Compte-Rendu Agora Mission 1

Objectifs du TP :

Sommaire :

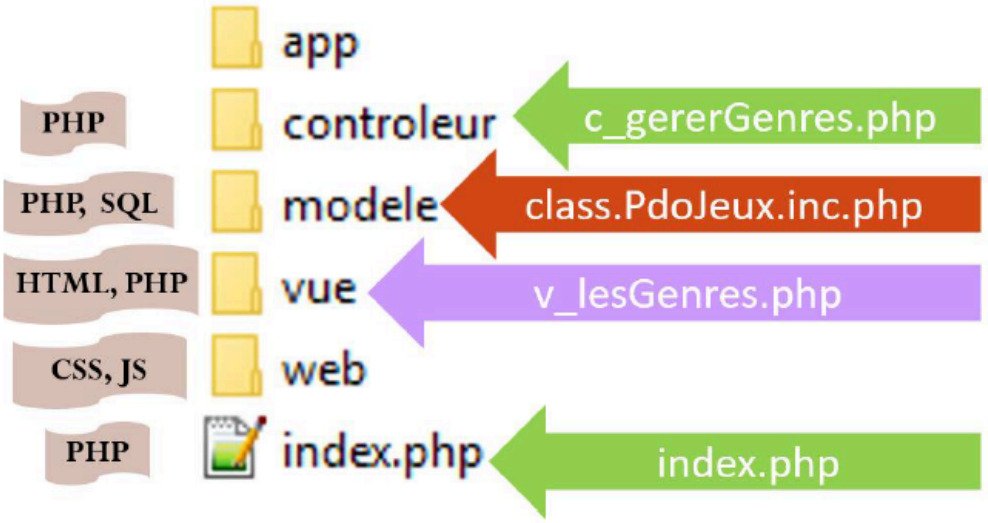
- Mise en architecture MVC
- Compléter le code lié au Genre
- Adapter le code pour les Marques
- Vérification

Première Étape : Mise en architecture MVC

Pour ce faire il faut déjà créer une architecture vide de type MVC comme ici :

app	26/03/2025 14:10	Dossier de fichiers
controleur	26/03/2025 14:12	Dossier de fichiers
modele	26/03/2025 14:10	Dossier de fichiers
vue	26/03/2025 14:11	Dossier de fichiers
web	26/03/2025 14:10	Dossier de fichiers
index.php	26/03/2025 14:12	Fichier source PHP

Ensuite il faut venir remplir nos fichiers comme tel, en décompressant le fichier joint aux bons endroits, comme ici :



Dans le dossier “app”, il faut créer un nouveau fichier qui va avoir comme seul but de définir les constantes afin de se connecter à la BDD :

_config.inc.php

D: > Cours > 2e-semestre > Narbey > TP > AGORA > AgoraBo > app > _config.inc.php

```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);

?>
```

L’architecture étant maintenant terminée, on peut s’atteler à la seconde étape.

Deuxième Étapes : Compléter le code lié au Genre :

Tout d’abord je vais expliquer le code lié au Genre qui est déjà compléter, du moins les différentes méthodes présentes :

-> Tout d’abord, la méthode pour obtenir un tableau contenant tous les genres

```php
//================================================================================
//
//  METHODES POUR LA GESTION DES GENRES
//
//================================================================================

/**
 * Retourne tous les genres sous forme d'un tableau d'objets
 *
 * @return array le tableau d'objets  (Genre)
 */
public function getLesGenres(): array {
    $requete =  'SELECT idGenre as identifiant, libGenre as libelle
                    FROM genre
                    ORDER BY libGenre';
    try {
        $resultat = PdoJeux::$monPdo->query($requete);
        $tbGenres  = $resultat->fetchAll();
        return $tbGenres;
    }
    catch (PDOException $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}
```

Pour expliquer rapidement son fonctionnement on doit déjà définir une requête, c'est une simple requête SELECT qui vient récupérer l'id et le libellé des Genres dans la table genre tout en les triant dans l'ordre alphabétique des libellés.
Une fois la requête défini, on vient essayer de récupérer les résultats (donc sous la forme d'un tableau d'objet) au sein de notre variable "$resultats". Pour ce faire nous allons simplement demander à notre connexion PDO d'exécuter notre "$requete". Pour pouvoir les avoirs sous la forme d'un tableau, il faut maintenant fetchAll() notre résultat dans une autre variables. Puis on retourne simplement notre tableau qui contient l'ensemble des résultats sous la forme d'un tableau. Si a un moment on à un problème, on revoit un message d'erreur.

