# **TD Web :** Développement d’un Quiz Dynamique en PHP

*Ce projet a pour objectif :*

- Approfondir les connaissances en **HTML**, **CSS**, et **JavaScript** pour la création d'une interface utilisateur interactive.
- Apprendre à utiliser **PHP** pour la gestion du backend (traitement des données, gestion des scores, manipulation de fichiers ou base de données).
- Comprendre l’intégration entre le **frontend** (HTML/CSS/JS) et le **backend** (PHP).
- Travailler en collaboration sur un projet complet en utilisant les bonnes pratiques de développement web.

*Pré-requis :*

- **HTML** de base et balise d’organisation
- **CSS** Alignement des blocs ou modèle de boite
- Manipulation du DOM avec **JavaScript**
- **PHP** Procédural

## Partie 1 : Introduction au Projet

Le projet consiste à développer un **Jeu de Quiz Dynamique**. Le joueur doit répondre à une série de questions à choix multiples dans un temps limité. Le score et le temps sont gérés par le serveur PHP. Il y aura également une interface d'administration pour ajouter ou modifier les questions.

Avant de commencer la partie technique du projet, il est important de comprendre les rôles et les utilisations de chaque langage dans le développement d'un site ou d'une application web. Voici une introduction aux principaux langages que vous utiliserez dans ce projet.

**1. HTML (HyperText Markup Language)**

- **Rôle** : HTML est le langage de balisage utilisé pour structurer le contenu d'une page web.
- **Dans ce projet** : HTML sera utilisé pour construire la structure de base du jeu de quiz, comme les zones où les questions et les options de réponse seront affichées, ainsi que les boutons pour naviguer entre les questions.

**2. CSS (Cascading Style Sheets)**

- **Rôle** : CSS permet de définir l'apparence et le style des pages web, en modifiant l'aspect visuel des éléments créés avec HTML.
- **Dans ce projet** : CSS sera utilisé pour styliser le jeu de quiz, en rendant l'interface attrayante et ergonomique pour les utilisateurs.

**3. JavaScript (JS)**

- **Rôle** : JavaScript est un langage de programmation côté client (exécuté dans le navigateur) qui permet de rendre les pages web interactives.

- **Dans ce projet** : JavaScript sera principalement utilisé pour :
    - **Gérer les interactions utilisateur** : répondre aux choix de réponses, passer à la question suivante.
    - **Gérer le chronomètre** : limiter le temps de réponse pour chaque question.
    - **Afficher les résultats** en temps réel en fonction des réponses.

**Concepts clés en JavaScript à maîtriser :**

- **DOM (Document Object Model)** : représente la structure HTML sous forme d'objet que JS peut manipuler.
- **Gestion des événements** : écoute des actions utilisateur comme les clics.
- **setInterval** et **clearInterval** : pour la gestion du temps (utilisé pour le timer).
- **Manipulation des classes et des styles** : modification dynamique des éléments HTML/CSS.

### 4. PHP (Hypertext Preprocessor)

- **Rôle** : PHP est un langage de programmation côté serveur, ce qui signifie qu'il est exécuté sur le serveur web et non sur le navigateur du client. PHP permet de gérer les interactions avec une base de données, traiter les formulaires soumis par les utilisateurs, et générer dynamiquement du contenu HTML.
- **Dans ce projet** : PHP sera utilisé pour :
    - **Générer dynamiquement les questions du quiz** depuis un fichier JSON ou une base de données.
    - **Sauvegarder les scores et statistiques** des joueurs.
    - **Fournir une interface d'administration** permettant d'ajouter ou modifier les questions du quiz.

**Concepts clés en PHP à maîtriser :**

- **Manipulation de fichiers** : lecture et écriture dans des fichiers (JSON).
- **Gestion des formulaires** : récupération et traitement des données envoyées par les utilisateurs.
- **Sessions** : pour gérer des informations comme les scores du joueur.
- **Sécurisation des données** : éviter les failles comme l'injection SQL ou la manipulation non sécurisée des fichiers.

