OUHEMMI
Imrane
SLAM1

Synthèse MISSION 1 ET 2 AgoraBo

Contexte du projet :

Nous sommes embauché dans une association d'éducation populaire de type loi 1901 : L a MJC AGORA. Son but est de développer l'épanouissement de la personne grâce à l'éducation et la culture pour tous. Mais, elle détient pas d'un service informatique. Notre mission principale est d'aider au développement de leur site d'administration des données.

Mission 1 :

L'application est développée en mode client/serveur et utilise PHP sur un serveur MySQL. Notre objectif est de rendre accessible cette application pour les employés voulant travailler à distance.

Les dossiers utilisés :

	
	→ Connexion à la base de données
	→ Générer la réponse à la requête HTTP demandée par l'application cliente. Il analyse et traite la requête.
	→Accès et manipulation des données
	→ L'interface et manipulation des données
	→Les fichiers destinés aux utilisateur d'un site (images, css, jvscript)
	→ Il s'agit du contrôleur principal

app > _config.inc.php > DB_PWD

```php
<?php
/**
 * paramètres de configuration de l'application AgoraBo
 *
 * @package default
 * @author md
 * @version    1.0
 */

// gestion d'erreur
ini_set('error_reporting', E_ALL);      // en phase de développement
//ini_set('error_reporting', 0);        // en phase de production

// constantes pour l'accès à la base de données
define('DB_SERVER', 'localhost');   // serveur MySql
define('DB_DATABASE', 'jeux');      // nom de la base de données
define('DB_USER', 'root');          // nom d'utilisateur
define('DB_PWD', '');               // mot de passe
define('DSN','mysql:dbname='.DB_DATABASE.';host='.DB_SERVER);

?>
```

Les lignes 15 à 18 permettent d'accéder à la BDD. On retrouve le serveur, le nom de la BDD ainsi que le nom d'utilisateur et mot de passe.

Nous allons nous intéresser à la table Pegis.

	idPegi	ageLimite	descPegi
Éditer Copier Supprimer	1	3	« Adapté à toutes les classes d'âge ». En effet, i...
Éditer Copier Supprimer	2	7	Déconseillé aux moins de 7 ans. Il contient des sc...
Éditer Copier Supprimer	29	5	ouii
Éditer Copier Supprimer	4	16	Déconseillé aux moins de 16 ans. Contenus possible...
Éditer Copier Supprimer	5	18	« destiné aux adultes ». Il peut contenir un degré...

Notre objectif, va être de modifier ajouter, supprimer et afficher ces informations.

Les fonctions permettant cela se trouvent dans le modèle.

```php
/**
 * Retourne tous les genres sous forme d'un tableau d'objets
 *
 * @return array le tableau d'objets  (Genre)
 */
0 references | 0 overrides
public function getLesPegis(): array {
    $requete =  'SELECT idPegi as identifiant, ageLimite as age, descPegi as description
                 FROM pegi
                 ORDER BY idPegi';
    try {
        $resultat = PdoJeux::$monPdo->query($requete);
        $tbPegis  = $resultat->fetchAll();
        return $tbPegis;
    }
    catch (PDOException $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}
```

```php
/**
 * Ajoute un nouveau genre avec le libellé donné en paramètre
 *
 * @param string $libGenre : le libelle du genre à ajouter
 * @return int l'identifiant du genre crée
 */
0 references | 0 overrides
public function ajouterPegi(int $ageLimite, string $descPegi): int {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("INSERT INTO pegi "
                . "(idPegi, ageLimite, descPegi) "
                . "VALUES (0, :unAgeLimite, :unDescPegi) ");
        $requete_prepare->bindParam(':unAgeLimite', $ageLimite, PDO::PARAM_INT);
        $requete_prepare->bindParam(':unDescPegi', $descPegi, PDO::PARAM_STR);

        $requete_prepare->execute();
        // récupérer l'identifiant crée
        return PdoJeux::$monPdo->lastInsertId();
    } catch (Exception $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}


/**
 * Modifie le libellé du genre donné en paramètre
 *
 * @param int $idGenre : l'identifiant du genre à modifier
 * @param string $libGenre : le libellé modifié
 */
0 references | 0 overrides
public function modifierPegi(int $idPegi, int $ageLimite, string $descPegi): void {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("UPDATE pegi "
                . "SET AgeLimite = :unAgeLimite, descPegi = :unDescPegi"
                . "WHERE pegi.idPegi = :unIdPegi");
        $requete_prepare->bindParam(':unIdPegi', $idPegi, PDO::PARAM_INT);
        $requete_prepare->bindParam(':unAgeLimite', $ageLimite, PDO::PARAM_INT);
        $requete_prepare->bindParam(':unDescPegi', $descPegi, PDO::PARAM_STR);
        $requete_prepare->execute();
    } catch (Exception $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}
```

Pour afficher les pegis nous allons utiliser un tableaux et afficher chaque ligne.

