

PROJET AGORA

Développement du site d'administration des données d'une MJC

Mission 1

Contexte et objectifs

Contexte

Le projet Agora, développement du site d'administration des données d'un MJC, consiste à créer une application de gestion de jeux vidéo pour un MJC. Je suis embauché par Logma une société de services du numérique et doit participer à ce travail qui a déjà été commencé par mes collègues afin de finaliser le projet.

Objectifs

La MJC Agora souhaite revoir entièrement son site web dans un double objectif :

- Améliorer la promotion et la gestion de ses activités
- Réduire le travail administratif fastidieux entraîné par le site actuel qui comporte de nombreuses informations statiques dont la maintenance est chronophage.

Normes de développement

Principe

Une norme de développement est un outil indispensable pour l'efficacité des équipes de développement. Elle a pour but d'uniformiser la communication entre les différents intervenants susceptibles de travailler à un projet informatique utilisant une base de données relationnelle.

Exemples

- Organisation et présentation des fichiers mis en jeu dans un site Web dynamique écrit en PHP.

Les fichiers PHP doivent obligatoirement se terminer par l'extension .php pour une question de sécurité. En procédant ainsi, il n'est pas possible de visualiser le code source des fichiers PHP (qui contiennent peut-être des mots de passe), le serveur web les fait interpréter par PHP.

Tout fichier .php ou page .html doit :

- Etre stocké comme du texte ASCII
- Utiliser le jeu de caractères UTF-8
- Etre formaté Dos

- Les pages Web doivent se conformer à la norme HTML5.

Toute page Web devra donc débuter par la directive <!DOCTYPE html> . Elles seront validées à l'aide du validateur en ligne <http://validator.w3.org>

Architecture de l'application

Présentation MVC

Le motif Modèle-Vue-Contrôleur (en abrégé MVC, de l'anglais Model-View-Controller) répond aux besoins des applications avec une interface Homme-Machine.

C'est un modèle qui sépare les problématiques liées aux différents composants au sein de leur architecture respective.

Il distingue :

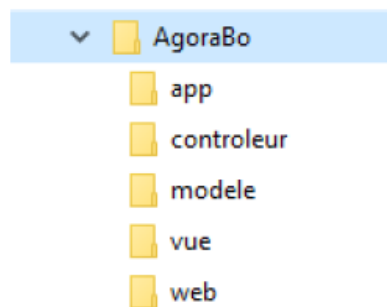
- un modèle (modèle de données),
- une vue (présentation, interface utilisateur)
- un contrôleur (logique de contrôle, gestion des événements, synchronisation)

Séparer le modèle et la vue repose sur une exigence simple, il faut traiter d'un côté le «fond», de l'autre la «forme».

Avantages

- Clarté de l'architecture
- Indépendance des 3 parties de l'application (logique métier, présentation, logique applicative). On peut changer les traitements ou la base de données sans changer la vue
- Maintenance plus souple
- Possibilité de répartir les tâches à des développeurs 'spécialisés'.

