PHP –MVC

SOMMAIRE

Edition : mars 2018

MVC

https://openclassrooms.com/courses/votre-site-php-presque-complet-architecture-mvc-et-bonnes-pratiques/avant-propos-comment-fonctionne-ce-tutoriel

https://openclassrooms.com/courses/adopter-un-style-de-programmation-clair-avec-le-modele-mvc

Problématique : quels fichiers, quels dossiers pour mes projets

Quels dossiers dois-je créer ? », « Comment dois-je organiser mes fichiers ? », « Ai-je besoin d'un dossier admin ? », etc.

Organisation non-MVC

Principes d'organisation non-MVC

On va avoir une page php par page de site.

Une page c'est comme une fonction : elle reçoit des paramètres ($_GET, $_POST). Mais la liste est variable selon l'entrée (un href, un autre href, un formulaire de saisie, un autre, un header). Et il y a des variables globales : $_SESSION.

La page réagit différemment en fonction de ce qu'elle reçoit comme information. Selon les cas, elle fera en totalité ou en partie :

- Des include de header, footer, connexion à la BD, etc.
- Des mises à jour de variables dans les $_TAB (et les SELECT associés si nécessaire)
- Des traitements des données : INSERT, UPDATE, DELETE et des SELECT associés si nécessaire.
- La construction de la page HTML à afficher (et les SELECT associés si nécessaire)

Défauts de l'organisation non-MVC

- Tout est un peu mélangé : le SQL, le PHP et le HTML.
- Les pages peuvent devenir très grosses.
- La maintenance n'est pas facile.
- Travail à plusieurs est rendu difficile.

Présentation du MVC

Présentation

MVC : Modèle - Vue - Contrôleur.

L'architecture MVC sépare la logique du code en trois parties, trois ensembles de fichiers.

Cela rend le code plus facile à mettre à jour et permet d'organiser le travail en 3 parties et donc de travailler en parallèle.

L'architecture MVC est une bonne pratique de programmation.

La connaissance de l'architecture MVC rend capable de créer un site web de qualité et facile à maintenir.

En pratique, les architectures MVC mises en œuvre s'appuient sur la théorie mais l'adaptent de façon pragmatique. Il y a donc plusieurs façons de mettre en œuvre le MVC.

Les principaux framework sont développés en MVC : CodeIgniter, CakePHP, Symfony, Jelix, Zend Framework.

Le Modèle (SQL)

Il gère les données du site. Essentiellement les accès à la BD. Mais aussi la gestion de fichiers. Il propose des fonctions pour faire des Insert, des Update, des Delete, des Select. Ces fonctions peuvent renvoyer des tableaux de données. Les résultats seront exploités par le contrôleur.

C'est une page pur php.

L'idée générale est que dans une application, la base de données est centrale. **Si la BD est bien conçue, l'application sera facile à maintenir et à faire évoluer.** Si la BD est mal conçue, l'application sera complexe à maintenir.

La Vue (HTML)

Elle affiche la page HTML. Elles récupèrent des variables du Contrôleur pour savoir ce qu'elles doivent afficher.

On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher les tableaux de données issus du Modèle.

La vue contient le DOCTYPE. Mais elle ne peut fonctionner qu'avec le contexte du contrôleur.

Le Contrôleur (PHP)

C'est la page appelée (le véritable index).

Il fonctionne en trois étapes :