On utilise du SQL dans toutes les fonctions.

Pour afficher les pegis nous allons utiliser un contrôleur secondaire nommé c_gererpegis.php

```php
if (!isset($_POST['cmdAction'])) {
    $action = 'afficherPegis';
}
else {
    // par défaut
    $action = $_POST['cmdAction'];
}

$idPegiModif = -1;      // positionné si demande de modification
$notification = 'rien'; // pour notifier la mise à jour dans la vue

// selon l'action demandée on réalise l'action
switch($action) {

    case 'ajouterNouveauPegi': {
        if (!empty($_POST['txtAgeLimite'])&&!empty($_POST['txtDescPegi'])) {
            $idPegiNotif = $db->ajouterPegi($_POST['txtAgeLimite'],$_POST['txtDescPegi']) ;
            // $idGenreNotif est l'idGenre du genre ajouté
            $notification = 'Ajouté';   // sert à afficher l'ajout réalisé dans la vue
        }
      break;
    }

    case 'demanderModifierPegi': {
            $idPegiModif = ($_POST['txtIdPegi']); // sert à créer un formulaire de modification pour ce genre
        break;
    }

    case 'validerModifierPegi': {
        $db->modifierPegi($_POST['txtIdPegi'],$_POST['txtAgeLimite'], $_POST['txtDescPegi']);
        $idPegiNotif = ($_POST['txtIdPegi']); // $idGenreNotif est l'idGenre du genre modifié
        $notification = 'Modifié';  // sert à afficher la modification réalisée dans la vue
        break;
    }

    case 'supprimerPegi': {
        $idPegi = $_POST['txtIdPegi'];
        $db-> supprimerPegi($_POST['txtIdPegi']);
        break;
    }
}

// l' affichage des genres se fait dans tous les cas
$tbPegis  = $db->getLesPegis();
require 'vue/v_lesPegis.php';
```

Les cases vont être les déclencheurs des demandes de modification, ajout, suppression et validation.

Pour afficher les pegis, il faut définir un tableau contenant le résultat de la requête dans la fonction getLesPegis().

Pour afficher le tableau, il faut avoir accès à la vue v_lesPegis.php. Il faut donc utiliser un REQUIRE permettant de lier ces deux pages php.

Dans cette pages v_lesPegis.php, on fait une boucle pour afficher chaque lignes.

```php
<?php
foreach ($tbPegis as $pegi) {
?>
  <tr>
    <!-- formulaire pour modifier et supprimer les genres-->
    <form action="index.php?uc=gererPegis" method="post">
    <td><?php echo $pegi->identifiant; ?><input type="hidden"  name="txtIdPegi" value="<?php echo $pegi->identifiant; ?>" /></td>
    <td><?php echo $pegi->age; ?></td>
    <td><?php
        if ($pegi->identifiant != $idPegiModif) {
            echo $pegi->description;
            ?>
            </td><td>
                <?php if ($notification != 'rien' && $pegi->identifiant == $idPegiNotif) {
                    echo '<button class="btn btn-success btn-xs"><i class="fa fa-check"></i>' . $notification . '</button>';

                } ?>
                <button class="btn btn-primary btn-xs" type="submit" name="cmdAction" value="demanderModifierPegi" title="Modifier"><i class="fa fa-pencil"></i></button>
                <button class="btn btn-danger btn-xs" type="submit" name="cmdAction" value="supprimerPegi" title="Supprimer" onclick="return confirm('Voulez-vous vraiment suppri
            </td>
        <?php
        }
        else {
            ?>
            <input type="number" id="txtAgeLimite" name="txtAgeLimite" size="24" min="1"  max="100"   value="<?php echo $pegi->age; ?>" />
            <input type="text" id="txtDescPegi" name="txtDescPegi" size="24" required minlength="4"  maxlength="24"   value="<?php echo $pegi->description; ?>" />
            </td>
            <td>
                <button class="btn btn-primary btn-xs" type="submit" name="cmdAction" value="validerModifierPegi" title="Enregistrer"><i class="fa fa-save"></i></button>
                <button class="btn btn-info btn-xs" type="reset" title="Effacer la saisie"><i class="fa fa-eraser"></i></button>
                <button class="btn btn-warning btn-xs" type="submit" name="cmdAction" value="annulerModifierPegi" title="Annuler"><i class="fa fa-undo"></i></button>
            </td>
        <?php
```