Structure de l'application

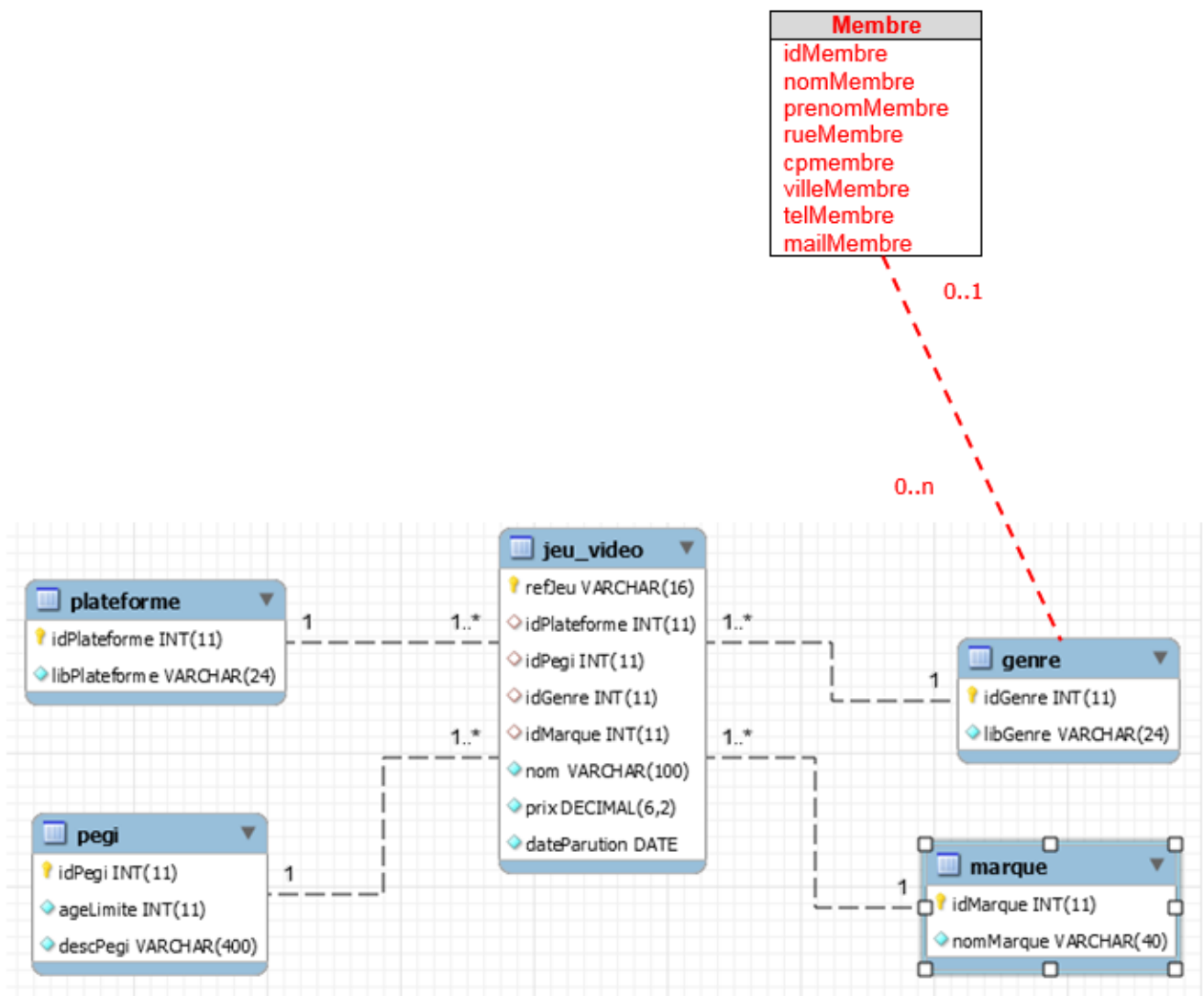


Objectifs des dossiers :

Dossier	Rôle
modele	Accès et manipulation des données
vue	Interface utilisateur (présentation des pages)
controleur	Son rôle est de générer la réponse à la requête HTTP demandée par l'application cliente. Il est la couche qui se charge d'analyser et de traiter la requête. Le contrôleur contient la logique de notre application et utilise le Modèle.
app	Tout ce qui concerne l'application sauf le code source : configuration
web	Les fichiers destinés aux utilisateurs d'un site : images, fichiers CSS et JavaScript.

