Tutoriel: Ajouter un système de sauvegarde des scores avec PHP pour le jeu Pong

Ce tutoriel vous guidera pas à pas pour connecter votre jeu Pong JavaScript à un backend PHP qui permettra de:

- Sauvegarder les scores de chaque partie
- Afficher un tableau des meilleurs scores (highscores)
- Suivre les performances des joueurs

Table des matières

1. Prérequis

Pour suivre ce tutoriel, vous aurez besoin de:

- Un serveur web local avec PHP installé (XAMPP, WAMP, MAMP, etc.)
- MySQL ou MariaDB pour la base de données
- Le jeu Pong de base développé dans le tutoriel précédent

2. Structure du projet

Voici la structure de fichiers que nous allons créer:

```
pong-game/
├── index.html          # Page principale du jeu
├── style.css           # Styles CSS
├── script.js           # JavaScript pour le jeu
├── config/
│   └── db_config.php   # Configuration de la base de données
├── api/
│   ├── save_score.php  # API pour sauvegarder un score
│   └── get_scores.php  # API pour récupérer les highscores
└── db/
    └── setup.sql       # Script SQL pour créer la table
```

3. Création de la base de données

Commençons par créer une base de données et une table pour stocker les scores.

Création du script SQL (db/setup.sql)

```sql
-- Création de la base de données
CREATE DATABASE IF NOT EXISTS pong_game;
USE pong_game;

-- Création de la table des scores
CREATE TABLE IF NOT EXISTS scores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    player_name VARCHAR(50) NOT NULL,
    score INT NOT NULL,
    game_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Insertion de quelques données de test
INSERT INTO scores (player_name, score) VALUES
('Joueur1', 5),
('Joueur2', 3),
('Champion', 10);
```

Exécution du script

Vous pouvez exécuter ce script via phpMyAdmin ou l'interface de ligne de commande de MySQL:

```
mysql -u votre_utilisateur -p < db/setup.sql
```

4. Configuration PHP

Créons un fichier de configuration pour la connexion à la base de données.

Fichier de configuration (config/db_config.php)

```php
<?php
// Désactiver l'affichage des erreurs en production
// ini_set('display_errors', 0);
// error_reporting(E_ALL);

// Configuration de la base de données
define('DB_HOST', 'localhost');
define('DB_USER', 'root');     // Remplacez par votre nom d'utilisateur
define('DB_PASS', '');         // Remplacez par votre mot de passe
define('DB_NAME', 'pong_game');
```

```
// Fonction pour créer une connexion PDO
function getDBConnection() {
    $dsn = "mysql:host=" . DB_HOST . ";dbname=" . DB_NAME . ";charset=utf8mb4";

    try {
        $pdo = new PDO($dsn, DB_USER, DB_PASS);
        // Configurer PDO pour lancer des exceptions en cas d'erreur
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $pdo;
    } catch (PDOException $e) {
        // En production, enregistrez l'erreur dans un fichier de log au lieu de
l'afficher
        die("Erreur de connexion: " . $e->getMessage());
    }
}
?>
```

5. API pour sauvegarder les scores

Créons l'API qui permettra de sauvegarder un score après une partie.

Script de sauvegarde (api/save_score.php)

```
<?php
// Inclure la configuration de la base de données
require_once '../config/db_config.php';

// Définir le type de contenu de la réponse comme JSON
header('Content-Type: application/json');

// Activer CORS pour permettre les requêtes cross-origin (utile en développement)
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: POST, GET, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type');

// Gérer les requêtes OPTIONS (pre-flight)
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    exit(0);
}

// Vérifier que la requête est en POST
if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
    http_response_code(405); // Méthode non autorisée
    echo json_encode(['error' => 'Méthode non autorisée']);
    exit;
}

// Récupérer les données JSON envoyées par le frontend
// Note: file_get_contents('php://input') lit les données brutes du corps de la
requête
$raw_data = file_get_contents('php://input');
```

```php
// Log des données reçues (à utiliser uniquement en développement)
// error_log('Données reçues: ' . $raw_data);

// Conversion du JSON en tableau associatif PHP
$data = json_decode($raw_data, true);

// Vérifier si le décodage JSON a fonctionné
if (json_last_error() !== JSON_ERROR_NONE) {
    http_response_code(400);
    echo json_encode([
        'error' => 'JSON invalide',
        'details' => json_last_error_msg()
    ]);
    exit;
}

// Vérifier que toutes les données nécessaires sont présentes
if (!isset($data['player_name']) || !isset($data['score'])) {
    http_response_code(400); // Requête incorrecte
    echo json_encode(['error' => 'Données manquantes']);
    exit;
}

// Valider les données
$player_name = trim($data['player_name']);
$score = (int)$data['score'];

// Vérifier que le nom du joueur n'est pas vide et que le score est positif
if (empty($player_name) || $score < 0) {
    http_response_code(400); // Requête incorrecte
    echo json_encode(['error' => 'Données invalides']);
    exit;
}

try {
    // Obtenir la connexion à la base de données
    $pdo = getDBConnection();

    // Préparer la requête SQL
    $stmt = $pdo->prepare("INSERT INTO scores (player_name, score) VALUES (?,
?)");

    // Exécuter la requête
    $stmt->execute([$player_name, $score]);

    // Récupérer l'ID de l'enregistrement inséré
    $id = $pdo->lastInsertId();

    // Renvoyer une réponse positive
    echo json_encode([
        'success' => true,
        'message' => 'Score enregistré avec succès',
        'id' => $id
```

```
    ]);

} catch (PDOException $e) {
    http_response_code(500); // Erreur serveur
    echo json_encode(['error' => 'Erreur lors de l\'enregistrement du score']);

    // En production, enregistrez l'erreur dans un fichier de log
    // error_log($e->getMessage());
}
?>
```