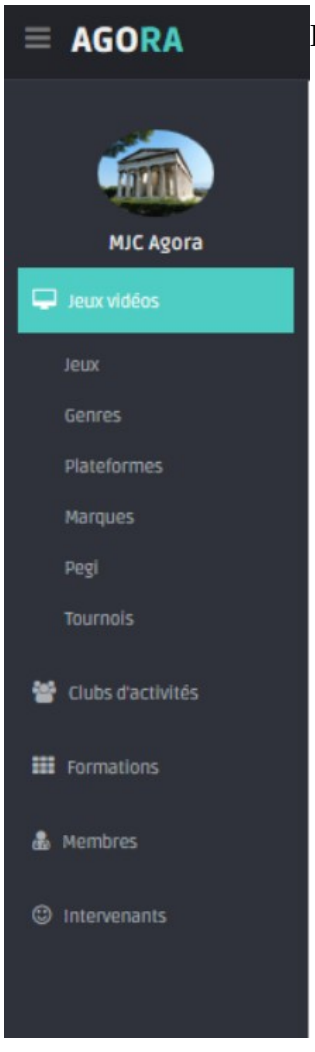
On retrouve également les demandes de modification, suppression et validation. Ces demandes sont utilisés avec des boutons.

Ce contrôleur secondaire est relier à un contrôleur principal appelé index.php pour des raisons de sécurité. Index.php est le seul point d'entrée dans l'application.

```php
require 'vue/v_header.html';    // entête des pages HTML

// inclure les bibliothèques de fonctions
require_once 'app/_config.inc.php';
require_once 'modele/class.PdoJeux.inc.php';

// Connexion au serveur et à la base (A)
$db = PdoJeux::getPdoJeux();

// Récupère l'identifiant de la page passé via l'URL
// Si non défini, on considère que la page demandée est la page d'accueil
if (!isset($_GET['uc'])){
    $_GET['uc'] = 'index';
}
$uc = $_GET['uc'];

// selon la valeur du use case demandé(uc) on inclut le contrôleur secondaire
switch($uc){
    case 'index' : {
        $menuActif = '';
        require 'vue/v_menu.php';
        require 'vue/v_accueil.html'; break;
    }
    case 'gererGenres' : {
        $menuActif = 'Jeux';    // pour garder le menu correspondant ouvert
        require 'vue/v_menu.php';
        require 'controleur/c_gererGenres.php';
        break;
    }
     case 'gererPlateformes' : {
        $menuActif = 'Jeux';    // pour garder le menu correspondant ouvert
        require 'vue/v_menu.php';
        require 'controleur/c_gererPlateformes.php';
        break;
    }
     case 'gererMarques' : {
        $menuActif = 'Jeux';    // pour garder le menu correspondant ouvert
        require 'vue/v_menu.php';
        require 'controleur/c_gererMarques.php';
        break;
    }
     case 'gererPegis' : {
        $menuActif = 'Jeux';    // pour garder le menu correspondant ouvert
        require 'vue/v_menu.php';
        require 'controleur/c_gererPegis.php';
        break;
    }
    case 'gererJeux' : {
        $menuActif = 'Jeux';    // pour garder le menu correspondant ouvert
        require 'vue/v_menu.php';
        require 'controleur/c_gererJeux.php';
        break;
    }
}

// Fermeture de la connexion (C)
$db = null;

// pied de page
require("vue/v_footer.html") ;
```

Résultat final :

La page d'accueil



La pages correspondants aux différentes tables dans la BDD

On distingue les boutons et les mêmes informations présentes dans la table.

L'utilisation du HTML a permis d'afficher les demandes faites. Le PHP a permis de se connecter à la BDD et aussi d'exécuter un certain nombre d'actions avec l'utilisation d'une BDD.

Difficultés rencontrées :

Il y a plusieurs tables qui doivent être affichées avec la possibilité de les modifier, afficher…

Ici, il s'agit que d'un exemple pour une table. J'ai rencontré des difficultés pour la modification de la table Jeux. Je n'ai pas compris la manière dont il fallait procéder pour la modifier.