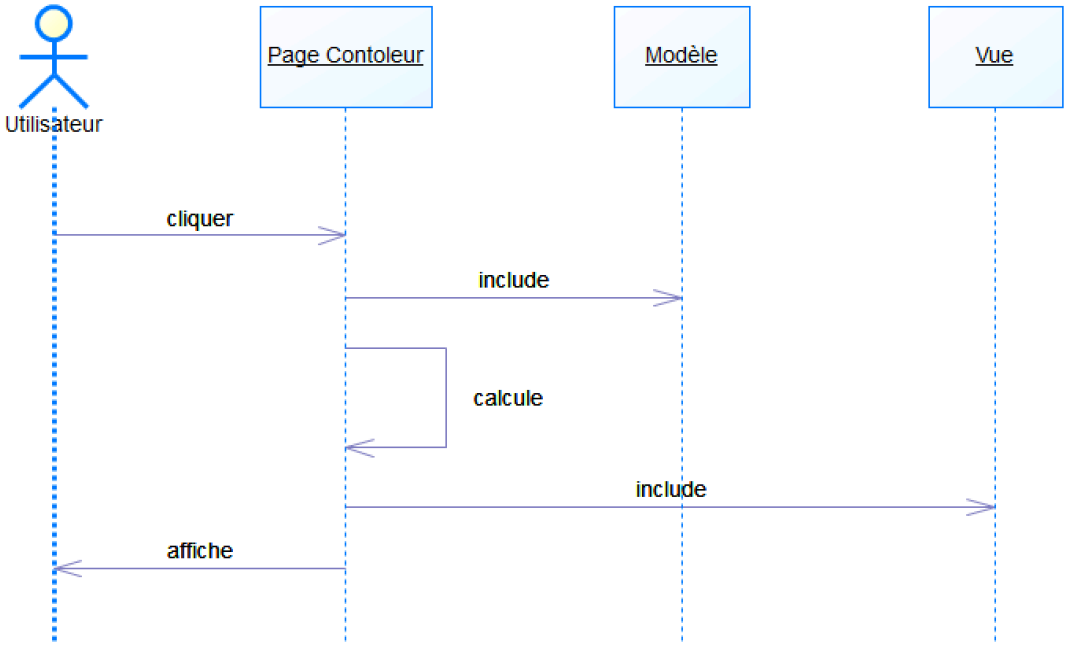
1. Il utilise les fonctions du Modèle (include et appel aux fonctions).
2. Il analyse et traite les données issues du Modèle mais aussi celles passées en paramètre à l'appel de la page ($_GET, $_POST, $_SESSION). Il détermine par exemple si le visiteur a le droit de voir la page ou non.
3. En fonction de ses calculs, il appel la vue correspondante par un include.

C'est une page pur php.

Le contrôleur est le « chef d'orchestre » : il récupère la demande de l'utilisateur à travers la vue (la page HTML) par un href, un formulaire ou un header. Il échange des données avec le modèle, fait les calculs (qui peuvent être complexes) puis choisit une vue à afficher en lui fournissant les variables.

Le rôle du contrôleur peut se limiter à faire le lien entre le modèle et la vue : de la colle !

Fonctionnement global



Un utilisateur, à travers une vue, fait appel à une page : un contrôleur (par un href ou un formulaire).
Le contrôleur « include » un modèle et utilise une des fonctions du modèle.
Il fait ensuite des calculs.
Selon les résultats, il include une vue ou une autre à afficher à l'utilisateur.
Et ainsi de suite.

MVC : design pattern tête la première

Car si l'modèle - est essentiel
Et si la vue - est magnifique
J'suis p'têt feignant - oui mais c'est fou
Ces lignes de code - qui sont d'la colle
Et c'code n'fait rien - d'vraiment magique
Il n'fait qu'transmettre - que des valeurs

On dit que le MCV est un design pattern 5DP). C'est en réalité un assemblage de DP élémentaires (un par lettre) : les DP « stratégie », « composite » et « observateur » (les DP ont des noms).

Si on code réellement ces DP, alors on aura une mise à jour automatique des notifications (DP observateur).

Organisation des fichiers dans le MVC

Organisation de répertoires MVC

À la racine du site, on crée 3 répertoires :

- modele
- vue
- controleur

Dans chacun de ces répertoires, on crée un sous-répertoire par « module » du site (par exemple : forum, blog, chat, etc.).

Les fichiers « modèle »

Chaque modèle est un fichier PHP contenant un appel à la BD et qui renvoie par exemple le tableau des résultats (fetchall plutôt que fetch un par un).

Le fichier PHP n'aura pas de balise fermante (?>) : ça évite des problèmes !

L'objet bdd sera déclaré en global pour ne pas avoir à refaire la connexion à chaque opération. Dans l'idéal, il faut utiliser un DP singleton qui permet de ne recréer un objet que s'il n'a pas déjà été créé.

Dans chaque sous-repertoire de module, on crée un fichier par fonction qui a le nom de la fonction. Ainsi on fait un include du fichier et un appel à la fonction.