-> La Deuxième fonction est la première CRUD, celle qui va permettre de pouvoir ajouter de nouveaux genres :

```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
 */
public function ajouterGenre(string $libGenre): int {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("INSERT INTO genre "
                . "(idGenre, libGenre) "
                . "VALUES (0, :unLibGenre) ");
        $requete_prepare->bindParam(':unLibGenre', $libGenre, 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>');
    }
}
```

Pour commencer, on va venir préparer notre requête CRUD, pour venir ajouter dans la table genre à l'aide d'un idGenre (ici sa valeur est de 0 car dans notre table cette colonne est en auto-incrément) et un libellé que l'on va venir compléter juste après.
Une fois notre requête préparée prête, il faut la compléter, du moins juste le libellé, pour la compléter on va utiliser notre variable "$libGenre" qui est donc

notre paramètre d'entrée lors de l'appel de notre fonction. Ensuite il ne reste plus qu'à exécuter la requête et ensuite de retourner à l'aide de notre connexion PDO le dernier ID inséré dans la table. Une nouvelle fois, si à un seul moment il y a une erreur cela nous retournera un message d'erreur.

-> La Troisième fonction va venir nous permettre de modifier un genre :

```php
/**
 * Modifie le libellé du genre donné en paramètre
 *
 * @param int $idGenre : l'identifiant du genre à modifier
 * @param string $libGenre : le libellé modifié
 */
public function modifierGenre(int $idGenre, string $libGenre): void {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("UPDATE genre "
                . "SET libGenre = :unLibGenre "
                . "WHERE genre.idGenre = :unIdGenre");
        $requete_prepare->bindParam(':unIdGenre', $idGenre, PDO::PARAM_INT);
        $requete_prepare->bindParam(':unLibGenre', $libGenre, PDO::PARAM_STR
        $requete_prepare->execute();
    } catch (Exception $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}
```

Cette fonction ressemble beaucoup à celle pour créer un nouveau Genre, donc on vient faire une requête préparer "UPDATE" du genre, avec la valeur à modifier dans notre libellé et ensuite dans le where l'ID de celle que l'on veut. On vient ensuite gérer l'attribution des paramètres ":unIdGenre" et ":unLibGenre" avant de venir exécuter notre requête.
Pour ne pas changer, si à un moment une erreur survient, on revoit toujours le même message d'erreur.

-> Notre dernière fonction est celle qui va venir nous permettre de supprimer un genre :

```php
/**
 * Supprime le genre donné en paramètre
 *
 * @param int $idGenre :l'identifiant du genre à supprimer
 */
public function supprimerGenre(int $idGenre): void {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("DELETE FROM genre "
                . "WHERE genre.idGenre = :unIdGenre");
        $requete_prepare->bindParam(':unIdGenre', $idGenre, PDO::PARAM_INT);
        $requete_prepare->execute();
    } catch (Exception $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}
```

Pour le coup, dans cette fonction, le libellé ne nous intéresse pas, l'id suffit. Donc on va également venir créer une requête préparée à l'aide de notre connexion PDO. Cette requête va venir delete dans la table genre, l'id du genre que l'on souhaite (et donc le genre complet). Pour savoir lequel on veut supprimer, on vient simplement définir notre paramètre ":unIdGenre" avant de pouvoir exécuter notre requête.
Ici aussi, si erreur il y a, message d'erreur il y aura.

Maintenant l'ensemble de nos méthodes sont définis, la connexion avec la BDD aussi, mais le code n'est pas encore tout à fait terminé.