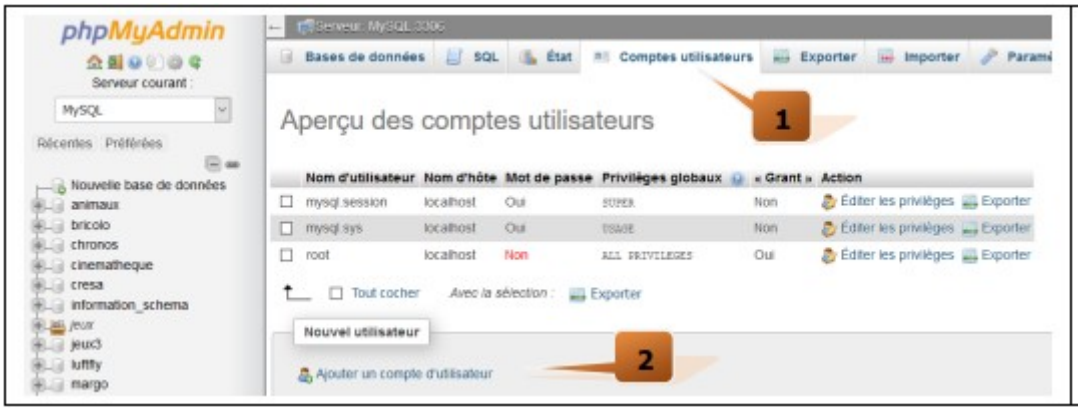
Référence Jeu	Identifiant Plateforme	Pegi	Genre	Marque	Nom	Prix	Date de parution
Nouveau	Plateforme	Pegi	Genre	Marque	Nom	Prix	Date de parution
BF8763098765	2	3	10	2	FIFA 18 - Édition essentielle	59.99	2017-09-29
U174645475GT	2	3	10	1	Gran Turismo 6	21.50	2013-06-12
ES47562098754	4	2	2	13	La Légende de Zelda - The Wind Waker HD	29.80	2016-04-15
ET86987453T5	7	5	1	10	La terre de milieu : L'Ombre de la Guerre	59.90	2017-10-10
YT65487BJI	3	1	2	13	Mario Kart 7	39.90	2012-11-28
ER6753FG987	3	3	2	1	Minecraft Story Mode - L'aventure Complète -	39.89	2016-12-16
TF98653JU8	15	3	2	1	Minecraft Story Mode - L'aventure Complète -	39.89	2016-12-16
RT4958673II2	4	2	2	13	Nouveau Super Mario Bros.	18.90	2016-04-15
CF47563837YG	3	1	13	8	Paddington : escapades à Londres	18.30	2015-06-19
EG763547598RF	3	2	6	13	Pokémon X	39.90	2013-10-12
ER493746Y78	8	5	1	3	Rise of the Tomb Raider	19.90	2015-11-13

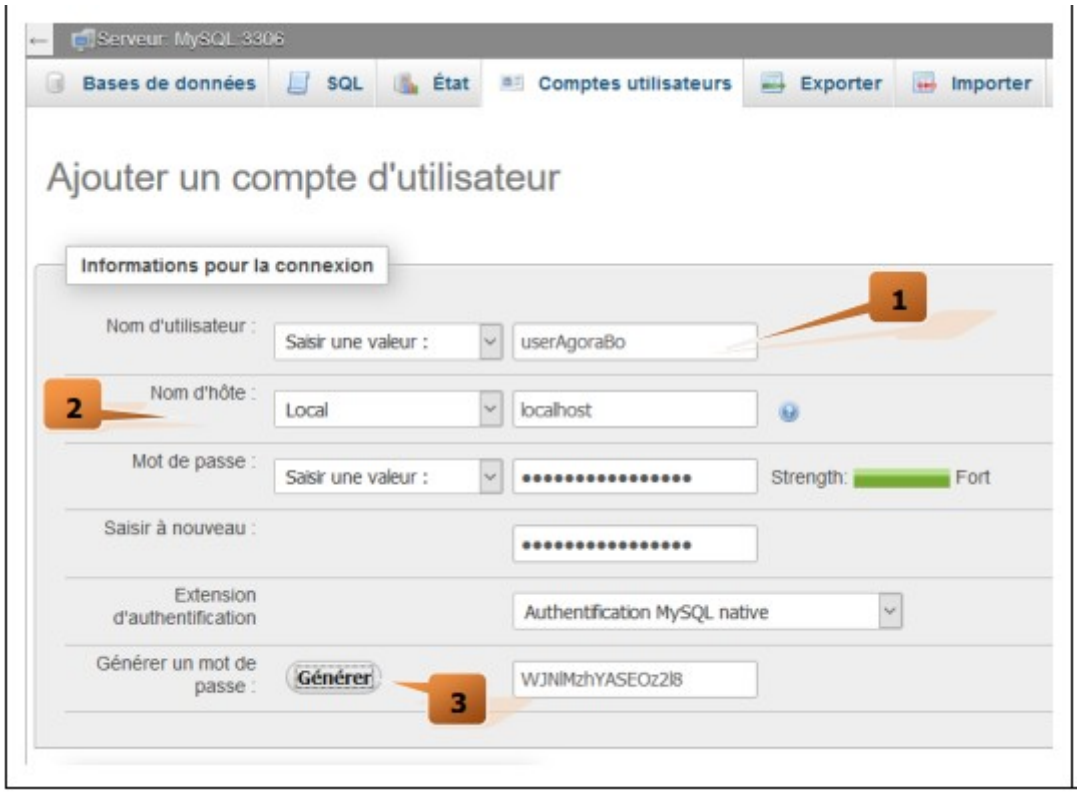
Mission 2 :

L'objectif de cette mission est de créer un formulaire d'authentification qui permettra aux utilisateurs de saisir leurs données d'identification pour accéder à l'application.

