Application JO2024 : Cahier des charges techniques

Sommaire

1. Contexte du projet
    1.1. Présentation du projet
    1.2. Date de rendu du projet
2. Besoins fonctionnels
3. Ressources nécessaires à la réalisation du projet
    3.1. Ressources matérielles
    3.2. Ressources logicielles
4. Gestion du projet
5. Conception du projet
    5.1. Le front-end
        5.1.1. Wireframes
        5.1.2. Maquettes
        5.1.3. Arborescences
    5.2. Le back-end
        5.2.1. Diagramme de cas d’utilisation
        5.2.2. Diagramme d’activités
        5.2.3. Modèles Conceptuel de Données (MCD)
        5.2.4. Modèle Logique de Données (MLD)
        5.2.5. Modèle Physique de Données (MPD)
6. Technologies utilisées
    6.1. Langages de développement Web
    6.2. Base de données
7. Sécurité
    7.1. Login et protection des pages administrateurs
    7.2. Cryptage des mots de passe avec Bcrypt
    7.3. Protection contre les attaques XSS (Cross-Site Scripting)
    7.4. Protection contre les injections SQL

1. Contexte du projet

1.1. Présentation du projet

Votre agence web a été sélectionnée par le comité d'organisation des jeux olympiques de Los Angeles 2028 pour développer une application web permettant aux organisateurs, aux médias et aux spectateurs de consulter des informations sur les sports, les calendriers des épreuves et les résultats des JO 2028.

Votre équipe et vous-même avez pour mission de proposer une solution qui répondra à la demande du client.

1.2. Date de rendu du projet

Le projet doit être rendu au plus tard le 08/11/2024

2. Besoins fonctionnels

Le site web devra avoir une partie accessible au public et une partie privée permettant de gérer les données.

Les données seront stockées dans une base de données relationnelle pour faciliter la gestion et la mise à jour des informations. Ces données peuvent être gérées directement via le site web à travers un espace administrateur.

3. Ressources nécessaires à la réalisation du projet

REPRENDRE RÉPONSES MISSION 1

3.1. Ressources matérielles

- PC Portable
- PC fixe
- Clavier
- Souris
- Ecran
- Unité centrale
-Cable RJ45 relie au PC fixe/ordinateur