En effet, il va falloir venir modifier notre contrôleur principal (index.php) pour pouvoir gérer le cas des Genres, pour pouvoir faire en sorte que nos méthodes définies plus tôt puissent-être appelées au besoin (si l'utilisateur veut en exécuter une).

```php
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'; // à compléter
        break;
    }
```

Ici, vis-à-vis du code fourni au début du projet, la seule ligne de code qu'il fallait ajouter et le "require" de notre contrôleur secondaire. En effet, notre "index.php" va venir gérer l'ensemble des projets, il va venir dire si l'action va se passer sur telle ou telle table (par exemple). Mais il ne va pas gérer en profondeur ce qui va se passer, pour ce faire il va avoir besoin du contrôleur associé à la table qui doit être modifié. Ici donc les Genres.

Notre contrôleur secondaire qui ne s'occupe que des Genres ressemble à ceci :

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

$idGenreModif = -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 'ajouterNouveauGenre': {
        if (!empty($_POST['txtLibGenre'])) {
            $idGenreNotif = $db->ajouterGenre($_POST['txtLibGenre']);
            // $idGenreNotif est l'idGenre du genre ajouté
            $notification = 'Ajouté';   // sert à afficher l'ajout réalisé dans la vue
        }
      break;
    }

    case 'demanderModifierGenre': {
            $idGenreModif = $_POST['txtIdGenre']; // sert à créer un formulaire de modification pour ce genre
        break;
    }

    case 'validerModifierGenre': {
        $db->modifierGenre($_POST['txtIdGenre'], $_POST['txtLibGenre']);
        $idGenreNotif = $_POST['txtIdGenre']; // $idGenreNotif est l'idGenre du genre modifié
        $notification = 'Modifié';  // sert à afficher la modification réalisée dans la vue
        break;
    }

    case 'supprimerGenre': {
        $idGenre = $_POST['txtIdGenre'];
        $db->supprimerGenre($idGenre); //  à compléter, voir quelle méthode appeler dans le modèle
        break;
    }
}

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

Pour expliquer rapidement ce qu'il fait, il vient vérifier si une action est effectuée ou non (en gros si le contrôleur principal l'appel). Si non, donc la valeur par défaut, il ne va rien faire, si oui alors il va agir différemment selon le

cas. Le cas va être défini au moment de son appel. Pour gérer quoi faire selon les cas on va utiliser un switch et entrer dans le code correspondant à l'action qu'on veut effectuer, donc soit Ajouter / Modifier / Supprimer. En fonction on rentre dans les différents cas des switchs qui vont eux-mêmes modifié les valeurs des ID et des Libellés qu'on a besoin dans nos fonctions précédemment créer.

Je ne vais pas présenter ici le code lié à la vue du genre, car il n'y avait rien à changé par rapport au code fourni à la base, et que contrairement au méthode, il n'y aurait rien de très important à présenter, si ce n'est que des formulaires à coup de "input" et de "button".

Par contre chose important à montrer / modifier, et notre vu de notre menu, où il va falloir venir modifier la redirection (au click) de notre Genres :

```
<li><a href="index.php?uc=gererGenres&action=afficherGenres">Genres</a></li>
```

Le code du Genre est donc maintenant terminé, pour avoir les autres (nous n'allons gérer que le cas des Marques par la suite) il va donc simplement falloir l'adapter et modifier quelques paramètres. Mais rien de bien sorcier !

Troisième Étapes : Adapter le code pour les Marques

À savoir, pour l'ensemble des endroits où il y avait écrit libGenres, ou du moins partout où il y avait "libellé" "lib" ou autre, cela sera remplacé par "nom". Car pour les marques on ne parle pas de libellé mais de nom.
De plus, je ne vais pas réexpliquer chaque méthode car ce sont littéralement les mêmes avec seulement des noms de paramètres différents.