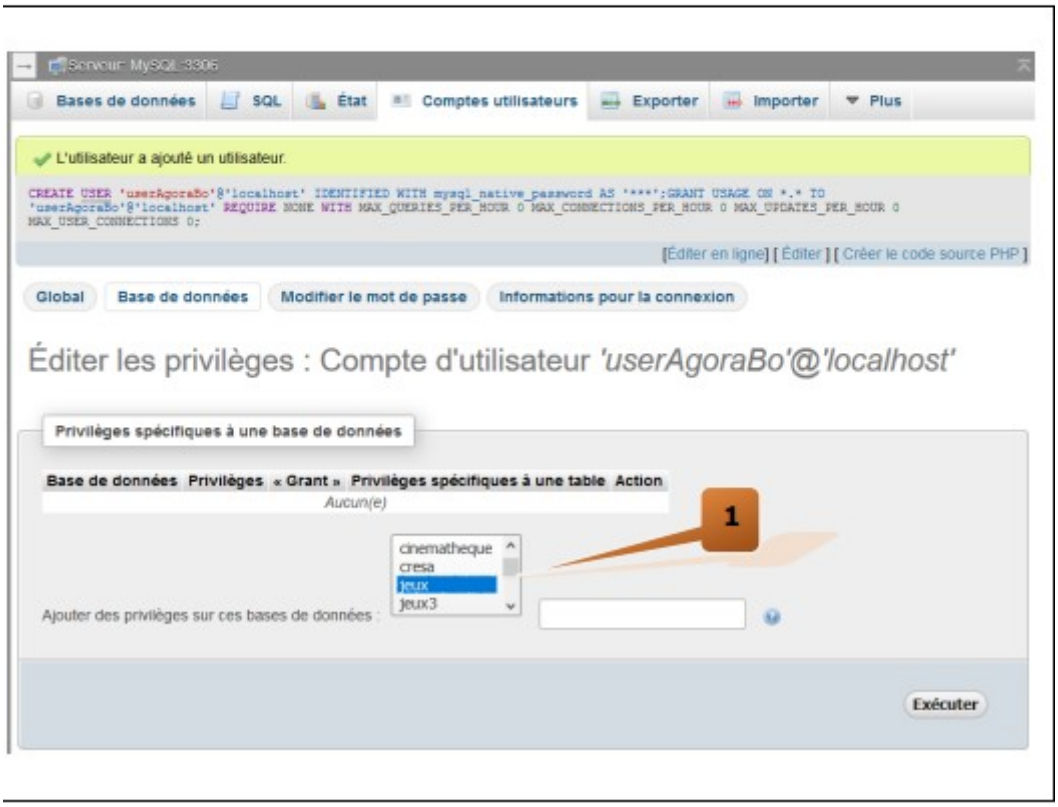
Il existe plusieurs manière de procéder à une authentification mais nous allons la réaliser au moyen de mots de passe.

Pour des raisons de sécurités, nous allons créer un utilisateur avec des droits restreints sur la BDD pour éviter qu'un éventuel pirate exécute des ordres SQL avec les droits du compte root.





On créé notre compte.



Il faudra enfin sélectionner les données de la BDD « Jeux » afin de les avoir dans le compte d'utilisateur.

Il faudra changer les données de connexion à la BDD dans APP.

```php
<?php
/**
 * paramètres de configuration de l'application AgoraBo
 *
 * @package default
 * @author md
 * @version    1.0
 */

// gestion d'erreur
ini_set('error_reporting', E_ALL);      // en phase de développement
//ini_set('error_reporting', 0);        // en phase de production

// constantes pour l'accès à la base de données
define('DB_SERVER', 'localhost');   // serveur MySql
define('DB_DATABASE', 'jeux');      // nom de la base de données
define('DB_USER', 'userAgoraBo');           // nom d'utilisateur
define('DB_PWD', 'LKJSDH84DzDfdF52E');                  // mot de passe
define('DSN','mysql:dbname='.DB_DATABASE.';host='.DB_SERVER);

?>
```

Comment fonctionne l'authentification ?

```php
public function getUnMembre(string $logMembre, string $mdpMembre): ?object {
   try {
       // préparer la requête
       $requete_prepare = PdoJeux::$monPdo->prepare(
           'SELECT idMembre, prenomMembre, nomMembre, mdpMembre, selMembre
           FROM membre
           WHERE loginMembre = :unLoginMembre');
       // associer les valeurs aux paramètres
       $requete_prepare->bindParam(':unLoginMembre', $logMembre, PDO::PARAM_STR);
       // exécuter la requête
       $requete_prepare->execute();
       if ($utilisateur = $requete_prepare->fetch()){
           $mdpHash = hash('SHA512', $mdpMembre.$utilisateur->selMembre);
           // vérifier le mot de passe
           if ($utilisateur->mdpMembre==$mdpHash){
               return $utilisateur;
           }
           else{
               return null;
           }

       }else{
           return null;
           }

       // le mot de passe transmis par le formulaire est le hash du mot de passe saisi
       // le mot de passe enregistré dans la base doit correspondre au hash du (hash transmis concaténé au sel)
       } catch (PDOException $e) {
           die('<div class = "erreur">Erreur dans la requête !<p>'
           .$e->getmessage().'</p></div>');
       }
```

Voici la fonction qui vérifie le login et mot de passe
Dans un premier temps, il faut entrer en argument un login et un mot de passe
On va utiliser un hash du mot de passe.