3.2. Ressources logicielles

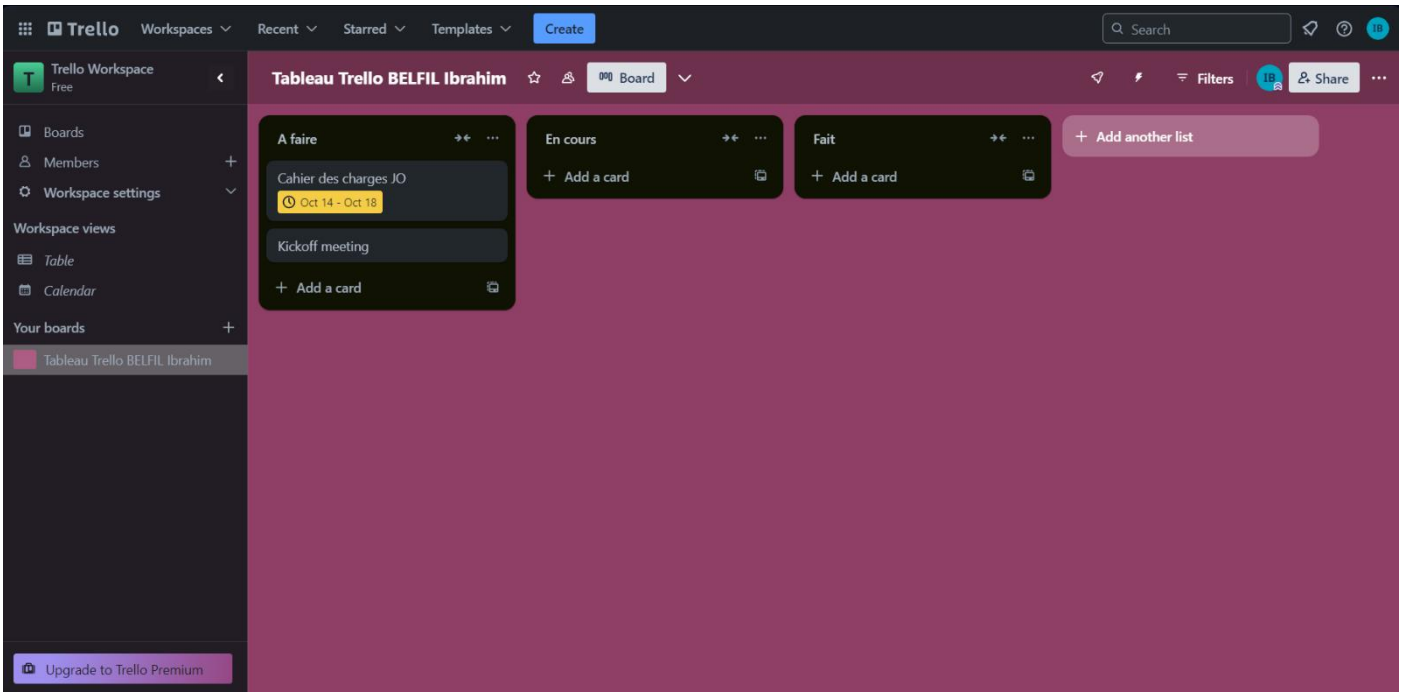
- IDE = Environnement de développement (VS CODE)
- GITHUB = Plateforme de développement collaborative
- APACHE = Serveur web contenue dans MAMP
- SGBDR = MYSQL contenue dans MAMP
- TRELLO = Outil de gestions de projet
- UML et arborescence = Visual paradigm online
- Figma = Maquette
- Mokodo = Conception BDD

3.3. Langages utilisé

-HTML 5
-CSS 3
-JavaScript
-Back end: PHP 8
- BDD: MySQL
- PHP

4. Gestion du projet

Pour réaliser le projet, nous utiliserons la méthode Agile Kanban. Nous utiliserons également l'outil de gestion de projet en ligne Trello.



Nous travaillons également sur GitHub, plateforme de développement collaboratif.

5. Conception du projet

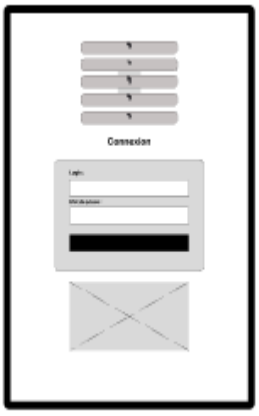
5.1. Le front-end

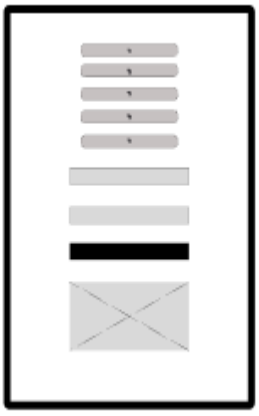
REPRENDRE RÉPONSES MISSION 4

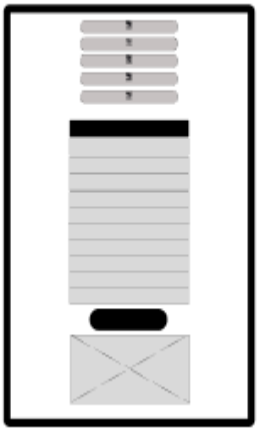
5.1.1. Wireframes

Version téléphone :

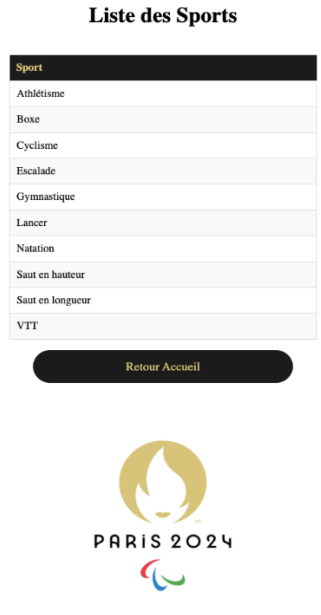
Version tablette :

