Synthèse à faire sur la mission 1 & 2 de AgoraBo

**Contexte général**
La MJC Agora ne dispose pas d'un service informatique. Elle a fait appel à la société de services du numérique (ESN) Logma dans laquelle vous êtes embauché. Vous participez au développement du site d'administration des données de la MJC.

**But**
Travailler sur l'organisation du code, ainsi que certains menus. Puis créer un système d'authentification au site.

**Outil**
Utilisation de Visual Studio Code.

**Qui êtes-vous ? Votre mission ?**
Employé de la société de services du numériques (ESN), je suis chargé de travailler au développement du site Agora. Je travaille plus précisément sur l'organisation du code, la création de menus ainsi que la mise en place d'une authentification.

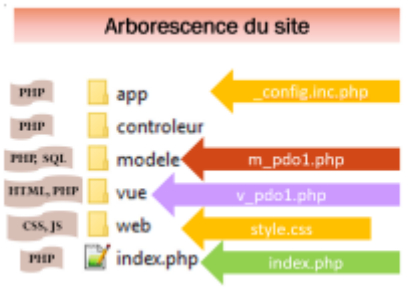
**Mission 1 :**
La mission 1 a pour objectif d'**organiser le code** afin de le rendre plus structuré et de s'y retrouver plus facilement. Puis par la suite **créer des menus** dans ce modèle.

Un modèle de conception est une solution à un problème récurrent dans la conception d'applications orientées objet. Le modèle de conception est au niveau de la conception donc il est indépendant des technologies et des langages de programmation.

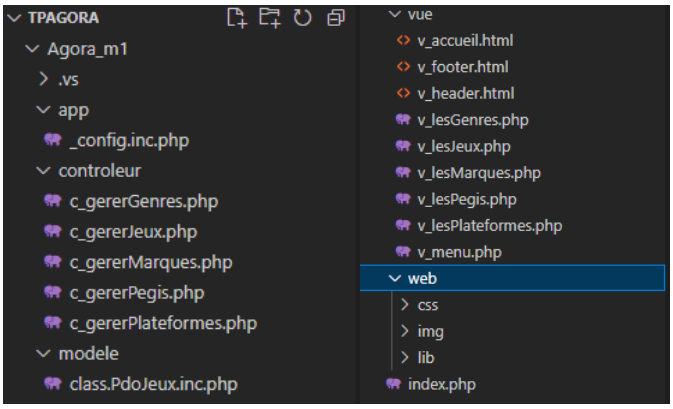
Utilisation du modèle de conception MVC (Model-View-Controller).

Principe du modèle de conception MVC : Diviser pour mieux régner
Séparer chaque partie du code : les modèles, les vues, les contrôleurs.
Avantages du MVC : indépendance des 3 parties de l'application / changement de certaines parties du code plus facile / possibilité de répartir les tâches à des développeurs « spécialisés »
Inconvénients du MVC : temps d'apprentissage important / risque de cloisonnement des développeurs
Conclusion : il faut privilégier l'utilisation du MVC dans le cadre de développement de grandes applications.

Exemple de MVC :

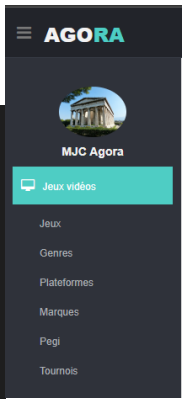



On regroupe tous les fichiers du contrôleur ensemble.  Pareil pour la vue, et la page modèle dans le modèle. Mise en place du modèle MVC avec Agora :




Création des menus dans l'index du MVC :

```php
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;
}
```




Création de la vue d'un des menus, exemple pour les genres :

```php
<!-- page start-->
<div class="col-sm-6">
    <section class="panel">
        <div class="chat-room-head">
            <h3><i class="fa fa-angle-right"></i> Gérer les genres</h3>
        </div>
        <div class="panel-body">
            <table class="table table-striped table-advance table-hover">
            <thead>
              <tr class="tableau-entete">
                <th><i class="fa fa-bullhorn"></i> Identifiant</th>
                <th><i class="fa fa-bookmark"></i> Libellé</th>
                <th></th>
              </tr>
            </thead>
            <tbody>
            <!-- 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>
```


Création du contrôleur, exemple pour les genres :

```php
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'];
        $db-> supprimerGenre($_POST['txtIdGenre']);
        break;
    }
}

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

Codage du modèle afin de lier la base de donnée au site, premièrement pour prendre la table en question. Puis lier cette table au bouton ajouter, modifier, supprimer. Exemple pour les genres :