## Partie 2 : Plan du projet

### Fonctionnalités du jeu :

1. **Affichage dynamique des questions** :
   - Le joueur voit une question à la fois avec 4 options de réponse.
   - Il a un temps limité pour répondre à chaque question.
2. **Gestion des scores et statistiques** :
   - Le score est calculé en fonction des bonnes réponses.
   - Le temps moyen de réponse est enregistré.
3. **Interface administrateur** :
   - Permet à l'administrateur d'ajouter, modifier ou supprimer des questions.
4. **Sauvegarde des résultats** :
   - Les scores et statistiques sont sauvegardés sur le serveur (via une base de données ou un fichier JSON).

### Arborescence :

Voici une arborescence typique pour le projet, avec des fichiers et dossiers bien organisés pour une meilleure gestion du code. Cette arborescence est organisée de manière à séparer les responsabilités entre les différentes technologies (HTML, CSS, JS, PHP) et les ressources nécessaires au projet.

```
/quiz_project                # Dossier racine du projet
│
├── /assets                  # Dossier pour les ressources statiques (images, fonts, etc.)
│   └── /images              # Dossier pour les images utilisées dans le projet
│
├── /css                     # Dossier pour les fichiers CSS
│   └── style.css            # Fichier CSS principal pour styliser l'application
│
├── /js                      # Dossier pour les fichiers JavaScript
│   └── script.js            # Fichier JS principal pour la logique du quiz (frontend)
│
├── /php                     # Dossier pour les fichiers PHP (backend)
│   ├── get_question.php     # Fichier PHP pour récupérer les questions (JSON ou DB)
│   ├── save_score.php       # Fichier PHP pour enregistrer les scores dans la base de données
│   └── add_question.php     # Fichier PHP pour ajouter une question dans le fichier JSON ou la base de données
│
├── /data                    # Dossier pour les fichiers de données (si JSON est utilisé)
│   └── questions.json       # Fichier JSON contenant les questions du quiz
│
├── /db                      # Dossier pour les fichiers liés à la base de données (scripts SQL)
│   └── init_db.sql          # Script SQL pour créer les tables de la base de données (questions, scores)
│
├── /admin                   # Dossier pour l'interface d'administration
│   └── index.php            # Interface admin pour ajouter/modifier les questions
│
├── index.php                # Page d'accueil du quiz (génère l'interface du jeu)
├── results.php              # Page pour afficher les résultats du quiz (scores)
├── README.md                # Fichier README expliquant le projet, comment le configurer et l'utiliser
└── .htaccess                # Fichier de configuration Apache (si nécessaire)
```

## Détails de l'arborescence :

1. **/assets** : Ce dossier contient tous les fichiers statiques comme les images, icônes, ou fonts.
2. **/css/style.css** : Le fichier principal pour les styles du projet. Il contient toutes les règles CSS pour mettre en forme l'interface utilisateur (quiz et admin).
3. **/js/script.js** : Ce fichier JavaScript gère les interactions du frontend (affichage des questions, gestion du timer, validation des réponses). Il peut également contenir la logique pour récupérer les données du backend via AJAX ou `fetch`.
4. **/php/** :
    - **get_question.php** : Ce fichier récupère une question aléatoire depuis le fichier JSON ou la base de données et la renvoie en format JSON pour que le frontend puisse l'afficher.
    - **save_score.php** : Il sauvegarde le score d'un joueur, soit dans un fichier texte ou dans une base de données.
    - **add_question.php** : Permet d'ajouter une nouvelle question dans le fichier JSON ou la base de données via l'interface d'administration.
5. **/data/questions.json** : Fichier contenant toutes les questions et réponses du quiz au format JSON. Ce fichier sera utilisé jusqu'à ce que l'on passe à une base de données.
6. **/db/init_db.sql** : Ce fichier contient le script SQL pour créer les tables nécessaires à la base de données (par exemple, tables `questions` et `scores`).
7. **/admin/index.php** : Interface pour l'administrateur permettant de gérer (ajouter, modifier, supprimer) les questions du quiz.
8. **index.php** : Fichier principal qui contient l'interface du quiz. C'est la page d'accueil où le joueur commence le quiz.
9. **results.php** : Cette page affichera les résultats (score final) du joueur à la fin du quiz.
10. **README.md** : Ce fichier contient des instructions pour installer, configurer, et utiliser le projet.
11. **.htaccess** : Ce fichier est optionnel. Il peut être utilisé pour des règles spécifiques dans Apache (par exemple, pour la redirection ou la gestion des URL).