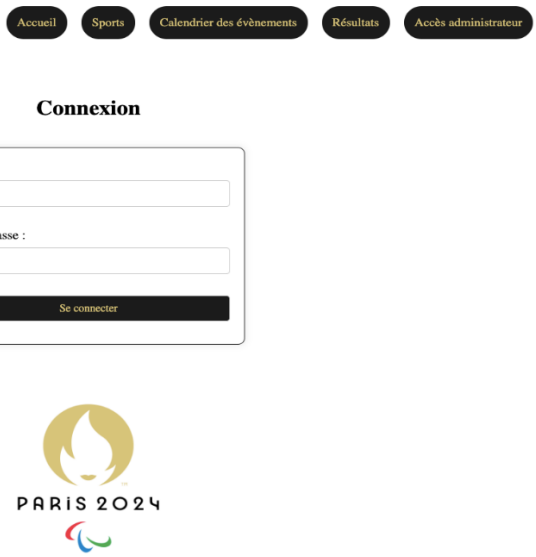






5.1.2. Maquettes





Accueil | Sports | Calendrier des épreuves | Résultats | Accès administrateur

Sports

Calendrier des épreuves

Résultats

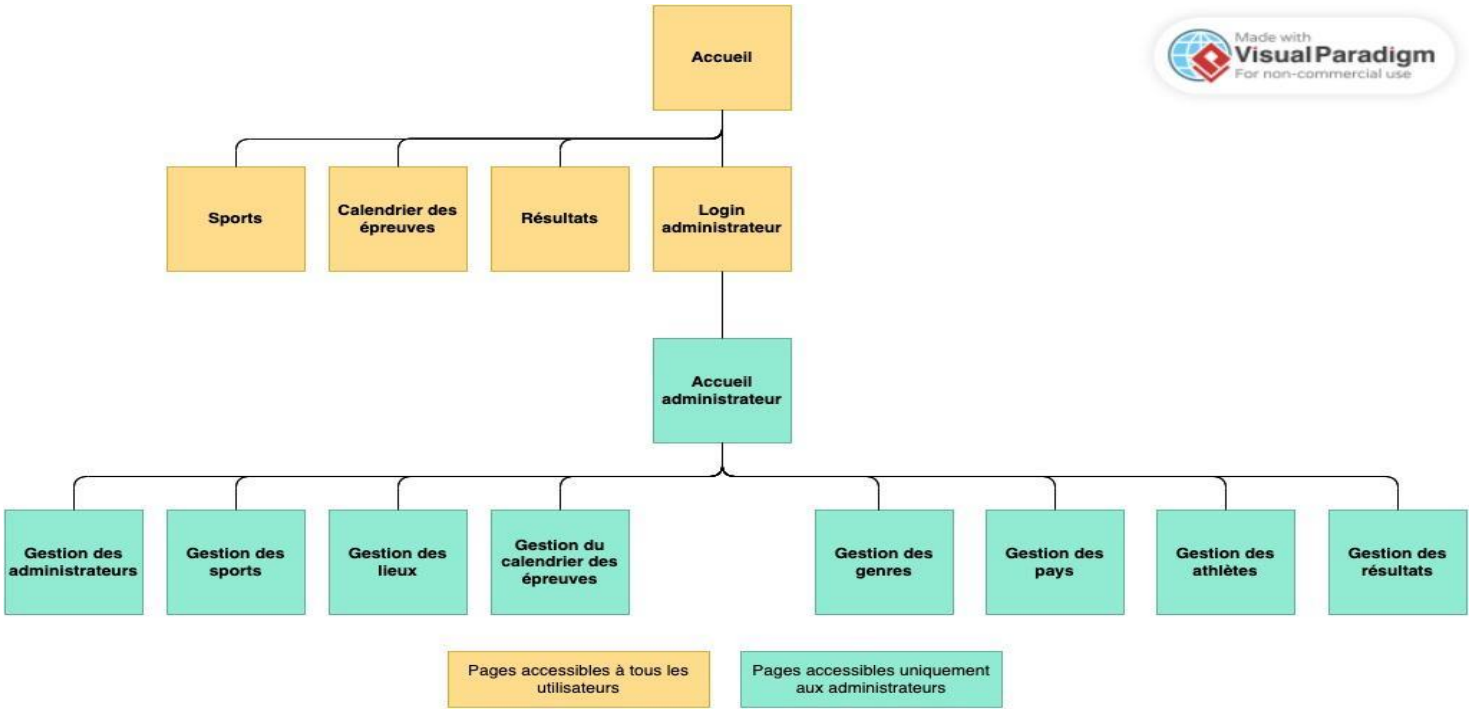
Accueil | Sports | Calendrier des évènements | Résultats | Accès administrateur