Les fichiers « contrôleur »

Chaque contrôleur est aussi un fichier PHP. Il inclut le modèle (include_once) au début et à la fin il inclura la vue. Autrement dit, en réalité il contient tout le code, mais on sépare le code en 3 parties (modèle, contrôle, vue) et chaque partie est dans son propre fichier.

Le contrôleur fait les calculs avant l'appel de la vue.

Quand on demande un fichier dans la barre d'adresse, c'est un contrôleur qu'on appelle.

Les fichiers « vue »

Chaque vue est un fichier HTML qui fera un simple affichage du jeu de données fourni par le contrôleur. Il peut y avoir une boucle PHP pour parcourir les tableaux.

Il y a un fichier vue par page utilisateur.

Le font contrôleur = contrôleur global = routeur

Présentation

On se dotera d'un contrôleur global qui permet l'entrée dans le site et qui permet de choisir le contrôleur à appeler donc la page à afficher.

C'est le « front contrôleur » ou « routeur ». On retrouve le terme de « routeur » dans les frameworks.

Les autres contrôleurs sont parfois appelés backContoller.

Son rôle est de déterminer quel contrôleur appeler et de faire des initialisations générales (connexion à la BD, affichage d'en-tête ou de pied de page, etc.).

Le fichier index.html appellera le routeur avec une route particulière, c'est-à-dire un contrôleur particulier, donc une page particulière à afficher.

Le fichier index.html

Il appelle ici le fichier indexSwitch qui est le front controleur.

```
<?php
// on appelle indexSwith en settant indexArticle avec n'importe
quel valeur
header('location: indexSwitch.php?indexArticles=set');
?>
```

Le fichier indexSwitch.html : le routeur

IndexSwitch est le front contrôleur qui choisit le contrôleur à exécuter. Ce front controleur, c'est ce qu'on appellera le « routeur » dans les framework php.

```
<?php
session_start();//On démarre la session
include('modele/connexion_sql.php'); // connexion à la BD

include 'vues/entete.php'; //HTML de l'entête du site

// grand SWITCH d'accès aux pages

// LE BLOG
if (isset($_GET['indexArticles']) {
    include('controleur/blog/indexArticles.php');
}
elseif (!empty($_GET['indexCommentaires']) {
    include('controleur/blog/indexCommentaires.php');
}
//etc.

include 'vues/pied.php'; //HTML du pied de page
```

empty() ou isset()

C'est presque la même chose. empty() est vrai pour non défini, =0 ou =null. isset() est faux uniquement pour non défini.

http://php.net/manual/fr/function.empty.php

Include entête et pied

Si on a toujours les mêmes en-têtes et pieds de page

Déconnexion

Pour libérer proprement la BD.

En PDO, la déconnexion n'est pas utile si on a fait des close cursor

```
include('modele/deconnexion_sql.php'); // deconnexion à la BD
```

Variante

On pourrait avoir un routeur qui reçoit le nom du contrôleur en paramètre et qui ainsi appelle le contrôleur avec ce paramètre : ainsi on n'a plus besoin de swich :

```
...
//On inclut le contrôleur s'il existe et s'il est spécifié

if (!empty($_GET['page']) &&
is_file('controleurs/'.$_GET['page'].'.php'))

    include 'controleurs/'.$_GET['page'].'.php';

else
     include 'controleurs/accueil.php'; // controleur accueil

...
```

is_file()

Si on passe le nom du fichier en paramètre, il faut vérifier. On peut aussi le passer en dur.

Exemple : le blog

Cahier des charges

On affiche les articles. On peut sélectionner un article et l'afficher en entier avec ses commentaires. On peut ajouter des commentaires.

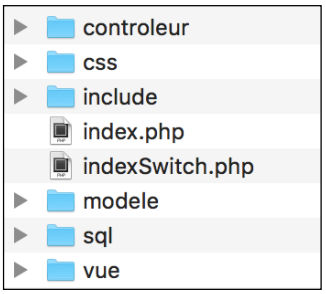
On peut circuler dans la listes des articles et celle des commentaires.