-> Pour obtenir le tableau contenant toutes les marques :

Comme vous pourrez le remarquer, le code fonctionne de la même manière, la seule différence sont le nom de l'id et du "libellé" ainsi que le nom de la table.

```php
    //================================================================================
    //
    //METHODES POUR LA GESTION DES MARQUES
    //
    //================================================================================

    /**
     * Retourne toutes les marques sous forme d'un tableau d'objets
     *
     * @return array le tableau d'objets  (Marque)
     */
    public function getLesMarques(): array {
        $requete =  'SELECT idMarque as identifiant, nomMarque as libelle
                     FROM marque
                     ORDER BY nomMarque';
        try{
            $resultat = PdoJeux::$monPdo->query($requete);
            $tbMarques  = $resultat->fetchAll();
            return $tbMarques;
        }
        catch (PDOException $e){
            die('<div class = "erreur">Erreur dans la requête !<p>'
                .$e->getmessage().'</p></div>');
        }
    }
```

-> Pour l'ajout d'une nouvelle marque :

```php
/**
 * Ajoute une nouvele marque avec le libellé donné en paramètre
 *
 * @param string $libMarque : le libelle de la marque à ajouter
 * @return int l'identifiant de la marque créée
 */
public function ajouterMarque(string $nomMarque): int {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("INSERT INTO marque "
                . "(idMarque, nomMarque) "
                . "VALUES (0, :unNomMarque) ");
        $requete_prepare->bindParam(':unNomMarque', $nomMarque, 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>');
    }
}
```

Même chose ici, même fonctionnement, même code, la seule différence sont les noms des tables / paramètres.

-> Pour la modification d'une marque :

```php
/**
 * Modifie le libellé de la marque donnée en paramètre
 *
 * @param int $idMarque : l'identifiant de la marque à modifier
 * @param string $nomMarque : le libellé modifié
 */
public function modifierMarque(int $idMarque, string $nomMarque): void {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("UPDATE marque "
                . "SET nomMarque = :unNomMarque "
                . "WHERE marque.idMarque = :unIdMarque");
        $requete_prepare->bindParam(':unIdMarque', $idMarque, PDO::PARAM_INT);
        $requete_prepare->bindParam(':unNomMarque', $nomMarque, PDO::PARAM_STR);
        $requete_prepare->execute();
    } catch (Exception $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}
```

Encore une fois, tout est pareil excepté les noms.

-> Enfin pour la suppression d'une marque :

```php
/**
 * Supprime la marque donnée en paramètre
 *
 * @param int $idMarque :l'identifiant de la marque à supprimer
 */
public function supprimerMarque(int $idMarque): void {
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("DELETE FROM marque "
                . "WHERE marque.idMarque = :unIdMarque");
        $requete_prepare->bindParam(':unIdMarque', $idMarque, PDO::PARAM_INT);
        $requete_prepare->execute();
    } catch (Exception $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            .$e->getmessage().'</p></div>');
    }
}
```

Pour ne pas changer, rien n'est changé sauf les différents noms que l'on utilise.

Maintenant que nos méthodes pour les Marques fonctionnent bien, il va falloir venir modifier nos contrôleurs et adapter nos vues.

Pour notre contrôleur principal il suffit de venir ajouter à notre switch un nouveau cas qui va s'occuper de gérer le cas des Marques :

```php
case 'gererMarques' : {
    $menuActif = 'Jeux';    // pour garder le menu correspondant ouvert
    require 'vue/v_menu.php';
    require 'controleur/c_gererMarques.php'; // à compléter
    break;
}
```

Encore une fois, aucune différence avec le cas des genres, outre le fait du nom et du nom de notre contrôleur secondaire.