Liste des Sports

Sport
Athlétisme
Boxe
Cyclisme
Escalade
Gymnastique
Lancer
Natation
Saut en hauteur
Saut en longueur
VTT

Retour Accueil

5.1.3. Arborescences

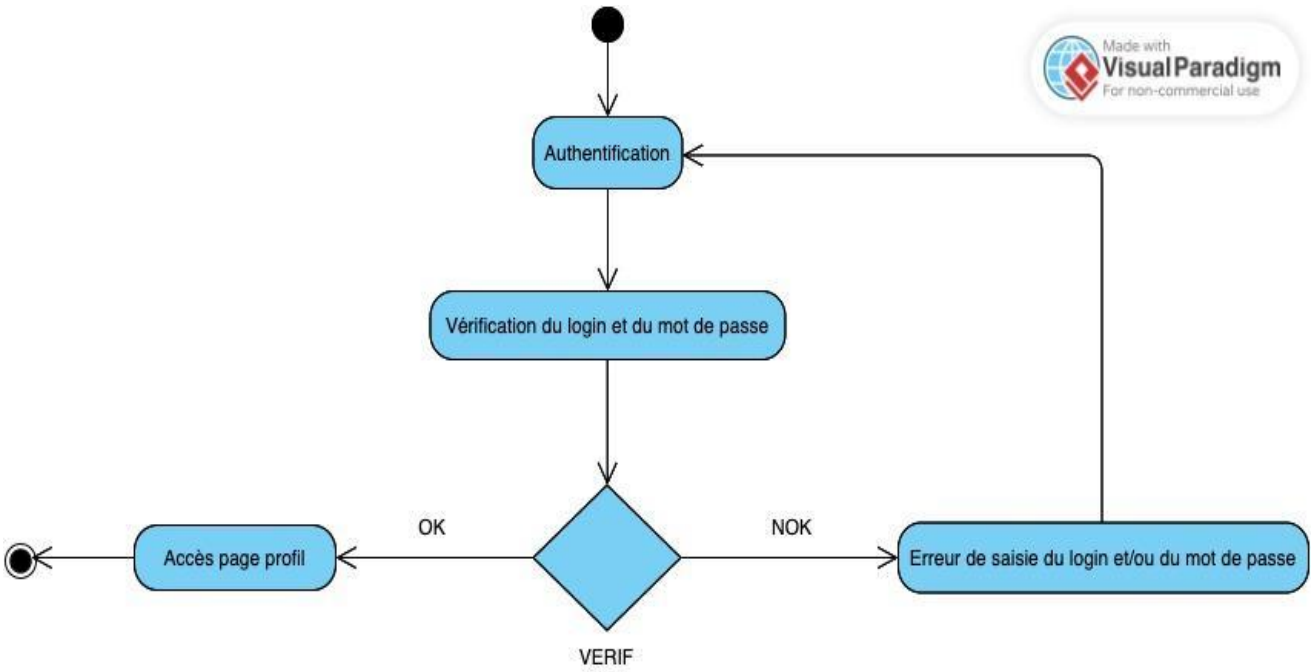


5.2. Le back-end

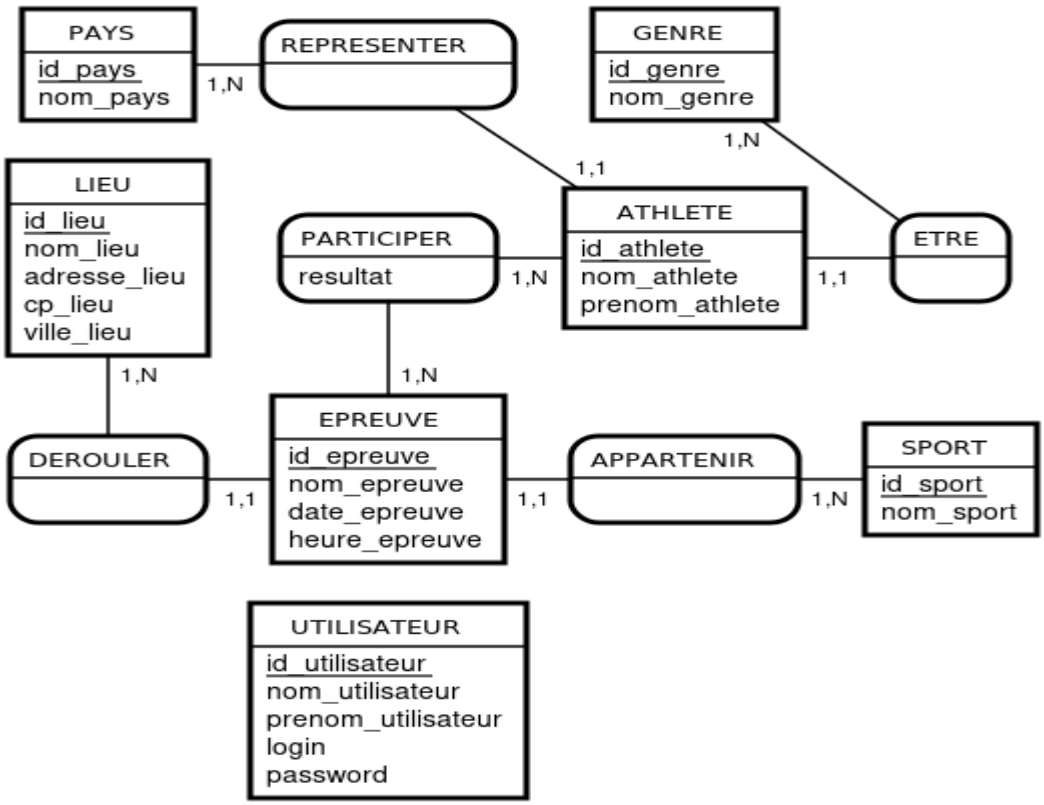
REPRENDRE RÉPONSES MISSIONS 2 ET 3

5.2.1. Diagramme de cas d'utilisation

5.2.2. Diagramme d'activités



5.2.3. Modèles Conceptuel de Données (MCD)



5.2.4. Modèle Logique de Données (MLD)

- ATHLETE (id_athlete, nom_athlete, prenom_athlete, #id_pays, #id_genre)
- EPREUVE (id_epreuve, nom_epreuve, date_epreuve, heure_epreuve, #id_lieu, #id_sport)
- GENRE (id_genre, nom_genre)
- LIEU (id_lieu, nom_lieu, adresse_lieu, cp_lieu, ville_lieu)
- PARTICIPER (#id_athlete, #id_epreuve, resultat)
- PAYS (id_pays, nom_pays)
- SPORT (id_sport, nom_sport)
- UTILISATEUR (id_utilisateur, nom_utilisateur, prenom_utilisateur, login, password)

5.2.5. Modèle Physique de Données (MPD)

6. Technologies utilisées

REPRENDRE RÉPONSES MISSION 1

6.1. Langages de développement Web

Les langages de développement Web utilisées sont :

1. HTML5
2. CSS3
3. PHP8
4. JS
5. SQL pour la base de données

6.2. Base de données

Pour la Base de données il faudra utilisée SQL.