Base de données

Modèle relationnel des données : schéma



Modèle relationnel des données : forme textuelle

Membre(idMembre, nomMembre, prenomMembre, rueMembre, cpMembre, villeMembre, telMembre, mailMembre)

clé primaire : idMembre

Genre(idGenre, libGenre, idSpecialiste)

clé primaire : idGenre

clé étrangère : idSpecialiste en référence à idMembre de Membre

Pegi(idPegi, ageLimite, descPegi)

clé primaire : idPegi

Plateforme(idPlateforme, libPlateforme)

clé primaire : idPlateforme

Marque(idMarque, nomMarque)

clé primaire : idMarque

Jeu_video(refJeu, nom, prix, dateParution, idGenre, idPegi, idPlateforme, idMarque)

clé primaire : refJeu

clé étrangère : idGenre en référence à idGenre de Genre

clé étrangère : idPegi en référence à idPegi dePegi

clé étrangère : idPlateforme en référence à idPlateforme de Plateforme

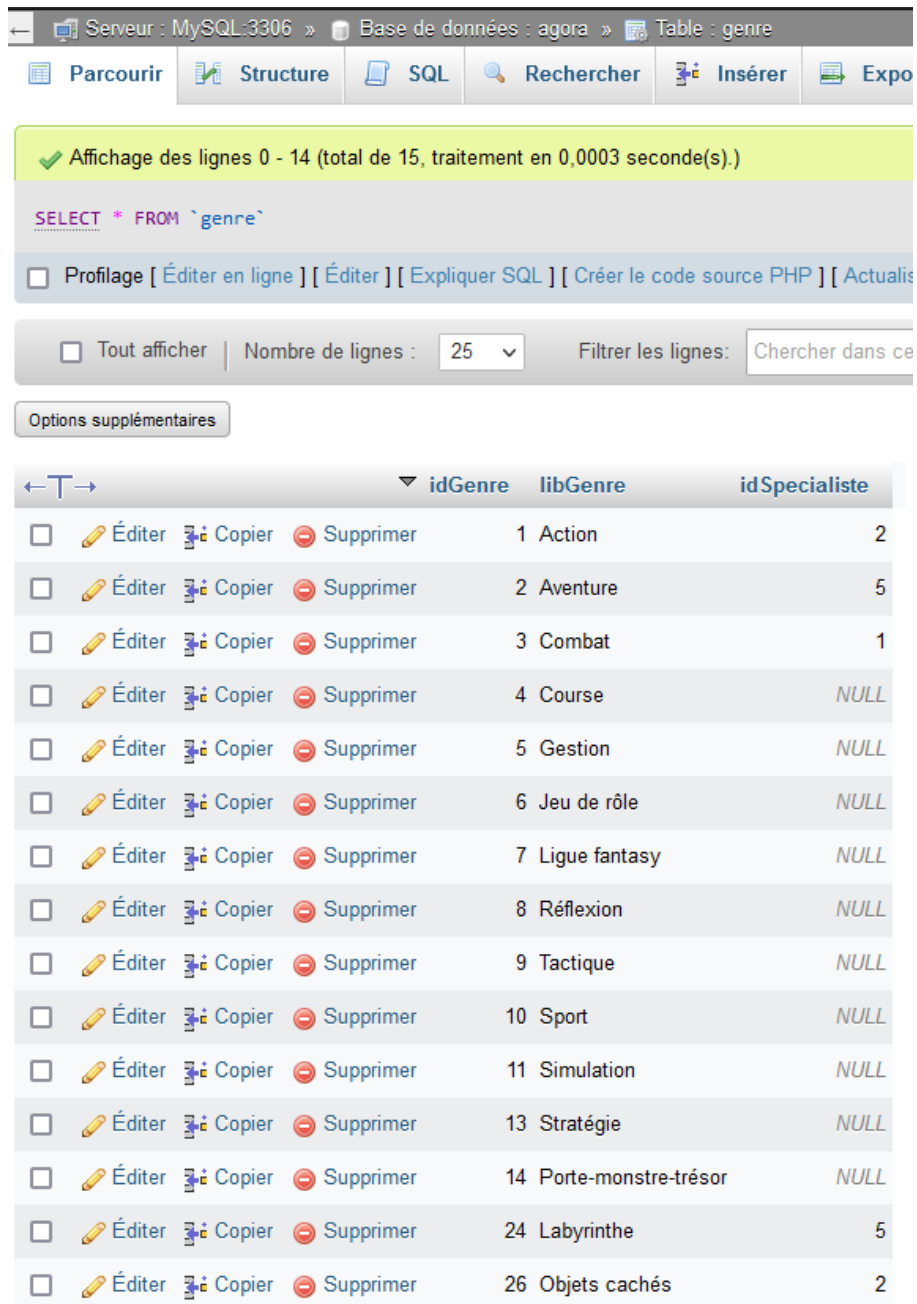
Gestion des genres

La réalisation de la gestion des genres s'est déroulée en classe.
La démarche suit le modèle MVC. Il est également possible de structurer la démarche par fonctionnalité (consultation, ajout, modification) en reprenant le classement MVC pour chaque fonctionnalité.

Tutoriel de mise en application de la gestion des genres

1. Modification de la base de données

1.1 Ajout de la clé étrangère *idSpecialiste* dans la table **Genre**



Seigneur : MySQL:3306 » Base de données : agora » Table : genre

Parcourir Structure SQL Rechercher Insérer Expo

✓ Affichage des lignes 0 - 14 (total de 15, traitement en 0,0003 seconde(s).)

```
SELECT * FROM `genre`
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans ce

Options supplémentaires

				idGenre	libGenre	idSpecialiste
<input type="checkbox"/>	Éditer	Copier	Supprimer	1	Action	2
<input type="checkbox"/>	Éditer	Copier	Supprimer	2	Aventure	5
<input type="checkbox"/>	Éditer	Copier	Supprimer	3	Combat	1