6. API pour récupérer les highscores

Maintenant, créons l'API qui permettra de récupérer les meilleurs scores.

Script de récupération (api/get_scores.php)

```
<?php
// Inclure la configuration de la base de données
require_once '../config/db_config.php';

// Définir le type de contenu de la réponse comme JSON
header('Content-Type: application/json');

// Activer CORS pour permettre les requêtes cross-origin (utile en développement)
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type');

// Gérer les requêtes OPTIONS (pre-flight)
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    exit(0);
}

// Vérifier que la requête est en GET
if ($_SERVER['REQUEST_METHOD'] !== 'GET') {
    http_response_code(405); // Méthode non autorisée
    echo json_encode(['error' => 'Méthode non autorisée']);
    exit;
}

// Paramètre optionnel: nombre de scores à récupérer (par défaut: 10)
$limit = isset($_GET['limit']) ? (int)$_GET['limit'] : 10;

// Limiter à un maximum de 100 scores pour éviter les abus
$limit = min(100, max(1, $limit));

try {
    // Obtenir la connexion à la base de données
    $pdo = getDBConnection();
```

```
    // Préparer la requête SQL pour récupérer les meilleurs scores
    $stmt = $pdo->prepare("
        SELECT player_name, score, DATE_FORMAT(game_date, '%d/%m/%Y %H:%i') AS
formatted_date
        FROM scores
        ORDER BY score DESC
        LIMIT ?
    ");

    // Exécuter la requête
    $stmt->execute([$limit]);

    // Récupérer tous les résultats
    $scores = $stmt->fetchAll(PDO::FETCH_ASSOC);

    // Renvoyer les scores au format JSON
    header('Content-Type: application/json');
    echo json_encode([
        'success' => true,
        'scores' => $scores
    ]);

} catch (PDOException $e) {
    http_response_code(500); // Erreur serveur
    echo json_encode(['error' => 'Erreur lors de la récupération des scores']);

    // En production, enregistrez l'erreur dans un fichier de log
    // error_log($e->getMessage());
}
?>
```

7. Modification du JavaScript du jeu

Modifions maintenant le fichier JavaScript du jeu pour lui permettre d'interagir avec nos API PHP.

Communication entre le frontend et le backend

Avant d'ajouter le code, comprenons comment fonctionne la communication entre le frontend (JavaScript) et le backend (PHP) :

Le cycle de communication frontend/backend

1. **Requête** : Le frontend (JavaScript) envoie une requête HTTP au backend (PHP)
2. **Traitement** : Le backend traite la requête (lecture/écriture en base de données)
3. **Réponse** : Le backend renvoie une réponse au frontend
4. **Affichage** : Le frontend met à jour l'interface utilisateur

Le format JSON

JSON (JavaScript Object Notation) est le format d'échange de données privilégié :

```
{
  "player_name": "Champion",
  "score": 10,
  "success": true
}
```

Avantages du JSON :

- Lisible par les humains
- Facilement analysable par les machines
- Natif en JavaScript (via `JSON.parse()` et `JSON.stringify()`)
- Supporté par PHP (via `json_encode()` et `json_decode()`)

L'API Fetch

L'API Fetch est moderne et puissante pour effectuer des requêtes HTTP :

```
// Requête GET basique
fetch('api/get_scores.php')
  .then(response => response.json())
  .then(data => console.log(data));

// Requête POST avec données
fetch('api/save_score.php', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ player_name: 'Joueur', score: 5 })
})
  .then(response => response.json())
  .then(data => console.log(data));
```