1. Sécurité

7. Sécurité

DÉFINISSEZ (SI POSSIBLE) ET EXPLIQUEZ BRIÈVEMENT VOTRE SOLUTION.
AJOUTEZ SI VOUS LE SOUHAITEZ DES COURS EXTRAITS DE CODE.

7.1. Login et protection des pages administrateurs

Une page de **Login** repose généralement sur la saisie d'un **nom d'utilisateur** et d'un **mot de passe**. Voici les principales techniques que nous pouvons utiliser pour garantir la sécurité et l'intégrité des données lors de la connexion :

1. **Utiliser la commande password_verify()** pour comparer le mot de passe haché stocké dans la base de données avec celui saisi par l'utilisateur. Cela permet de valider le mot de passe de manière sécurisée sans stocker celui-ci en clair.
2. **Utiliser la commande bind_param()** (ou des requêtes préparées avec PDO) pour prévenir les **injections SQL**. Cette méthode permet de lier les paramètres dans les requêtes SQL, évitant ainsi toute injection malveillante.
3. **Utiliser la commande htmlspecialchars()** pour qu'il permet de filtrer les entrées utilisateur. Cette fonction empêche l'exécution de scripts malveillants (comme les attaques XSS), en convertissant les caractères spéciaux en entités HTML.
4. **Utiliser le protocole HTTPS** pour chiffrer les données transmises entre l'utilisateur et le serveur, notamment les informations sensibles comme le nom d'utilisateur et le mot de passe. Cela empêche les attaques de type **man-in-the-middle**.
5. **Mettre en place un fichier de logs** qui enregistre chaque tentative de connexion, qu'elle soit réussie ou échouée. Cela permet de garder une trace de qui s'est connecté, à quel moment et avec quel succès, pour renforcer la sécurité et détecter d'éventuelles activités suspectes.

}

7.2. Cryptage des mots de passe avec Bcrypt

Bcrypt est un algorithme de hachage sécurisé, conçu pour protéger les mots de passe. Au lieu de stocker un mot de passe en clair, Bcrypt le transforme en une chaîne de caractères hachée, difficile à déchiffrer. Cela améliore la sécurité, car même si la base de données est piratée, les mots de passe ne peuvent pas être facilement récupérés. L'une des forces de Bcrypt est qu'il fonctionne de manière volontairement lente, ce qui complique les attaques par force brute (qui consistent à tester de nombreux mots de passe) ou par dictionnaire (qui utilisent des listes de mots de passe courants).

Exemple code :

```
<?php

$mot_de_passe = 'IB92'; // Mot de passe saisi par l'utilisateur lors de l'inscription

$password = password_hash($mot_de_passe, PASSWORD_BCRYPT); // Hachage du mot de passe avec Bcrypt

echo "Mot de passe haché : " . $password; // Affiche le mot de passe haché (ceci est ce qui sera stocké dans la base de données)

echo "Mot de passe haché : " . $password;

// Vérification du mot de passe

if (Bcrypt::check($mot_de_passe_saisi, $hachage_stock)) {
```

```
    echo "Le mot de passe est correct, accès autorisé.";

} else {
    echo "Mot de passe incorrect, accès refusé.";
}

?>
```

7.3. Protection contre les attaques XSS (Cross-Site Scripting)

Les attaques XSS (Cross-Site Scripting) sont des failles de sécurité qui permettent aux hackers d'injecter du code malveillant, souvent du JavaScript, dans une page web. Lorsque d'autres utilisateurs visitent cette page, le script peut voler des informations sensibles comme des identifiants, ou rediriger les utilisateurs vers des sites frauduleux, compromettant ainsi leur sécurité.