Un hash va convertir un texte en un chaine de caractère assez courte. Il est impossible de déchiffrer le hash pour revenir au texte d'origine.

Pour utiliser le hash il nous faudra un fichier java script fourni au préalable nommé sha512.js.

c_connexion.php

```php
if (!isset($_POST['cmdAction'])){
    $action = 'demanderConnexion';
} else {
    // par défaut
    $action = $_POST['cmdAction'];
}

switch ($action) {
    case 'demanderConnexion': {
        require 'vue/v_connexion.php';
        break;
    }
    case 'validerConnexion': {
        // vérifier si l'utilisateur existe avec ce mot de passe
        $utilisateur = $db->getUnMembre($_POST['txtLogin'], $_POST['hdMdp']);
        // si l'utilisateur n'existe pas
        //if(isset($utilisateur)) {
        if ($utilisateur==NULL){
            // positionner le message d'erreur $erreur
            $erreur = 'Identifiant ou mot de passe incorrect';
            // inclure la vue correspondant au formulaire d'authentification
            require 'vue/v_connexion.php';
        } else {
            // créer trois variables de session pour id utilisateur, nom et prénom
            $_SESSION['idUtilisateur'] = $utilisateur->idMembre;
            $_SESSION['nomUtilisateur'] = $utilisateur->nomMembre;
            $_SESSION['prenomUtilisateur'] = $utilisateur->prenomMembre;

            // redirection du navigateur vers la page d'accueil
            header('Location: index.php');
            exit;
        }
        break;
    }
}
```

demande de validation, après validation la fonction getUnMembre() s'exécute. Si les log et mdp sont incorrectes, on retourne à la page de connexion sinon on redirige vers la page d'accueil en utilisant un REQUIRE

v_connexion.php

```
16                      echo '<dix class = "erreirCnx"><p>' .$erreur.'</p></div>';
17                  }
18                  ?>
19                  <input type="text" class="form-control" name="txtLogin" id="txtLogin"
20      placeholder="Login" required autofocus />
21                  <br>
22                  <input type="password" class="form-control" name="txtMdp" id="txtMdp"
23      placeholder="Mot de passe" required />
24              <div class="pull-right login-social-link">
25              <!--La version 5 du langage HTML a ajouté un nouveau type d'attribut, qui
26      n'est pas interprété par le navigateur.
27              Ce sont tous les attributs dont le nom commence par les lettres data.
28              L'attribut data-toggle contient le type d'événement qui va être lié à un
29      bouton
30                  data-toggle=modal /* Bouton qui ouvre une fenêtre modale -->
31              <a data-toggle="modal" href="v_connexion.php#myModal"> Mot de passe oublié
32      ?</a>
33              </div>
34              <button class="btn btn-theme btn-block" type="submit" name="cmdAction"
35          value="validerConnexion"
36                  title="Se connecter" onclick="document.getElementById('hdMdp').value =
37  hex_sha512(document.getElementById('txtMdp').value);document.getElementById('txtMdp').valu
38  e = ' ';">
39  <i class="fa fa-lock"></i> Se connecter</button>
40                  <!-- champ caché pour le mot de passe haché -->
41                  <input type="hidden" name="hdMdp" id="hdMdp" />
```

Les lignes 36 et 37 correspondent au valeur des champ saisi (txtMdp) et caché (hdMdp).

Dans index.php il faut un session_start() pour démarrer la session après authentification.

```php
<?php
    //supprimer la session
    session_destroy();
    // redirection vers le controleur principal !!!!! pas de HTML avant !!!!!
    header('Location : index.php');
?>
```

On supprime cette session une fois que l'utilisateur s'est déconnecté.

Voici le formulaire final. →

Les difficultés rencontrées : trouver les erreurs dans c_connexion dans la création des 3 variables id utilisateur, nom et prénom.