Avantages de Fetch :

- Promesses natives (plus moderne que XMLHttpRequest)
- Support de async/await pour un code plus propre
- Gestion flexible des en-têtes et du corps des requêtes

Ajout du code pour gérer les scores (script.js)

Ajoutez ces nouvelles fonctions à votre fichier script.js existant:

```
// Variables pour la gestion des scores
let playerName = '';
let gameOver = false;

// Fonction pour demander le nom du joueur
```

```
function askPlayerName() {
    const name = prompt('Entrez votre nom:', 'Joueur');
    return name ? name.trim().substring(0, 50) : 'Anonyme';
}

// Fonction pour sauvegarder le score
async function saveScore(player, score) {
    try {
        // Préparation des données à envoyer au backend
        const scoreData = {
            player_name: player,
            score: score
        };

        console.log('Envoi au backend:', scoreData);

        // Requête POST vers le backend PHP
        const response = await fetch('api/save_score.php', {
            method: 'POST',                          // Méthode HTTP
            headers: {
                'Content-Type': 'application/json'    // Format des données
            },
            body: JSON.stringify(scoreData)           // Conversion en JSON
        });

        // Vérification de la réponse HTTP
        if (!response.ok) {
            throw new Error(`Erreur HTTP: ${response.status}`);
        }

        // Conversion de la réponse JSON en objet JavaScript
        const data = await response.json();
        console.log('Réponse du backend:', data);

        if (data.success) {
            console.log('Score sauvegardé avec succès! ID:', data.id);
            // Actualiser les highscores après la sauvegarde
            loadHighscores();
        } else {
            console.error('Erreur renvoyée par le backend:', data.error);
        }
    } catch (error) {
        console.error('Erreur lors de la communication avec le backend:', error);
        alert('Impossible de sauvegarder le score. Vérifiez votre connexion.');
    }
}

// Fonction pour charger les meilleurs scores
async function loadHighscores() {
    try {
        console.log('Récupération des highscores depuis le backend...');

        // Requête GET vers le backend PHP
        const response = await fetch('api/get_scores.php');
```

```
        // Vérification de la réponse HTTP
        if (!response.ok) {
            throw new Error(`Erreur HTTP: ${response.status}`);
        }

        // Conversion de la réponse JSON en objet JavaScript
        const data = await response.json();
        console.log('Données reçues du backend:', data);

        if (data.success) {
            console.log(`${data.scores.length} scores récupérés avec succès`);
            // Mettre à jour l'affichage des highscores
            displayHighscores(data.scores);
        } else {
            console.error('Erreur renvoyée par le backend:', data.error);
        }
    } catch (error) {
        console.error('Erreur lors de la communication avec le backend:', error);

        // Afficher un message d'erreur dans le tableau des scores
        const highscoresList = document.getElementById('highscores-list');
        highscoresList.innerHTML = '<div class="error-message">Impossible de
charger les scores. Veuillez réessayer plus tard.</div>';
    }
}

// Fonction pour afficher les highscores dans le DOM
function displayHighscores(scores) {
    const highscoresList = document.getElementById('highscores-list');

    // Vider la liste existante
    highscoresList.innerHTML = '';

    // Créer un élément pour chaque score
    scores.forEach((item, index) => {
        const scoreItem = document.createElement('div');
        scoreItem.className = 'score-item';

        // Ajouter une classe spéciale pour les 3 premiers
        if (index < 3) {
            scoreItem.classList.add('top-three');
        }

        scoreItem.innerHTML = `
            <span class="rank">${index + 1}</span>
            <span class="player-name">${item.player_name}</span>
            <span class="score">${item.score}</span>
            <span class="date">${item.formatted_date}</span>
        `;

        highscoresList.appendChild(scoreItem);
    });
}
```

```
// Modification de la fonction resetGame pour réinitialiser gameOver
function resetGame() {
    gameRunning = false;
    gameOver = false;
    resetBall();
    paddleLeftY = 160;
    paddleRightY = 160;
    leftScore = 0;
    rightScore = 0;
    scoreLeft.textContent = leftScore;
    scoreRight.textContent = rightScore;
    updatePositions();
}

// Modification de la fonction updateBall pour détecter la fin de partie
function updateBall() {
    ballX += ballSpeedX;
    ballY += ballSpeedY;

    // Collision avec les bords haut et bas
    if (ballY <= 0 || ballY >= gameBoard.clientHeight - ball.clientHeight) {
        ballSpeedY = -ballSpeedY;
    }

    // Détection pour la raquette gauche
    if (
        ballX <= paddleLeft.clientWidth + 10 &&
        ballY + ball.clientHeight >= paddleLeftY &&
        ballY <= paddleLeftY + paddleLeft.clientHeight
    ) {
        ballSpeedX = -ballSpeedX;
    }

    // Détection pour la raquette droite
    if (
        ballX >= gameBoard.clientWidth - paddleRight.clientWidth -
ball.clientWidth - 10 &&
        ballY + ball.clientHeight >= paddleRightY &&
        ballY <= paddleRightY + paddleRight.clientHeight
    ) {
        ballSpeedX = -ballSpeedX;
    }

    // Balle qui sort du terrain
    if (ballX < 0) {
        // Point pour le joueur droit
        rightScore++;
        scoreRight.textContent = rightScore;

        // Vérifier si la partie est terminée (score de 10)
        if (rightScore >= 10 && !gameOver) {
            endGame('right');
        } else {
```

```
            resetBall();
        }
    } else if (ballX > gameBoard.clientWidth - ball.clientWidth) {
        // Point pour le joueur gauche
        leftScore++;
        scoreLeft.textContent = leftScore;

        // Vérifier si la partie est terminée (score de 10)
        if (leftScore >= 10 && !gameOver) {
            endGame('left');
        } else {
            resetBall();
        }
    }
}

// Fonction pour terminer la partie
function endGame(winner) {
    gameRunning = false;
    gameOver = true;

    // Déterminer le joueur gagnant et son score
    const winnerName = winner === 'left' ? 'Joueur 1' : 'Joueur 2';
    const finalScore = winner === 'left' ? leftScore : rightScore;

    // Afficher un message
    alert(`${winnerName} a gagné avec un score de ${finalScore}!`);

    // Demander le nom du joueur
    playerName = askPlayerName();

    // Sauvegarder le score
    saveScore(playerName, finalScore);
}

// Charger les highscores au démarrage
document.addEventListener('DOMContentLoaded', () => {
    loadHighscores();
});
```