## Développement progressif :

1. **Phase 1 (JSON)** : Dans un premier temps, vous commencerez à travailler avec le fichier **questions.json** pour stocker et récupérer les questions. Tous les scripts PHP fonctionnent avec ce fichier pour la gestion du quiz.

2. **Phase 2 (Base de données)** : Une fois le quiz fonctionnel avec le fichier JSON, vous pourrez migrer vers une base de données MySQL. Vous ajouterez les tables nécessaires avec le fichier **init_db.sql** et modifiez les scripts PHP pour récupérer les questions et sauvegarder les scores dans la base de données.

## Partie 3 : Développement de l'interface utilisateur (HTML/CSS/JS)

**TD Web : Quiz Dynamique**

## Tâche 1 : Mise en place du HTML de base

- **Exercice** : Créez la structure HTML pour le jeu. Incluez une zone pour afficher les questions, les réponses, un bouton « Suivant », et un espace pour afficher le timer.

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Quiz Dynamique</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="quiz-container">
        <h2 id="question">Question ici</h2> <!-- Affiche la question actuelle -->
        <div id="choices">
            <button class="choice">Réponse 1</button>
            <button class="choice">Réponse 2</button>
            <button class="choice">Réponse 3</button>
            <button class="choice">Réponse 4</button>
        </div>
        <button id="next">Suivant</button> <!-- Bouton pour passer à la question suivante -->
        <p id="timer">Temps restant : 30s</p> <!-- Affiche le compte à rebours -->
    </div>
    <script src="script.js"></script> <!-- Lié au fichier JavaScript pour gérer la logique du quiz -->
</body>
</html>
```

- **Explication** : Cette structure de base vous permet d'afficher la question, les choix et un bouton pour passer à la question suivante.

`<div>` : conteneur pour structurer les éléments. Ici, `#quiz-container` regroupe la question, les choix de réponse, et le timer.

`<h2>` : balise pour afficher le titre de la question.

`<button>` : utilisé pour afficher les réponses sous forme de boutons cliquables.

`<script>` : inclusion du fichier JavaScript pour ajouter des comportements interactifs au quiz.

## Tâche 2 : Styliser l'interface avec CSS

Le CSS permet d'améliorer l'expérience utilisateur en rendant l'interface plus agréable à utiliser. On y définit les couleurs, la disposition des éléments, les espacements, et les interactions visuelles (survol des boutons, transitions...).

**Exercice** : Créez un fichier `style.css` pour rendre l'interface attrayante. Utilisez des propriétés comme `flexbox` pour centrer les éléments, et des transitions CSS pour animer les boutons lorsqu'ils sont survolés.

*Exemple de style de base :*

```css
/* Style global pour la page */
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4; /* Couleur de fond légère */
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh; /* Remplit tout l'écran verticalement */
}

/* Conteneur principal du quiz */
#quiz-container {
    background: white;
    padding: 20px;
    border-radius: 10px; /* Angles arrondis */
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Ombre légère */
    width: 300px;
    text-align: center;
}

/* Style des boutons de réponse */
button.choice {
    display: block;
    width: 100%;
    margin: 10px 0; /* Espacement entre chaque bouton */
    padding: 10px;
    border: none;
    background-color: #007BFF; /* Couleur bleue */
    color: white;
    border-radius: 5px;
    cursor: pointer; /* Curseur en main */
    transition: background-color 0.3s; /* Transition douce */
}