On peut passer en mode administrateur et ajouter des articles, mais aussi modifier et supprimer articles et commentaires.

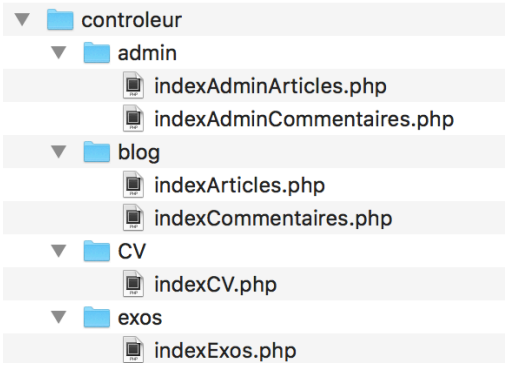
On peut aussi aller sur une page CV et sur une page exercices.

Bilan d'organisation

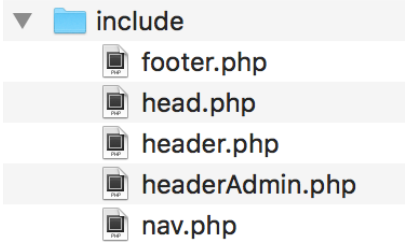
Dossiers et routeur



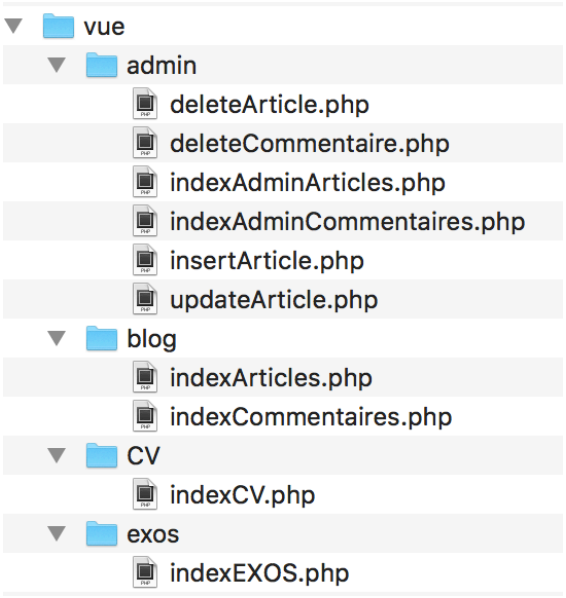
Les contrôleurs



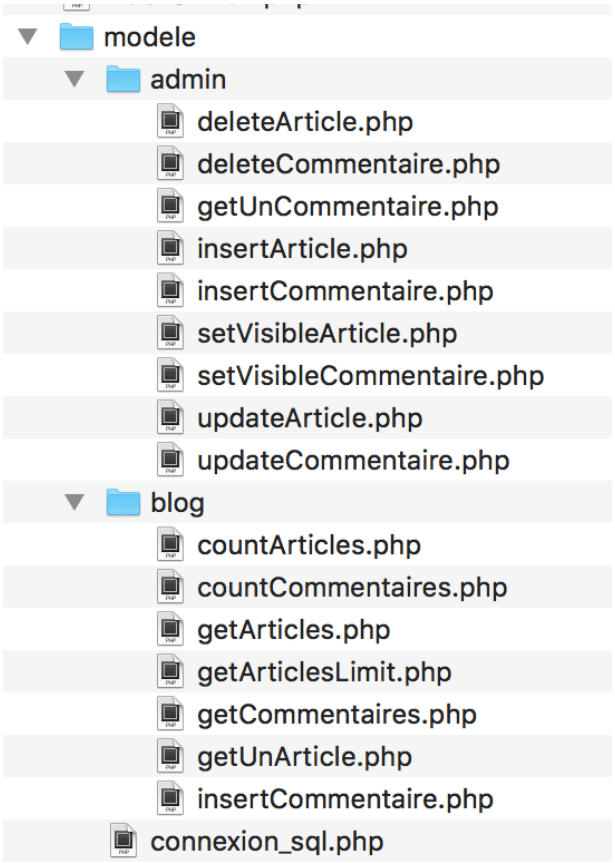
Les includes



Les vues


vue
admin
deleteArticle.php
deleteCommentaire.php
indexAdminArticles.php
indexAdminCommentaires.php
insertArticle.php
updateArticle.php
blog
indexArticles.php
indexCommentaires.php
CV
indexCV.php
exos
indexEXOS.php