8. Affichage du tableau des scores

Modifions le HTML pour ajouter une section qui affichera les meilleurs scores.

Modification du HTML (index.html)

Ajoutez cette section après la section des contrôles :

```
<div class="highscores">
    <h2>Meilleurs Scores</h2>
    <div class="highscores-header">
```

```
            <span class="rank">Rang</span>
            <span class="player-name">Joueur</span>
            <span class="score">Score</span>
            <span class="date">Date</span>
        </div>
        <div id="highscores-list">
            <!-- Les scores seront ajoutés ici dynamiquement -->
        </div>
    </div>
```

Ajout des styles CSS (style.css)

Ajoutez ces styles à votre fichier CSS:

```
/* Styles pour le tableau des scores */
.highscores {
    width: 100%;
    max-width: 600px;
    margin-top: 30px;
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    padding: 15px;
}

.highscores h2 {
    text-align: center;
    color: #333;
    margin-top: 0;
}

.highscores-header {
    display: grid;
    grid-template-columns: 50px 2fr 1fr 2fr;
    padding: 10px 0;
    border-bottom: 2px solid #ddd;
    font-weight: bold;
}

.score-item {
    display: grid;
    grid-template-columns: 50px 2fr 1fr 2fr;
    padding: 8px 0;
    border-bottom: 1px solid #eee;
}

.score-item:hover {
    background-color: #f9f9f9;
}

.top-three {
```

```
    background-color: #f8f8d7;
}

.rank {
    font-weight: bold;
    text-align: center;
}

.player-name {
    font-weight: 500;
}

.score {
    text-align: center;
    font-weight: bold;
}

.date {
    text-align: right;
    color: #777;
    font-size: 0.9em;
}
```

9. Approfondissement des échanges Front-End/Back-End

Cycle de vie complet d'une requête

Pour mieux comprendre les échanges entre le front-end et le back-end, voici le cycle de vie complet d'une requête:

```
Frontend (JS)               |               Backend (PHP)
--------------------------- | -------------------------------
1. Événement utilisateur    |
   (fin de partie)          |
                            |
2. Préparation des données  |
   (player_name, score)     |
                            |
3. Conversion en JSON       |
   JSON.stringify()         |
                            |
4. Envoi requête HTTP       | 5. Réception de la requête
   fetch() ----------------->  $_POST ou php://input
                            |
                            | 6. Décodage JSON
                            |    json_decode()
                            |
                            | 7. Validation des données
                            |
                            | 8. Opération en base de données
                            |    (INSERT INTO scores...)
```

```
                              |
                              | 9. Préparation de la réponse
                              |    (succès ou erreur)
                              |
                              | 10. Encodage JSON
                              |     json_encode()
                              |
11. Réception réponse         | 12. Envoi de la réponse HTTP
    .then() ou await      <---
                              |
12. Décodage JSON             |
    response.json()           |
                              |
13. Traitement côté client    |
    (mise à jour UI)          |
```