Les solutions pour se protéger contre les attaques XSS :

1.Échapper les données en sortie :

- Utilisez la fonction **htmlspecialchars()** en PHP pour échapper les caractères spéciaux lorsque vous affichez des données provenant d'utilisateurs.

  Exemple code :

  ```
  echo htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8');
  ```

2.Utilisation de Content Security Policy (CSP) :

- Implémentez une politique de sécurité de contenu (CSP) pour réduire le risque d'exécution de scripts malveillants.

Exemple code :

```
Content-Security-Policy: default-src 'self'; script-src 'self';
```

3.Utilisation de frameworks et bibliothèques sécurisées :

- Utilisez des frameworks modernes qui incluent des protections contre les attaques XSS par défaut. Par exemple, des frameworks comme **React**, **Vue.js**, intègrent des mécanismes de sécurité pour éviter l'injection de scripts.

Exemple code :

1. **React :**

```
import React from 'react';

function Comment ({comment}) {
```

```
    return (
      <div>
        <p>{comment}</p> {/* Le contenu sera automatiquement échappé */}
      </div>
    );
}

// Si un utilisateur soumet ce commentaire : <script>alert('XSS!');</script>
// Il sera affiché comme du texte, pas comme un script..
```

2.Vue.js

```
<template>
  <div>
    <p>{{ userComment }}</p> <!-- Le contenu sera échappé -->
  </div>
</template>

<script>
export default {
  data() {
    return {
      userComment: '<script>alert("XSS!");</script>' // Input potentiellement dangereux
    };
  }
};
</script>
```

7.4. Protection contre les injections SQL

Les **injections SQL** sont des attaques de sécurité où un pirate peut insérer ou exécuter des commandes SQL malveillantes dans une requête de l'application. Cela se produit généralement lorsque les données fournies par l'utilisateur ne sont pas correctement vérifiées ou sécurisées.
Ces attaques peuvent avoir de graves conséquences, comme le vol de données sensibles (comme des mots de passe ou des informations personnelles), la modification de données existantes ou même la suppression complète de données dans la base de données. En résumé, une injection SQL permet à un attaquant d'accéder à des informations qu'il ne devrait pas pouvoir voir ou manipuler.

Les solutions pour se protéger contre les attaques d'injection SQL :

1. **Utilisation de PDO pour les requêtes SQL :**

Il est conseillé d'utiliser PDO (PHP Data Object) pour exécuter des requêtes SQL. Ce mécanisme permet de préparer des instructions avec des paramètres liés, ce qui aide à prévenir les attaques par injection SQL.

2. **Validation et filtrage des entrées utilisateur :**

Il est crucial de valider et de filtrer toutes les données saisies par les utilisateurs avant de les traiter. Cela garantit que les informations respectent les formats attendus et réduit les risques de vulnérabilités dans l'application.

Exemple code :

```
<?php
// Définir des constantes pour les informations de connexion
define('DB_HOST', 'localhost'); // Adresse du serveur de base de données
define('DB_NAME', 'nom_de_la_base_de_donnees'); // Nom de la base de données
define('DB_USER', 'utilisateur'); // Nom d'utilisateur pour la connexion
define('DB_PASS', 'mot_de_passe'); // Mot de passe de l'utilisateur

try {
    // Création d'une nouvelle instance PDO pour se connecter à la base de données
    $pdo = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME . ";charset=utf8", DB_USER, DB_PASS);

    // Configuration des attributs de PDO pour gérer les erreurs
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); // Lancer des exceptions pour les erreurs
} catch (PDOException $exception) {
    // Gestion des erreurs de connexion
    die("Erreur de connexion : " . $exception->getMessage()); // Terminer le script avec un message d'erreur
}

?>
```

Définition du code d'information de connexion :

- **$host** : L'adresse du serveur de base de données, qui peut être localhost ou une adresse IP distante.
- **$db** : Le nom de la base de données que vous souhaitez utiliser.
- **$user** : Le nom d'utilisateur ayant accès à la base de données.
- **$password** : Le mot de passe associé à cet utilisateur.