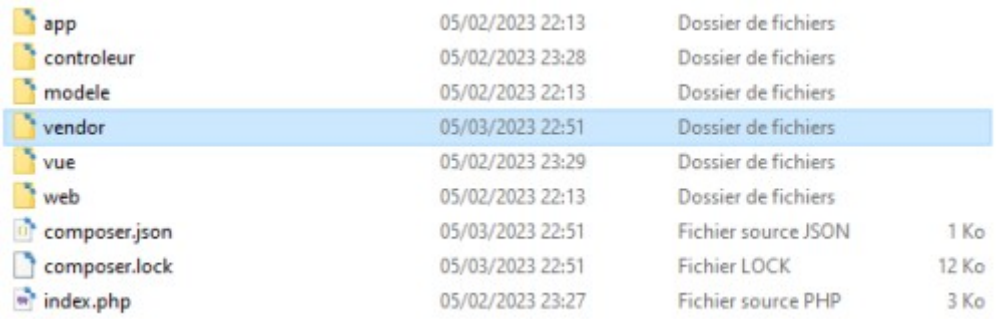
MISSION 3 AGORA : **TWIG**

Qu'est ce que twig ?

Twig est un moteur de template qui nous permettra de séparer la présentation des données HTML du traitement PHP. Il va rendre le code plus lisible et aussi sécuriser automatiquement les variable. Il s'agit d'un langage rapide.

L'objectif est d'apprendre à utiliser twig dans notre projet AgoraBo, afin d'améliorer la lisibilité et la maintenabilité du code des vues.

Après l'installation de twig, un dossier vendor se créera. Il contient les bibliothèques externe à l'application.



Le fichier « composer.json » contient les dépendances à installer pour l'application.

Modification, traduction de fichiers :

Il faut dans un premier temps, créer un fichier **layout.html.twig** dans la **vue.**

```
      {% if (session.idUtilisateur is defined) %}
        <a href="" class="userAgo"> {{ session.prenomUtilisateur }} {{ session.nomUtilisateur
}}</a>
```

session.idUtilisateur désigne la variable de session **$_SESSION[‘idUtilisateur’]** en PHP.
Cette partie est donc traduite en twig.

```
    {% block menu %}
      {% if (session.idUtilisateur is defined) %}
        {{ include('menu.html.twig') }}
      {% endif %}
    {% endblock %}
```

On inclus le **menu.html.twig.**

Ensuite il faut créer un fichier **v_headerTwig.html** dans la **vue.** Il remplacera le fichier **v_header.php**. Et contiendra le début de ce fichier.

Ensuite, nous allons créer un fichier **menu.html.twig** qui contiendra le code du **v_menu.php.**

```
{% if menuActif is defined and menuActif == 'Formations' %} class="active"  {% endif %}
```

Il faut traduire le PHP en twig.

Il faut également créer un fichier **accueil.html.twig**, qui contiendra le contenu du fichier **v_accueil.php**.

```
{% extends "layout.html.twig" %}
{% block central %}
```

Cette template héritera de la template de base.

Nous allons également créer une nouvelle vue **connexion.html.twig**, et ce fichier héritera aussi du **layout.html.twig**.

Dans l'**index.php**, il va falloir retirer le require du **v_header.php** et le remplacer par le **v_headerTwig.html.**

```
require 'vue/v_headerTwig.html';  //balise <head> et styles
```

```
{# Bonjour <?php echo $_SESSION["prenomUtilisateur"].' '.$_SESSION["nomUtilisateur"] ; ?>#}
 Bonjour {{ session.prenomUtilisateur }} {{ session.nomUtilisateur }}
```

Voici la différence d'affichage d'un texte en **PHP** et en **twig**. Ici on demande d'afficher le nom et prénom de l'utilisateur.

```
switch ($uc) {
    case 'index' :
        {
            echo $twig->render('accueil.html.twig');
            break;
        }
        case 'gererGenres' : {
            require 'controleur/c_gererGenres.php';
            break;
        }
        case 'gererPlateformes' : {
            require 'controleur/c_gererPlateformes.php';
            break;
        }
        case 'gererMarques' : {
            require 'controleur/c_gererMarques.php';
            break;
        }
        case 'gererPegis' : {
            require 'controleur/c_gererPegis.php';
            break;
        }
        case 'gererJeux' : {
            require 'controleur/c_gererJeux.php';
            break;
        }

        // . . . .autres cas à conserver
```

Le contenu de chaque case est également à supprimer. Chaque template **.twig** appelée héritera du menu.

On retire également le require du **v_footer.html** car il est déjà inclus dans la template de base.