/* Changement de couleur au survol */
button.choice:hover {
    background-color: #0056b3; /* Couleur plus foncée */
}
```

**Flexbox** : permet d'aligner les éléments au centre de l'écran (verticalement et horizontalement).

**Box-shadow** : applique une ombre subtile au conteneur pour le rendre plus esthétique.

**Transition** : ajoute une transition douce lorsque l'utilisateur survole les boutons, améliorant l'expérience utilisateur.

## Tâche 3 : Timer en JavaScript

Dans un quiz, chaque question doit être limitée par un temps défini. Le **timer** joue un rôle crucial dans la gestion du stress et la difficulté du jeu.

**Exercice** : Implémentez un **compte à rebours** en JavaScript qui commence à 30 secondes pour chaque question. Si le joueur ne répond pas avant la fin du temps, la question est automatiquement comptée comme incorrecte.

*Exemple de code JavaScript pour le timer :*

```javascript
let timer;
let timeLeft = 30; // Durée initiale en secondes

function startTimer() {
    timer = setInterval(function () {
        if (timeLeft <= 0) {
            clearInterval(timer); // Arrête le timer quand il atteint 0
            alert("Temps écoulé !");
            // Logic pour passer à la question suivante automatiquement
        } else {
            document.getElementById('timer').innerHTML = "Temps restant : " + timeLeft + "s";
            timeLeft--; // Diminue le temps restant d'une seconde
        }
    }, 1000); // Le timer diminue toutes les secondes (1000ms)
}

startTimer();
```

**setInterval** : fonction JavaScript qui exécute une action (ici, diminuer le timer) à intervalles réguliers (toutes les 1000 ms, soit 1 seconde).

**clearInterval** : utilisé pour arrêter le timer une fois le temps écoulé.

**Manipulation du DOM** : mise à jour du texte du `timer` en fonction du temps restant.

## Partie 4 : Gestion de la logique côté serveur avec PHP

Dans cette partie, nous allons nous concentrer sur l'utilisation de **PHP** pour gérer la logique de notre jeu de quiz dynamique. Nous allons d'abord utiliser un fichier **JSON** pour stocker et récupérer les questions, puis une fois que tout fonctionnera avec succès, nous passerons à la gestion des données via une **base de données** MySQL. Cette approche permettra aux étudiants d'apprendre progressivement la gestion des données avec PHP, d'abord avec un fichier, puis avec une base de données.

**Pourquoi utiliser PHP ?**

PHP est un langage de programmation exécuté sur le serveur, ce qui signifie qu'il permet de traiter des informations en arrière-plan avant que les résultats ne soient envoyés à l'utilisateur. Dans le cadre de ce projet, PHP jouera les rôles suivants :

1. **Générer dynamiquement les questions** : PHP va lire les questions stockées dans un fichier JSON ou une base de données, puis renvoyer ces données au navigateur de l'utilisateur pour qu'il puisse répondre.
2. **Sauvegarder les scores** : PHP permettra d'enregistrer les scores des joueurs dans un fichier ou une base de données.
3. **Interface d'administration** : PHP sera utilisé pour permettre à l'administrateur d'ajouter, modifier ou supprimer des questions depuis une interface.

## Étape 1 : Gestion des questions avec un fichier JSON

**TD Web : Quiz Dynamique**

Dans cette première étape, nous allons stocker les questions et réponses dans un fichier JSON. Ce fichier sera utilisé par PHP pour récupérer les données et les envoyer au frontend sous forme de JSON, que JavaScript pourra ensuite utiliser pour afficher les questions.

## Tâche 1 : Création du fichier JSON

Le fichier **questions.json** contiendra les questions du quiz sous forme d'un tableau d'objets. Chaque objet représente une question avec ses choix de réponse et la réponse correcte.

*Exemple de fichier **questions.json** :*

```json
[
    {
        "question": "Quelle est la capitale de la France ?",
        "choices": ["Paris", "Lyon", "Marseille", "Nice"],
        "correct": 0
    },
    {
        "question": "Quelle est la couleur du ciel ?",
        "choices": ["Rouge", "Bleu", "Vert", "Jaune"],
        "correct": 1
    }
]
```

*Dans cet exemple, chaque objet a trois clés principales :*

- `question` : la question posée.
- `choices` : un tableau contenant quatre options de réponse.
- `correct` : l'indice de la bonne réponse dans le tableau `choices` (ici, 0 signifie que la bonne réponse est le premier élément du tableau).

## Tâche 2 : Création d'un script PHP pour récupérer les questions

L'objectif est de créer un fichier PHP qui va lire les questions depuis le fichier **questions.json** et les renvoyer au navigateur sous forme de données JSON. Voici comment procéder :