<input type="checkbox"/>	Éditer	Copier	Supprimer	4	Course	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	5	Gestion	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	6	Jeu de rôle	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	7	Ligue fantasy	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	8	Réflexion	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	9	Tactique	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	10	Sport	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	11	Simulation	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	13	Stratégie	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	14	Porte-monstre-trésor	NULL
<input type="checkbox"/>	Éditer	Copier	Supprimer	24	Labyrinthe	5
<input type="checkbox"/>	Éditer	Copier	Supprimer	26	Objets cachés	2

1.2 Modification de quelques genres pour indiquer leur spécialiste

Seveur : MySQL:3306 » Base de données : agora » Table : genre

Parcourir Structure SQL Rechercher Insérer Expo

✓ Affichage des lignes 0 - 14 (total de 15, traitement en 0,0003 seconde(s).)

```
SELECT * FROM `genre`
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans ce

Options supplémentaires

	idGenre	libGenre	idSpecialiste
<input type="checkbox"/> Éditer Copier Supprimer	1	Action	2
<input type="checkbox"/> Éditer Copier Supprimer	2	Aventure	5
<input type="checkbox"/> Éditer Copier Supprimer	3	Combat	1
<input type="checkbox"/> Éditer Copier Supprimer	4	Course	NULL
<input type="checkbox"/> Éditer Copier Supprimer	5	Gestion	NULL
<input type="checkbox"/> Éditer Copier Supprimer	6	Jeu de rôle	NULL
<input type="checkbox"/> Éditer Copier Supprimer	7	Ligue fantasy	NULL
<input type="checkbox"/> Éditer Copier Supprimer	8	Réflexion	NULL
<input type="checkbox"/> Éditer Copier Supprimer	9	Tactique	NULL
<input type="checkbox"/> Éditer Copier Supprimer	10	Sport	NULL
<input type="checkbox"/> Éditer Copier Supprimer	11	Simulation	NULL
<input type="checkbox"/> Éditer Copier Supprimer	13	Stratégie	NULL
<input type="checkbox"/> Éditer Copier Supprimer	14	Porte-monstre-trésor	NULL
<input type="checkbox"/> Éditer Copier Supprimer	24	Labyrinthe	5
<input type="checkbox"/> Éditer Copier Supprimer	26	Objets cachés	2