En parlant de contrôleur secondaire, pour le créer il suffit de simplement dupliquer notre contrôleur qui gère les genres, et modifier tous les "genres" par "marques" :

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

$idMarqueModif = -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 'ajouterNouvelleMarque': {
        if (!empty($_POST['txtNomMarque'])) {
            $idMarqueNotif = $db->ajouterMarque($_POST['txtNomMarque']);
            // $idMarqueNotif est l'idMarque du marque ajouté
            $notification = 'Ajouté';   // sert à afficher l'ajout réalisé dans la vue
        }
      break;
    }

    case 'demanderModifierMarque': {
            $idMarqueModif = $_POST['txtIdMarque']; // sert à créer un formulaire de modification pour cette marque
        break;
    }

    case 'validerModifierMarque': {
        $db->modifierMarque($_POST['txtIdMarque'], $_POST['txtNomMarque']);
        $idMarqueNotif = $_POST['txtIdMarque']; // $idMarqueNotif est l'idMarque de la marque modifiée
        $notification = 'Modifié';  // sert à afficher la modification réalisée dans la vue
        break;
    }

    case 'supprimerMarque': {
        $idMarque = $_POST['txtIdMarque'];
        $db->supprimerMarque($idMarque); //  à compléter, voir quelle méthode appeler dans le modèle
        break;
    }
}

// l' affichage des marques se fait dans tous les cas
$tbMarques  = $db->getLesMarques();
require 'vue/v_lesMarques.php';
```

Mais encore une fois, si on fait vraiment attention, le code reste inchangé (dans le fond), la forme bouge un tout petit peu pour pouvoir s'adapter aux marques.

Pour ce qui est de la vue que je n'ai pas présenté pour les genres. Je ne vais pas la présenter ici non plus car encore une fois rien ne change, si ce n'est le nom qu'on donne aux paramètres. Pour créer le fichier j'ai littéralement dupliquer celui du genre, fait un CTRL + F pour trouver tous les endroits où il avait "genre" et modifié avec "marque". (en adaptant le mot libellé comme précisé au tout début).

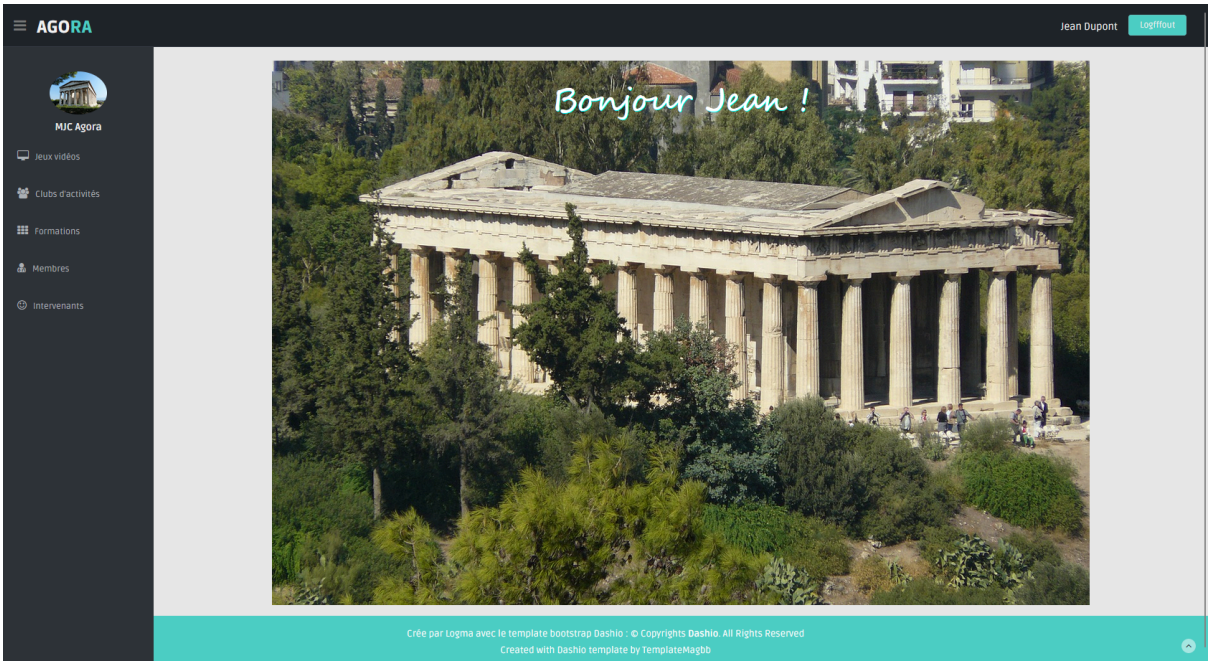
Pour finir cette partie, il fallait également, comme pour les genres, venir modifié notre vue du menu latéral comme ceci :