Les modèles


modele
admin
deleteArticle.php
deleteCommentaire.php
getUnCommentaire.php
insertArticle.php
insertCommentaire.php
setVisibleArticle.php
setVisibleCommentaire.php
updateArticle.php
updateCommentaire.php
blog
countArticles.php
countCommentaires.php
getArticles.php
getArticlesLimit.php
getCommentaires.php
getUnArticle.php
insertCommentaire.php
connexion_sql.php

Exemple de contrôleur : controleur/blog/indexArticles.php

Inclusion du modèle

```
<?php
// MODELE : inclusion des fonctions
//////////////////////////////////////////////
    include_once('modele/blog/getArticles.php');
    include_once('modele/blog/getArticlesLimit.php');
    include_once('modele/blog/countCommentaires.php');
    include_once('modele/blog/countArticles.php');
```

Début du contrôleur

```
// CONTROLEUR
```

- ***Mise à jour des variables de sessions : si on est admin ou pas***

```
// INITIALISATION  des variables de SESSION
    // ENTREE par DECONNEXION de l'admin
    if(isset($_POST['deconnexion'])) unset($_SESSION['admin']);
```

- ***Mise à jour d'une variable : $tousLesArticles si on doit afficher tous les articles***

```
// MISE A JOUR de $tousLesArticles
    // ENTREE POST par boutonTousArticles
    //    ou par hidden sur haut, bas, monter descendre
    // ENTREE GET par retour des commentaires
    if( ( isset($_POST['tousLesArticles']) AND
$_POST['tousLesArticles']==1 ) OR
        ( isset($_GET['tousLesArticles'])  AND
$_GET['tousLesArticles']==1 ) OR
        isset($_POST['boutonTousLesArticles'])
    )
        $tousLesArticles=1;
    else
        $tousLesArticles=0;
```

- ***Mise à jour d'une variable : $debut pour savoir quels articles afficher***

C'est compliqué !!!

```
// MISE A JOUR de $debut
    // ENTREE par BOUTONS haut, bas, monter, descendre : POST -->
    // ENTREE par RETOUR de commentaires :                GET -->
    // il faut redonner la bonne valeur à $debut
    $nbMessages=3; // nombre de messages affichés en même temps
    $step=1; // la marche de montée et de descente
    $debut=0; // position de début d'affichage


    // si on revient d'un commmentaire : c'est par GET
    if(isset($_GET['debut'])) $debut=$_GET['debut'];

    //  si on a cliqué sur début, fin, suivant, précédent :
```

```
    //  $debut, c'est l'indice du premier message à afficher dans
la table : utilisé dans le SELECT
    //  on récupère la valeur du $debut de la page d'appel dans
$_POST['debut']
    //  on va calculer la nouvelle valeur de $debut selon les cas

     // si on veut monter tout en haut, $debut repasse à 0 (le
haut est à 0)
    if(isset($_POST['haut']) ){
        $debut=0;
        $tousLesArticles=0;
    }
    // on veut monter d'un step : aller vers 0 en décrémentant
debut de step
    // si la décrémentation reste >=0 on la fait, sinon debut
passe à 0
    elseif(isset($_POST['monter']) AND $tousLesArticles==0 ) {
        if ($_POST['debut']-$step>=0) $debut=$_POST['debut']-
$step;
        else $debut=0;
    }
    // si c'est bas, le début vaut $countArticles-$nbMessages
    elseif(isset($_POST['bas'])) {
        // on compte le nombre d'articles
        $countArticles=MODELE_countArticles();
        $debut=$countArticles - $nbMessages;
        $tousLesArticles=0;
    }
    elseif(isset($_POST['descendre']) AND !$tousLesArticles ) {
        // on compte le nombre d'articles
        $countArticles=MODELE_countArticles();
     // si on n'est pas tout en bas, on descend
     if($_POST['debut'] < $countArticles - $nbMessages)
       $debut=$_POST['debut']+$step;
     else // sinon on ne bouge pas
       $debut=$_POST['debut'];
    }
```