2 Modifier l'application agoraBo

2.1 Modèle : Dans *class.PdoJeux.inc.php*

2.1.1 Ajout de la méthode **getLesMembres** pour récupérer les membres. Il faut renommer les colonnes identifiant et libellé afin d'utiliser la fonction existante **afficherListe**. Il faut également concaténer prénom et nom du membre spécialiste. La requête SQL peut être testée au préalable (phpMyAdmin ou Mysql Workbench)

```
/**
 * Retourne l'identifiant et le nom complet de toutes les membres sous forme d'un tableau d'objets
 *
 * @return le tableau d'objets
 */
public function getLesMembres()
{
    $requete = 'SELECT idMembre as identifiant, CONCAT(prenomMembre, " ", nomMembre) AS libelle
    FROM membre
    ORDER BY nomMembre';
    try {
        $resultat = PdoJeux::$monPdo->query($requete);
        $tbMembres = $resultat->fetchAll();
        return $tbMembres;
    } catch (PDOException $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            . $e->getMessage() . '</p></div>');
    }
}
```

2.1.2 Adaptation de la méthode **ajouterGenre** pour traiter le spécialiste

```
/**
 * Ajoute un nouveau genre avec le libellé donné en paramètre
 *
 * @param string $libGenre : le libelle du genre à ajouter
 * @param int $idSpecialiste : l'identifiant du spécialiste à ajouter
 * @return int l'identifiant du genre crée
 */
public function ajouterGenre(string $libGenre, int $idSpecialiste): int
{
    try {
        $requete_prepare = PdoJeux::$monPdo->prepare("INSERT INTO genre "
            . "(idGenre, libGenre, idSpecialiste) "
            . "VALUES (0, :unLibGenre, :unIdSpecialiste) ");
        $requete_prepare->bindParam(':unLibGenre', $libGenre, PDO::PARAM_STR);
        $requete_prepare->bindParam(':unIdSpecialiste', $idSpecialiste, PDO::PARAM_INT);
        $requete_prepare->execute();
        // récupérer l'identifiant crée
        return PdoJeux::$monPdo->lastInsertId();
    } catch (Exception $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
            . $e->getMessage() . '</p></div>');
    }
}
```

2.1.3 Adaptation de la méthode *modifierGenre* pour traiter le spécialiste

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

2.1.4 Méthode *getLesGenres*

```
/**
 * 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>');
    }
}
```

2.2 Vues :

2.2.1 Adaptation de la vue `v_lesGenres.php` : cas de l'ajout, de la consultation, de la modification

```
<!-- formulaire pour ajouter un nouveau genre-->
<tr>
<form action="index.php?uc=gererGenres" method="post">
  <td>Nouveau</td>
  <td>
    <input type="text" id="txtLibGenre" name="txtLibGenre" size="24" required minlength="4" maxlength="24" placeholder="Libellé" title="De 4 à 24 caractères" />
  </td>
  <td>
    <button class="btn btn-primary btn-xs" type="submit" name="cmdAction" value="ajouterNouveauGenre" title="Enregistrer nouveau genre"><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>
  </td>
</form>
</tr>
```

```
<!-- formulaire pour modifier et supprimer les genres-->
<form action="index.php?uc=gererGenres" method="post">
<td><?php echo $genre->identifiant; ?><input type="hidden" name="txtIdGenre" value="<?php echo $genre->identifiant; ?>" /></td>
<td><?php
  if ($genre->identifiant != $idGenreModif) {
    echo $genre->libelle;
  }
  </td><td>
    <?php if ($notification != 'rien' && $genre->identifiant == $idGenreNotif) {
      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="demanderModifierGenre" title="Modifier"><i class="fa fa-pencil"></i></button>
    <button class="btn btn-danger btn-xs" type="submit" name="cmdAction" value="supprimerGenre" title="Supprimer" onclick="return confirm('Voulez-vous vraiment supprimer ce genre?');"><i class="fa fa-trash-o"></i></button>
  </td>
<?php
}
else {
  ?><input type="text" id="txtLibGenre" name="txtLibGenre" size="24" required minlength="4" maxlength="24" value="<?php echo $genre->libelle; ?>" />
  </td>
  <td>
    <button class="btn btn-primary btn-xs" type="submit" name="cmdAction" value="validerModifierGenre" 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="annulerModifierGenre" title="Annuler"><i class="fa fa-undo"></i></button>
  </td>
<?php
}
?>
</form>
```