Gestion avancée des requêtes avec Fetch

Gestion des timeout

```
// Définir un timeout pour les requêtes fetch
const fetchWithTimeout = (url, options, timeout = 5000) => {
  return Promise.race([
    fetch(url, options),
    new Promise((_, reject) =>
      setTimeout(() => reject(new Error('Timeout')), timeout)
    )
  ]);
};

// Utilisation
try {
  const response = await fetchWithTimeout('api/get_scores.php', {}, 3000);
  // Traitement de la réponse
} catch (error) {
  if (error.message === 'Timeout') {
    console.error('La requête a pris trop de temps');
  } else {
    console.error('Autre erreur:', error);
  }
}
```

Gestion des états de chargement

```
// État de chargement
const [loading, setLoading] = useState(false);

async function loadHighscores() {
```

```
    setLoading(true);
    try {
      const response = await fetch('api/get_scores.php');
      const data = await response.json();
      displayHighscores(data.scores);
    } catch (error) {
      console.error('Erreur:', error);
    } finally {
      setLoading(false);
    }
  }

  // Dans l'affichage
  if (loading) {
    return <div>Chargement en cours...</div>;
  }
```

Stratégies de mise en cache

Pour optimiser les performances, vous pouvez mettre en cache les highscores côté client:

```
// Mise en cache des highscores avec timestamp
function cacheHighscores(scores) {
  const cacheData = {
    timestamp: Date.now(),
    scores: scores
  };
  localStorage.setItem('highscores_cache', JSON.stringify(cacheData));
}

// Récupération des scores avec vérification du cache
async function getHighscores() {
  // Vérifier si des données en cache existent
  const cachedData = localStorage.getItem('highscores_cache');

  if (cachedData) {
    const cache = JSON.parse(cachedData);
    const cacheAge = Date.now() - cache.timestamp;

    // Si le cache a moins de 5 minutes, l'utiliser
    if (cacheAge < 300000) {
      console.log('Utilisation des données en cache');
      return cache.scores;
    }
  }

  // Sinon, faire une requête au serveur
  console.log('Récupération depuis le serveur');
  const response = await fetch('api/get_scores.php');
  const data = await response.json();
```

```
    if (data.success) {
      // Mettre en cache les nouvelles données
      cacheHighscores(data.scores);
      return data.scores;
    }

    throw new Error('Impossible de récupérer les scores');
  }
```

10. Améliorations possibles

Voici quelques idées pour améliorer ce système de scores:

1. **Authentification des joueurs**
    - Ajouter un système de login pour suivre les performances d'un joueur particulier
    - Utiliser des tokens JWT pour sécuriser les requêtes API
2. **Statistiques avancées**
    - Suivre le nombre de parties jouées, la moyenne des scores, etc.
    - Créer des endpoints API dédiés aux différentes statistiques
3. **Filtres pour les highscores**
    - Permettre de filtrer par date, par joueur, etc.
    - Utiliser des paramètres de requête dans les appels API (`?player=Jean&limit=10`)
4. **Mode multijoueur en ligne**
    - Utiliser WebSockets pour une communication en temps réel
    - Implémenter un système de matchmaking
5. **Système de classement**
    - Implémenter un système ELO pour classer les joueurs
    - Créer des saisons de classement avec réinitialisation périodique

Code complet

Consultez les sections précédentes pour le code complet de chaque fichier. L'intégration de ce système de sauvegarde des scores et d'affichage des highscores va transformer votre jeu Pong en une application web complète avec persistence des données.

Notes importantes

1. **Sécurité** : Dans un environnement de production, assurez-vous de:
    - Valider et filtrer toutes les entrées utilisateur
    - Utiliser des requêtes préparées (comme nous l'avons fait)
    - Protéger contre les injections SQL et XSS

2. **Performance** : Pour un grand nombre d'utilisateurs, envisagez :
    - La mise en cache des highscores
    - L'optimisation des requêtes SQL avec des index
3. **Déploiement** : Pour déployer cette application :
    - Assurez-vous que votre hébergeur supporte PHP et MySQL
    - Adaptez les paramètres de connexion à la base de données
    - Testez abondamment avant la mise en production