```
<li><a href="index.php?uc=gererMarques&action=afficherMarques">Marques</a></li>
```

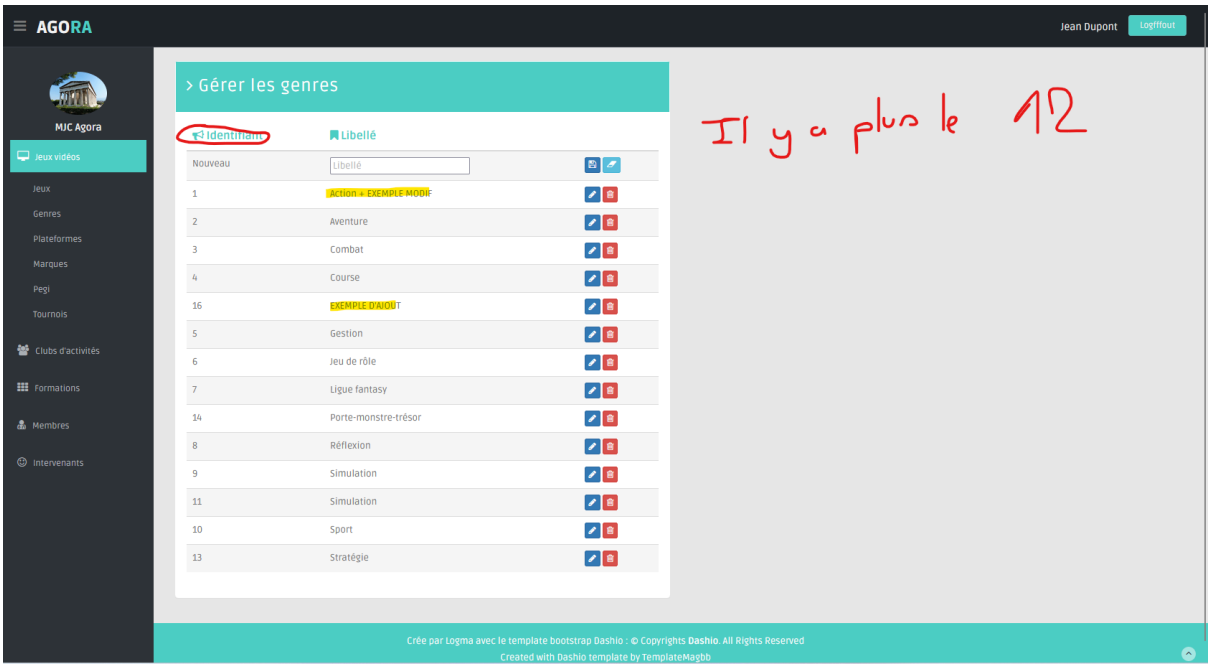
Le code Marque est donc maintenant terminé, et si on applique ceci aux autres colonnes soit Pegi et Plateforme et bien la Mission 1 est quasiment terminée. (Bon pour les Pegi il y a une troisième colonne à gérer, celle des âges, mais pour voir la différence -> cf : compte-rendu de Léandro).