2.3 Contrôleurs :

2.3.1 Adaptation du contrôleur secondaire `c_gererGenres.php` : il faut traiter également le spécialiste pour les cas d'ajout, de modification et de consultation

```
<?php
// si le paramètre action n'est pas positionné alors
// si aucun bouton "action" n'a été envoyé alors par défaut on affiche les genres
// sinon l'action est celle indiquée par le bouton

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
$idGenreNotif = -1; // positionné si 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'], $_POST['lstMembre']);
            // $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'], $_POST['lstMembre']);
        $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
$tbMembres = $db->getLesMembres();
$tbGenres = $db->getLesGenresCompleter();
echo $twig->render('lesGenres.html.twig', array(
    'menuActif' => 'Jeux',
    'tbGenres' => $tbGenres,
    'tbMembres' => $tbMembres,
    'idGenreModif' => $idGenreModif,
    'idGenreNotif' => $idGenreNotif,
    'notification' => $notification
));




?>
```

2.4 Tests des différents cas (CRUD)

Nouveau Kevin Garance 0  

> Gérer les genres



















Identifiant	Libellé	Spécialiste	Nombre jeux	
Nouveau	<input type="text" value="Libellé"/>	Elodie Celon	0	 
1	Action	Elodie Celon	2	 
33	ajout	Elodie Celon	0	 Ajouté  

29 Didier Dubois   

> Gérer les genres

































Identifiant	Libellé	Spécialiste	Nombre jeux	
Nouveau	<input type="text" value="Libellé"/>	Elodie Celon	0	 
1	Action	Elodie Celon	2	 
33	ajout modification	Didier Dubois	0	 Modifié  

> Gérer les genres

Identifiant	Libellé	Spécialiste	Nombre jeux	
Nouveau	<input type="text" value="Libellé"/>	Elodie Celon	0	 
1	Action	Elodie Celon	2	 
33	ajout modification	Didier Dubois	0	 
2	Aventure		5	 
3	Combat	Didier Dubois	0	 
4	Course			 
5	Gestion			 
6	Jeu de rôle			 
24	Labyrinthe		0	 

localhost
Voulez-vous vraiment supprimer ce genre ?

> Gérer les genres

Identifiant	Libellé	Spécialiste	Nombre jeux	
Nouveau	<input type="text" value="Libellé"/>	<input type="text" value="Elodie Celon"/>	0	 
1	Action	Elodie Celon	2	 
2	Aventure		5	 
3	Combat	Didier Dubois	0	 
4	Course		0	 
5	Gestion		0	 
6	Jeu de rôle		1	 
24	Labyrinthe		0	 
7	Ligue fantasy		0	 
26	Objets cachés	Elodie Celon	0	 
14	Porte-monstre-trésor		0	 
8	Réflexion		0	 
11	Simulation		0	 
10	Sport		2	 
13	Stratégie		1	 
9	Tactique		0	 



Gestion des marques, pegis, plateformes

En ce qui concerne la gestion des marques, pegis et plateformes leur réalisation s'est faite en autonomie et est similaire à la gestion des genres.

Gestion des jeux

Les jeux ont pour attributs un genre, une marque, une plateforme et un pegi. L'ajout d'un jeu nécessite la sélection de ces attributs via des listes déroulantes.