- ***Récupération des articles à afficher dans $articles en fonction de ce qu'on a fait avant***

```
// RECUPERATION DES ARTICLES
    if($tousLesArticles==0)
        $articles = MODELE_getArticlesLimit($debut, $nbMessages,
1); // 1 : les visibles
    else
        $articles = MODELE_getArticles(1); // les visibles
    // echo '<pre>'; print_r($articles); echo '</pre>';
```

Inclusion de la vue

```
// VUE : On affiche la page
////////////////////////////////////////////////////////
include_once('vue/blog/indexArticles.php');
```

```
}
```

Exemple de vue : vue/blog/indexArticles.php

Cette vue va utiliser les variables du contrôleur : $articles

```
<DOCTYPE html>
<html>
    <head>
        <title>Blog MVC - Articles</title>
        <meta charset="UTF-8" />
        <link href="css/style.css" rel="stylesheet"
type="text/css" />
    </head>
<body>
<?php include('include/header.php'); ?>
<section class="section">
<article>
```

Affichage du formulaire : avec $debut et $tousLesArticles

```
<!-- AFFICHAGE DE LA PAGE  : d'abord les boutons -->
<form action="indexSwitch.php" method="POST">
<p> Circuler dans les articles du blog :
    <input type="submit" name="haut" value="haut">
    <input type="submit" name="descendre" value="descendre">
    <input type="submit" name="monter" value="monter">
    <input type="submit" name="bas" value="bas">
    <input type="submit" name="boutonTousLesArticles" value="Tous
les articles">
    <input type="hidden" name="indexArticles" value=1>
    <input type="hidden" name="debut"
        value=' <?php  echo $debut; // debut circule de page en
page ?>'>
     <input type="hidden" name="tousLesArticles"
        value=' <?php  echo $tousLesArticles; // tousLesArticles
circule de page en page ?>'>
 </p>
</form>
```

Affichage des articles : avec $articles

```
<?php
//echo '<pre>'; print_r($billets); echo '</pre>';
foreach($articles as $article)
{
    //echo '<pre>'; print_r($article); echo '</pre>';
    ?>
    <div class="toutArticle">
        <h3>
            <?php echo htmlspecialchars($article['titre']).' -
id:'.htmlspecialchars($article['id']); ?>
        </h3>

        <div class="articleEtDate">
            <p class="dateArticle"><em><?php
                // AFFICHAGE DE LA DATE
                // on crée un objet date
```

```
                $date=date_create($article['dateCreation']);
                echo date_format($date,'d M Y').'<br/>';
                // ou echo $date->format('d M Y').'<br/>';
                echo date_format($date,'H:i:s');
            ?></em></p>
            <p class="article"> <?php
                // AFFICHAGE DU CONTENU DU BILLET
                // nl2br : new ligne to br : transforme les
passages à la ligne du texte en <br> html
                echo
nl2br(htmlspecialchars($article['contenu']));
```

Affichage du nombre de commentaires : on utilise une fonction du modèle

```
$nbCommentaires=MODELE_countCommentaires($article['id'], 1); // 1
pour les commentaires visibles?>
                <em><a
href="indexSwitch.php?indexCommentaires=1&amp;idArticle=<?php
echo $article['id'] ?>&amp;debut=<?php echo $debut
?>&amp;tousLesArticles=<?php echo $tousLesArticles ?>">
                    <br/> <?php echo $nbCommentaires;?>
commentaire<?php if ($nbCommentaires >1) echo 's';?>
                </a></em>
            </p>
        </div>
    </div>
<?php } ?>

</article>

<aside class="aside">
    Mon aside à remplir
</aside>

</section>
<?php include("include/footer.php"); ?>
</body>
</html>
```

Bilan

On a peu de code php dans la page. C'est gérable par un non expert.

On pourrait même récupérer $nbCommentaires dans le contrôleur pour éviter l'usage de la fonction MODELE_countCommentaires().

De même, le nettoyage du texte avec nl2br(htmlspecialchars()) pourrait être confié au contrôleur pour limiter la charge php de la vue.