Donc si on applique cela, notre page web ressemble donc à ceci (avec à chaque fois les exemples d'ajout / de modif / de sup) :

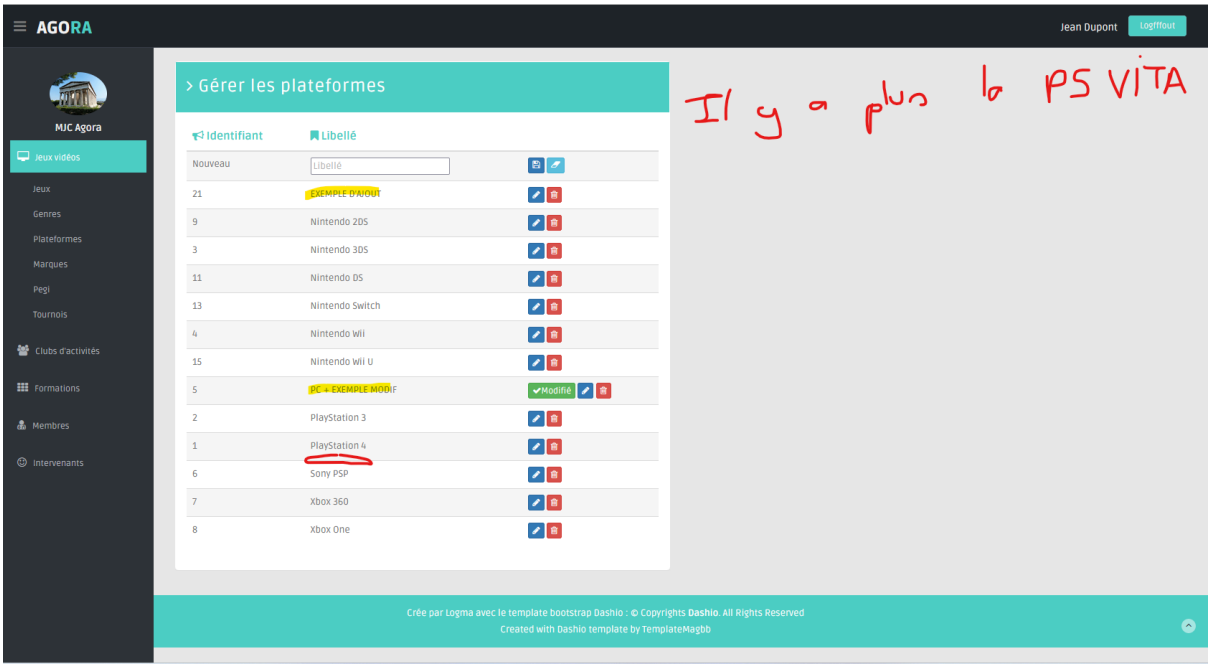
<u>Accueil</u>



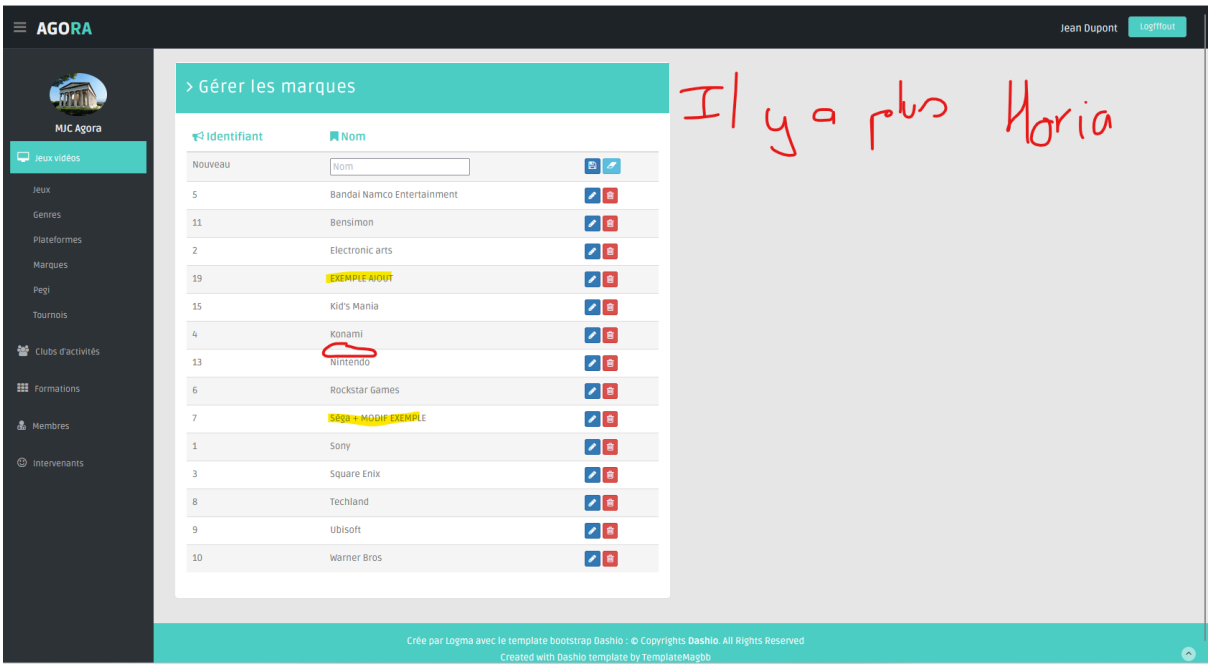
Genres



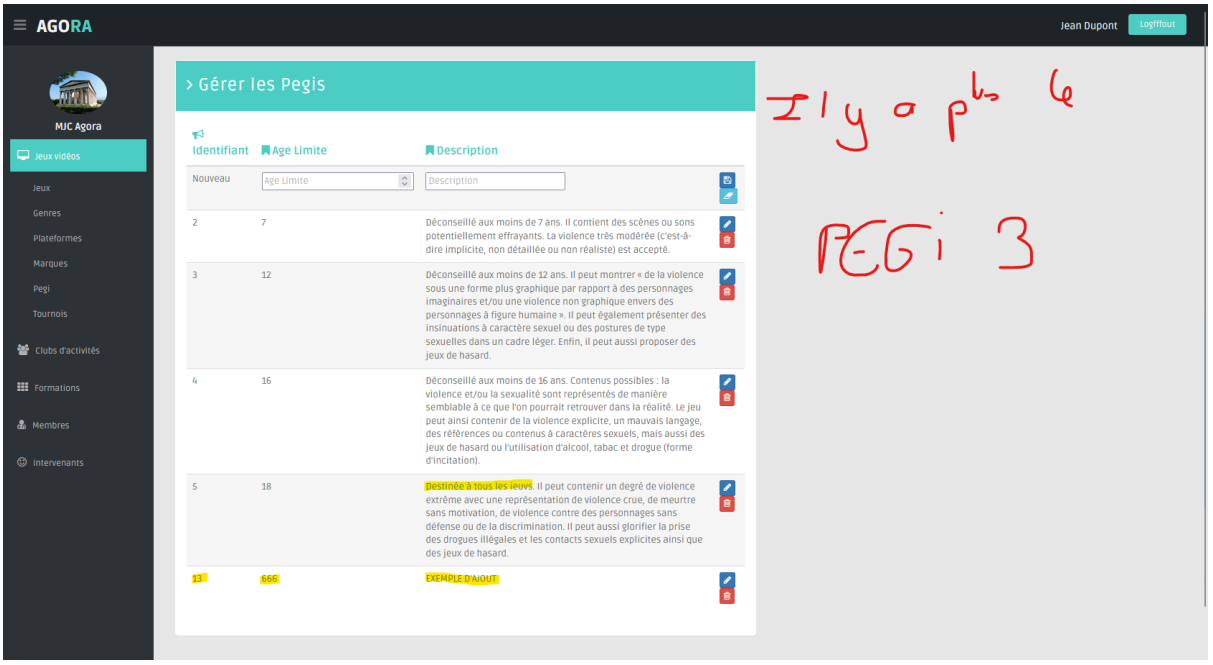
Plateformes



Marques



Pegi



Et voilà. Tout fonctionne bien,
Pour la suite du TP, à savoir la Partie JEUX,
Il va aussi falloir aller regarder chez Léandro car c'est lui qui s'en est chargé. En sachant que j'avais créer moi-même la liste pour gérer les Marques et les Genres, mais il a tout supprimer pour faire sa liste

“Générale”. N’ayant plus de screens à l’appui, je ne peux donc pas la présenter, mais il explique bien dans le sien le fonctionnement de sa liste générale (en vrai elle est simple).

KNAFF Mathéo

PS: Apercu de notre Trello (terminé car C-R date du 26/03)