateur est redirigé vers le formulaire d'inscription.

Vérification de l'unicité de l'email

```php
$stmt = $conn->prepare("SELECT email FROM Utilisateurs WHERE email = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$result = $stmt->get_result();
```

Cela prépare une requête SQL pour vérifier si l'email fourni existe déjà dans la base de données. Si l'email est trouvé, l'utilisateur est redirigé vers le formulaire d'inscription.

```php
$hash = password_hash($password, PASSWORD_DEFAULT);
```

Avant d'insérer le mot de passe dans la base de données, il est sécurisé en utilisant un hachage. Cela signifie qu'il est transformé en une chaîne de caractères qui ne peut pas être facilement convertie en retour au mot de passe original.

Insertion des données dans la base de données

```php
$stmt = $conn->prepare("INSERT INTO Utilisateurs (Username, Email, Passwordh) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $username, $email, $hash);
```

Cette portion de code prépare une requête SQL pour insérer les données de l'utilisateur (nom d'utilisateur, email, mot de passe haché) dans la table `Utilisateurs`. Les valeurs sont passées à la requête de manière sécurisée en utilisant des paramètres liés pour empêcher les injections SQL.

Gestion du résultat de la requête

En cas de succès de l'insertion, l'ID de l'utilisateur est stocké dans la session et l'utilisateur est redirigé vers la page d'accueil. Si l'insertion échoue, l'utilisateur est renvoyé au formulaire d'inscription.

Ce script est un exemple de base de la manière dont un formulaire d'inscription peut être traité en PHP avec des mesures de sécurité comme la prévention des attaques XSS et l'assurance que les mots de passe sont correctement hachés avant d'être stockés dans une base de données.

Connexion :

Traitement de la connexion
```php
if (isset($_POST['username']) && isset($_POST['password'])) {
    ...
}
```
Cette condition vérifie si les données de nom d'utilisateur (`username`) et de mot de passe (`password`) ont été envoyées par la méthode POST.

Nettoyage des entrées utilisateur
```php
$clean_username = antixss($_POST['username']);
$clean_password = antixss($_POST['password']);
```
Les données reçues sont nettoyées à l'aide de la fonction `antixss` pour éviter les attaques XSS.

Requête de vérification des identifiants
```php
$test = $conn->prepare("SELECT UserID, Passwordh FROM Utilisateurs WHERE Username = ?");
$test->bind_param("s", $clean_username);
...
```
On prépare une requête SQL pour récupérer l'ID utilisateur et le mot de passe haché associé au nom d'utilisateur saisi. Les paramètres liés (`bind_param`) sont utilisés pour se prémunir contre les injections SQL.

Vérification du mot de passe
```php
if ($result->num_rows > 0) {
    $user = $result->fetch_assoc();
    if (password_verify($clean_password, $user['Passwordh'])) {
        ...
    } else {
        ...
    }
} else {
```

    ...
}
```
Si le nom d'utilisateur est trouvé dans la base de données, on compare le mot de passe saisi avec le mot de passe haché stocké grâce à la fonction `password_verify`. Si la vérification est réussie, l'utilisateur est considéré comme authentifié.

### Gestion de la session et redirection
```php
$_SESSION['user_id'] = $user['UserID'];
header("Location: index.php");
...
```
Une fois l'utilisateur authentifié, son ID est stocké dans la session et il est redirigé vers la page d'accueil du site (`index.php`). S'il y a une erreur d'authentification, il est redirigé vers la page de connexion (`login.php`).

Clôture de la connexion
```php
$conn->close();
```
Finalement, la connexion à la base de données est fermée proprement.

Ce script met en œuvre les bonnes pratiques de sécurité telles que la prévention des injections SQL et des attaques XSS, ainsi que la vérification sécurisée des mots de passe. Il assure que les utilisateurs ne peuvent se connecter qu'avec des identifiants valides et stocke leurs informations dans une session pour les utiliser sur d'autres pages.
```