> Gérer les jeux

Référence	Libellé	Date de parution	Prix	Genre	Plateforme	Marque	Pegi	
<input type="text" value="Référence"/>	<input type="text" value="Libellé"/>	<input type="text" value="jj / mm / aaaa"/>	<input type="text" value="Prix"/>	<input type="text" value="Action"/>	<input type="text" value="PlayStation 4"/>	<input type="text" value="Sony"/>	<input type="text" value="3"/>	 

Aperçu

```

<td>
  <!-- <input type="select" id="txtLibGenre" name="txtLibGenre" size="24" required minlength="4" maxlength="24"
  placeholder="Genre" title="De 4 à 24 caractères" /> -->
  <?php
  afficherListe($tbGenres, 'lstGenre', 1, 1);
  ?>
</td>
<td>
  <!-- <input type="select" id="txtLibPlateforme" name="txtLibPlateforme" size="24" required minlength="4" maxlength="24"
  placeholder="Plateforme" title="De 4 à 24 caractères" /> -->
  <?php
  afficherListe($tbPlateformes, 'lstPlateformes', 1, 1);
  ?>
</td>
<td>
  <!-- <input type="select" id="txtNomMarque" name="txtNomMarque" size="24" required minlength="4" maxlength="24"
  placeholder="Marque" title="De 4 à 24 caractères" /> -->
  <?php
  afficherListe($tbMarques, 'lstMarques', 1, 1);
  ?>
</td>
<td>
  <!-- <input type="select" id="txtAgeLimite" name="txtAgeLimite" min="3" max="99" required placeholder="Age" /> -->
  <?php
  afficherListePegi($tbPegi, 'lstPegi', 1, 1);
  ?>
</td>

```

Vue

```

case 'ajouterNouveauJeu': {
    if (!empty($_POST['txtRefJeu'])) {
        $unJeu = (object) [
            'reference' => $_POST['txtRefJeu'],
            'nom' => $_POST['txtNomJeu'],
            'dateParution' => $_POST['txtDateParution'],
            'prix' => $_POST['txtPrix'],
            'genre' => $_POST['lstGenre'],
            'plateforme' => $_POST['lstPlateformes'],
            'marque' => $_POST['lstMarques'],
            'pegi' => $_POST['lstPegi']
        ];
        $idJeuNotif = $db->ajouterJeu($unJeu);
        $notification = 'Ajouté'; // sert à afficher l'ajout réalisé dans la vue
    }
    break;
}

case 'demanderModifierJeu': {

    $idJeuModif = $_POST['txtRefJeu'];

    break;
}

case 'validerModifierJeu': {
    $unJeu = (object) [
        'reference' => $_POST['txtRefJeu'],
        'nom' => $_POST['txtNomJeu'],
        'dateParution' => $_POST['txtDateParution'],
        'prix' => $_POST['txtPrix'],
        'genre' => $_POST['lstGenre'],
        'plateforme' => $_POST['lstPlateformes'],
        'marque' => $_POST['lstMarques'],
        'pegi' => $_POST['lstPegi']
    ];
    $db->modifierJeu($unJeu);
    $idJeuNotif = $_POST['txtRefJeu'];
    $notification = 'Modifié';
    break;
}

```

Contrôleur

```

// l' affichage des jeux se fait dans tous les cas
$tbJeux = $db->getLesJeux();
$tbGenres = $db->getLesGenres();
$tbPlateformes = $db->getLesPlateformes();
$tbMarques = $db->getLesMarques();
$tbPegi = $db->getLesPegi();
echo $twig->render('lesJeux.html.twig', array(
    'menuActif' => 'Jeux',
    'tbJeux' => $tbJeux,
    'tbGenres' => $tbGenres,
    'tbPlateformes' => $tbPlateformes,
    'tbMarques' => $tbMarques,
    'tbPegi' => $tbPegi,
    'idJeuModif' => $idJeuModif,
    'idJeuNotif' => $idJeuNotif,
    'notification' => $notification
))

```

Récupération des listes par le controleur pour la vue

L'ajout, la modification et la suppression des jeux se font de manière similaire à ceux des genres.