Exemple de modèle : fichier modele/blog/getArticlesLimit.php

```
<?php
function MODELE_getArticlesLimit($offset, $limit, $visible)
{
```

On récupère le $bdd en le déclarant global

```
    global $bdd;
```

On caste les paramètres pour éviter les problèmes

```
    $offset = (int) $offset;
     $limit = (int) $limit;
     $visible = (int) $visible; // si visible vaut 10, (1 et 0
                                // on prend tous les articles
                              // sinon que ceux qui valent visible
```

Calcul de reqSQL

```
    if($visible==10){ // pour une raison obscure
        $reqSQL='
            SELECT id, titre, contenu, dateCreation, visible
            FROM articles
            ORDER BY dateCreation DESC LIMIT :offset, :limit
        ';
    }
    else{
        $reqSQL='
            SELECT id, titre, contenu, dateCreation, visible
            FROM articles
            WHERE visible= :visible
            ORDER BY dateCreation DESC LIMIT :offset, :limit
        ';
    }
    //  DATE_FORMAT(dateCreation, \'%d/%m/%Y à %Hh%imin%ss\') AS
dateCreationFR
```

On fait des prepare + bind + execute

```
    $requete = $bdd->prepare($reqSQL);
    $requete->bindValue(':offset', $offset, PDO::PARAM_INT);
    $requete->bindValue(':limit', $limit, PDO::PARAM_INT);
    if($visible!=10) $requete->bindValue(':visible', $visible,
PDO::PARAM_INT);
    $requete->execute();
    // echo '<pre>'; print_r($requete); echo '</pre>';
```

On fetchAll pour récupérer un tableau de résultats

```
    $articles = $requete->fetchAll();

    $requete->closeCursor();
```

On retourne le tableau de résultats

```
return $articles;
}
```

- bindParam permet d'associer une valeur à un alias dans une requête. Ainsi le execute est allégé.
- L'intérêt est de contrôler le type de données avec le PDO::PARAM_INT
- On peut utiliser bindParam ou bindValue. bindValue fonctione par valeur : à l'exécution, on prend la valeur passée « en dur ». bindParam fonctionne par adresse. On peut donc faire plusieurs execute de la même requête préparée et bindParamée en changeant la valeur de la variable dont l'adresse est bindParamée.
- http://php.net/manual/fr/pdostatement.bindvalue.php
- http://php.net/manual/fr/pdostatement.bindparam.php

PROJET : QUIZZ en ligne

Application de quiz en ligne.

Méthode générale

Problème n°1 : la conception de la BD.

Le projet se fait par étape. On sauvegarde chaque étape et on fait monter la progression à partir de l'étape précédente.

L'application doit être restructurée à chaque fois pour garder une application propre.

On utlise le versionnement en conservant les étapes. Dans l'idéal, il faudrait utiliser le versionnement Git pour marquer les différentes versions du projet (repository Git, commits progressifs et commits de "release », usage pertinent des branches).

Etape 1 :

Un quiz comporte entre 1 et 10 questions

Pour chaque question, on doit choisir 1 réponse parmi 2 à 5 choix de réponses proposés

Lorsque le quiz est envoyé, on affiche les résultats : un score (en % de bonnes réponses) et la liste des bonnes et mauvaises réponses s'affiche. Si on a fait une erreur, on doit nous indiquer quelle était la bonne réponse

Une page d'accueil recense tous les quiz. Chaque quiz a un titre.

Etape 2 :

Au lieu d'une seule réponse possible, on doit pouvoir sélectionner plusieurs réponses.

Certaines questions n'autorisent qu'une seule réponse (vous afficherez des boutons radio) et certaines questions autorisent plusieurs réponses (vous afficherez des cases à cocher).

On veut que la personne ne puisse faire le quiz qu'une seule fois. Dès que le quiz a été rempli, il n'est plus possible de le faire à nouveau.

Etape 3 :

On veut aussi pouvoir proposer 2 autres types de questions :

Un nombre à saisir

Des éléments à ordonner

Un quiz peut donc être composé de 4 questions.

De plus, il faut que l'utilisateur puisse refaire le quiz autant de fois qu'il le veut et qu'il puisse afficher l'historique de ses réponses.