```php
<?php
// Lire le contenu du fichier JSON
$jsonData = file_get_contents('questions.json');

// Décoder les données JSON en un tableau PHP
$questions = json_decode($jsonData, true);

// Sélectionner une question aléatoire parmi le tableau de questions
$randomQuestion = $questions[array_rand($questions)];

// Encoder la question sélectionnée en JSON pour l'envoyer au navigateur
echo json_encode($randomQuestion);
?>
```

**file_get_contents('questions.json')** : cette fonction lit le contenu du fichier `questions.json`. Le fichier doit être dans le même répertoire que le script PHP, ou bien

vous devez spécifier le chemin correct.

**json_decode($jsonData, true)** : cette fonction convertit le contenu JSON en un tableau associatif PHP. Le deuxième paramètre (`true`) est important car il indique à PHP de convertir le JSON en un tableau associatif (au lieu d'objets PHP).

**array_rand($questions)** : cette fonction PHP sélectionne une clé aléatoire dans le tableau de questions.

**json_encode($randomQuestion)** : cette fonction PHP encode la question sélectionnée en format JSON, prêt à être envoyé à JavaScript.

### Tâche 3 : Récupération des données avec JavaScript

Une fois que le script PHP est en place, le frontend (JavaScript) doit récupérer ces données et les afficher dynamiquement dans l'interface.

Voici un exemple de code JavaScript pour récupérer une question à partir du script PHP :

```javascript
// Fonction pour récupérer une question via PHP
function fetchQuestion() {
    fetch('get_question.php') // Appelle le fichier PHP qui génère la question
    .then(response => response.json()) // Convertit la réponse en JSON
    .then(data => {
        // Affiche la question et les choix de réponse
        document.getElementById('question').innerText = data.question;

        // Affiche les choix
        const choices = document.querySelectorAll('.choice');
        choices.forEach((choice, index) => {
            choice.innerText = data.choices[index]; // Remplit les boutons avec les choix
        });
    })
    .catch(error => console.error('Erreur : ', error)); // Gère les erreurs
}

// Appelle la fonction au démarrage pour charger la première question
fetchQuestion();
```

**fetch('get_question.php')** : fait une requête vers le fichier PHP `get_question.php` pour récupérer une question.

**response.json()** : transforme la réponse en format JSON.

**data.question** et **data.choices[index]** : accède aux données JSON renvoyées par PHP (la question et les choix) pour les afficher dans l'interface.

## Étape 2 : Sauvegarder les scores dans un fichier texte

**TD Web : Quiz Dynamique**

Une fois que le joueur a terminé le quiz ou une question, nous voulons sauvegarder son score pour référence ultérieure. Nous allons commencer par stocker ces scores dans un simple fichier texte.

*Exemple de formulaire pour soumettre le score :*

```html
<form id="scoreForm" method="POST" action="save_score.php">
    <input type="hidden" id="scoreInput" name="score" value="">
    <button type="submit">Sauvegarder le score</button>
</form>
```

Ici, le formulaire est invisible pour l'utilisateur. Il soumet simplement le score au fichier PHP `save_score.php`.

*Exemple de code PHP pour sauvegarder un score dans un fichier texte :*

```php
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Récupère le score depuis le formulaire
    $score = $_POST['score'];

    // Ouvre le fichier en mode "append" (ajouter à la fin du fichier)
    $file = fopen('scores.txt', 'a');

    // Écrit le score dans le fichier
    fwrite($file, "Score : " . $score . "\n");