Dans le fichier de l'**app**, on va ajouter les nouvelles constantes.

```
// constantes pour twig
define('TWIG_CACHE', false);          // mise en cache, en production à remplacer par
'/path/to/compilation_cache'
define('TWIG_DEBUG', true);           // mode debug
```

Dans le contrôleur **c_connexion.php**, il faudra faire des changements dus aux modifications et ajouts fait précédemment.

```
echo $twig->render('connexion.html.twig');
```

Nous allons ensuite modifier la vue de chaque page genre, plateforme, pegis…

Modifications pegis :

```
{% for key, pegi in tbPegis %}
    <tr>
        <!-- formulaire pour modifier et supprimer les genres-->
        <form action="index.php?uc=gererPegis" method="post">
            <td>{{ pegi.identifiant }}<input type="hidden" name="txtIdPegi" value="{{ pegi.identifiant }}"/></td>
            <td>
                {% if pegi.identifiant != idPegiModif %}
                    {{ pegi.age }}</td>
            </td>
            <td>
                {{ pegi.description }}
            </td>
            <td>
                {% if notification != 'rien' and pegi.identifiant == idPegiNotif %}
                    <button class="btn btn-success btn-xs">
                        <i class="fa fa-check"></i>
                        {{notification }}</button>
                {% endif %}
```

Nous allons convertir en twig le formulaire de modification.

On parcourt chaque ligne des pegis avec le « **for key, pegi in tbPegis »**
On récupère l'identifiant.
On récupère également l'âge et la description.

Afin de pouvoir modifier les pegis, il faut aussi faire les modifications nécessaires dans le contrôleur **c_gererlespegis.php**.

```
$idPegiModif = -1;      // positionné si demande de modification
$notification = 'rien'; // pour notifier la mise à jour dans la vue
$idPegiNotif = -1; // positionné si mise à jour dans la vue
```

On ajoute la variable **idPegiNotif**.

```
$tbPegis = $db->getLesPegis();
echo $twig->render('lesPegis.html.twig', array(
    'menuActif' => 'Jeux',
    'tbPegis' => $tbPegis,
    'idPegiModif' => $idPegiModif,
    'idPegiNotif' => $idPegiNotif,
    'notification' => $notification
));
```

On affiche les pegis
On fait correspondre les variables PHP avec les variables en string correspondant aux variables twig.

Je n'ai pas eu de réel problème lors de l'utilisation de twig sauf un. Lors de la connexion, j'avais un problème avec le hachage qui a disparu tout seul.

MISSION 4 : Symfony

L'objectif est d'utiliser le framework Symfony sur Agora.
Symfony est rapide et efficace. Il permet de maintenir la stabilité des applications. Symfony est très rechercher dans le monde du travail.

```php
 * @Route("/jeux", name="jeux_afficher")
 */
0 references | 0 overrides
public function index(SessionInterface $session)
{
    if ($session->has('idUtilisateur')) {
        $db = PdoJeux::getPdoJeux();
        return $this->afficherJeux($db, -1, -1, 'rien');
    } else {
        return $this->render('connexion.html.twig');
    }
}
/**
 * @Route("/jeux/ajouter", name="jeux_ajouter")
 */
0 references | 0 overrides
public function ajouter(SessionInterface $session, Request $request)
{
    $db = PdoJeux::getPdoJeux();
    if (!empty($request->request->get('txtNom'))) {
        $refJeuNotif = $db->ajouterJeu(
            $request->request->get('txtRefJeu'), $request->request->get('txtIdPlateforme'), $request->request->get('txtIdPegi')
            , $request->request->get('txtIdGenre'), $request->request->get('txtIdMarque'), $request->request->get('txtNom'), $request->request->get('txtPrix')
            , $request->request->get('txtDateParution')
        );
        $notification = 'Ajouté';
    }
    return $this->afficherJeux($db, -1, $refJeuNotif, $notification);
}
/**
 * @Route("/jeux/demandermodifier", name="jeux_demandermodifier")
 */
0 references | 0 overrides
public function demanderModifier(SessionInterface $session, Request $request)
{
    $db = PdoJeux::getPdoJeux();
    return $this->afficherJeux($db, $request->request->get('txtRefJeu'), -1, 'rien');
}
/**
 * @Route("/jeux/validermodifier", name="jeux_validermodifier")
 */
0 references | 0 overrides
public function validerModifier(SessionInterface $session, Request $request)
{
    $db = PdoJeux::getPdoJeux();
    $db->modifierJeu(
        $request->request->get('txtRefJeu'), $request->request->get('txtIdPlateforme'), $request->request->get('txtIdPegi')
        , $request->request->get('txtIdGenre'), $request->request->get('txtIdMarque'), $request->request->get('txtNom'), $request->request->get('txtPrix')
        , $request->request->get('txtDateParution')
    );
    return $this->afficherJeux($db, $request->request->get('txtRefJeu'), -1, 'Modifié');

}
/**
 * @Route("/jeux/supprimer", name="jeux_supprimer")
 */
0 references | 0 overrides
public function supprimer(SessionInterface $session, Request $request)
```

Ces encadrés permettront d'utiliser chaque fonction dans la vue lesJeux.html.twig

```twig
<!-- formulaire pour ajouter un nouveau jeux-->
<tr>
    <form action="{{path('jeux_ajouter')}}" method="post">
        <td>Nouveau</td>
        <td></td>
```