    // Ferme le fichier
    fclose($file);

    echo "Score sauvegardé avec succès !";
}
?>
```

**$_POST['score']** : récupère le score envoyé depuis le formulaire via la méthode POST.

**fopen('scores.txt', 'a')** : ouvre le fichier `scores.txt` en mode `append` (ce qui signifie que chaque nouveau score sera ajouté à la fin du fichier, sans effacer les scores existants).

**fwrite** : écrit le score dans le fichier.

**fclose** : ferme le fichier pour libérer les ressources.

## Étape 3 : Passer à l'utilisation d'une base de données MySQL

Après avoir implémenté la gestion des questions et des scores avec des fichiers (JSON pour les questions, fichier texte pour les scores), nous allons maintenant remplacer ces fichiers par une **base de données MySQL** pour un système plus robuste et performant.

**Tâche 1 : Configuration de la base de données MySQL**

1. **Création de la base de données** :
   - Connectez-vous à votre serveur MySQL (via PHPMyAdmin ou un terminal MySQL).
   - Créez une base de données appelée `quiz`.

```
CREATE DATABASE quiz;
```

2. **Création de la table `questions`** :
   - Nous allons créer une table pour stocker nos questions et leurs réponses.

```
CREATE TABLE questions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    question TEXT NOT NULL,
    choice1 VARCHAR(255) NOT NULL,
    choice2 VARCHAR(255) NOT NULL,
    choice3 VARCHAR(255) NOT NULL,
    choice4 VARCHAR(255) NOT NULL,
    correct INT NOT NULL
);
```

3. **Insertion des questions dans la base de données** :

```
INSERT INTO questions (question, choice1, choice2, choice3, choice4, correct)
VALUES ('Quelle est la capitale de la France ?', 'Paris', 'Lyon', 'Marseille', 'Nice', 0);
```

**TD Web : Quiz Dynamique**

## Tâche 2 : Connexion à la base de données avec PHP

Nous devons maintenant adapter notre fichier PHP pour qu'il récupère les questions depuis la base de données plutôt que depuis un fichier JSON.

*Exemple de code PHP pour récupérer une question depuis la base de données :*

```php
<?php
// Informations de connexion à la base de données
$host = 'localhost';
$db = 'quiz';
$user = 'root';
$pass = '';

try {
    // Connexion à la base de données
    $pdo = new PDO("mysql:host=$host;dbname=$db;charset=utf8", $user, $pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Récupérer une question aléatoire
    $stmt = $pdo->query("SELECT * FROM questions ORDER BY RAND() LIMIT 1");
    $question = $stmt->fetch(PDO::FETCH_ASSOC);

    // Renvoie la question sous forme de JSON
    echo json_encode($question);
} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
?>
```

**PDO** : permet de se connecter à une base de données MySQL en toute sécurité.

**query("SELECT * FROM questions ORDER BY RAND() LIMIT 1")** : récupère une question aléatoire depuis la table `questions`.

**fetch(PDO::FETCH_ASSOC)** : récupère la question sous forme de tableau associatif.

**json_encode** : encode la question en JSON pour l'envoyer au frontend.

## Étape 4 : Sauvegarder les scores dans la base de données

Maintenant que nous avons connecté notre quiz à une base de données MySQL, nous allons également stocker les scores dans une table.

1. **Création de la table `scores`** :

```sql
CREATE TABLE scores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    score INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. **Enregistrement des scores via PHP** :

Modifions le fichier `save_score.php` pour qu'il enregistre les scores dans la base de données au lieu d'un fichier texte.

```php
<?php
// Informations de connexion à la base de données
$host = 'localhost';
$db = 'quiz';
$user = 'root';
$pass = '';

try {
    // Connexion à la base de données
    $pdo = new PDO("mysql:host=$host;dbname=$db;charset=utf8", $user, $pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Récupérer le score depuis le formulaire
    $score = $_POST['score'];

    // Insertion du score dans la base de données
    $stmt = $pdo->prepare("INSERT INTO scores (score) VALUES (:score)");
    $stmt->execute(['score' => $score]);

    echo "Score sauvegardé avec succès !";
} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
?>
```

**PDO** : utilisé pour se connecter à la base de données MySQL.

**prepare()** : prépare une requête SQL avec des paramètres sécurisés (ici, le score).

**execute()** : exécute la requête SQL pour insérer le score dans la table `scores`.