```php
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>');
    }
}
```

```php
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>');
    }
}
```

```php
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>');
    }
}
```

```php
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>');
    }
}
```

**Mission 2 :**

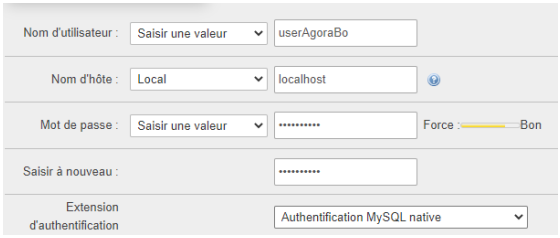
La mission 2 a pour objectif de mettre en place un **système d'authentification** afin de s'assurer des utilisateurs qui utilisent le site.
L'authentification a pour but de s'assurer de la légitimité et de savoir si la personne qui s'authentifie est autorisé à accéder aux ressources. Ce système permet de contrôler l'accès.
L'authentification mot de passe présente des risques.
- Mot de passe simple, tel que azerty, nom, prénom, date de naissance etc.. sont très facilement piratable.
- Une attaque par force brute peut aussi être effectué. Une **attaque par force brute** est une technique relativement courante, mais qui nécessite beaucoup de temps d'exécution. Avec un script le pirate essaye toutes les combinaisons de mots de passe possibles. L'ordre de test peut être optimisé afin d'obtenir de meilleurs rendements et faire appel à différentes techniques, par exemple l'attaque par dictionnaire. Dans ce cas, le hacker se base sur d'énormes dictionnaires contenant des millions voire des milliards de mots de passe régulièrement utilisés.

- Une attaque sur base de données est possible.
- Le phising est une méthode permettant de récupérer les logins d'un utilisateur via un faux lien qui pourrait prétendre à un lien officiel. Une méthode de piratage populaire de nos jours.

Dans cette mission, pour protéger les mots de passe des utilisateurs, nous utiliserons l'algorithme de hachage SHA512 qui produit un hash de 512 bits soit 128 caractères hexadécimaux. Une méthode qui consiste à stocker une empreinte du mot de passe. Ce hachage ne permet pas de retrouver la valeur initiale. (Hachage utilisé dans les systèmes de blockchain)

Les identifiants et mots de passe des utilisateurs d'Agora seront stockés dans la base de données Agora. Leurs fonctionnalités d'accessibilités seront tous les mêmes.

Création d'un utilisateur dans la base de données :




Attribution des privilèges :

```
GRANT SELECT, INSERT, UPDATE, DELETE ON `jeux`.* TO 'userAgoraBo'@'localhost'; ALTER USER 'userAgoraBo'@'localhost' ;
```

Implantation de l'utilisateur dans le code :

```
// 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', 'Azerty123!');                 // mot de passe
define('DSN','mysql:dbname='.DB_DATABASE.';host='.DB_SERVER);
```

Table des membres dans la base de données :

| lMembre | nomMembre | prenomMembre | mailMembre | telMembre | rueMembre | cpMembre | villeMembre | loginMembre | mdpMembre |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Dubois | Didier | dubois.didier@gmail.com | 0685451236 | 48 rue des acacias | 57000 | Metz | dubois | 7e3c5f890206b7cad6a01 |
| 2 | Celon | Elodie | elodie35@gmail.com | 0689451235 | 18 rue des Tilleuls | 57000 | Metz | celon | f332c43c4aed2f69497df8 |
| 3 | Garance | Kevin | garance@gmail.com | 0678451236 | 5 avenue Victor hugo | 57000 | Metz | garance | 2ac84572232ba491a8963 |

Gestion des membres dans le code :

```
//====================================================================
//
// METHODES POUR LA GESTION DES MEMBRES
//
//====================================================================
/**
* Retourne l'identifiant, le nom et le prénom de l'utilisateur correspondant au compte et mdp
*
* @param string $compte le compte de l'utilisateur
* @param string $mdp le mot de passe de l'utilisateur
* @return ?object l'objet ou null si ce membre n'existe pas
*/
public function getUnMembre(string $loginMembre, 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', $loginMembre, PDO::PARAM_STR);
    $requete_prepare->bindParam(':unMdpMembre', $mdpMembre, PDO::PARAM_STR);
    // exécuter la requête
    $requete_prepare->execute();
    // récupérer l'objet
    if ($utilisateur = $requete_prepare->fetch()) {
        $requete = 'SELECT mdpMembre from membre where mdpMembre = unMdpMembre';
    // vérifier le mot de passe
    // 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)
    // random_bytes ( int $length ) : string (fonction PHP pour générer des sels)
    }
    catch (PDOException $e) {
        die('<div class = "erreur">Erreur dans la requête !<p>'
        .$e->getmessage().'</p></div>');
    }
    }
```

Création de la vue de connexion dans le code :

```
********************************************************************************
********************************************************************** -->
<div id="login-page">
 <div class="container connexion">
 <form class="form-login" method="post" action="index.php?uc=connexion">
 <h2 class="form-login-heading">Identification utilisateur</h2>
 <div class="login-wrap">
 <?php
 if (isset($erreur)) {
 echo '<div class = "erreurCnx"><p>'.$erreur.'</p></div>';
 }
 ?>

 <input type="text" class="form-control" name="txtLogin" id="txtLogin"
placeholder="Login" required autofocus />
 <br>
 <input type="password" class="form-control" name="txtMdp" id="txtMdp"
placeholder="Mot de passe" required />
 <div class="pull-right login-social-link">
 <!--La version 5 du langage HTML a ajouté un nouveau type d'attribut, qui
n'est pas interprété par le navigateur.
Ce sont tous les attributs dont le nom commence par les lettres data.
L'attribut data-toggle contient le type d'événement qui va être lié à un
bouton
 data-toggle=modal /* Bouton qui ouvre une fenêtre modale -->
 <a data-toggle="modal" href="v_connexion.php#myModal"> Mot de passe oublié
?</a>
 </div>
 <button class="btn btn-theme btn-block" type="submit" name="cmdAction"
value="validerConnexion" title="Se connecter" onclick="document.getElementById('hdMdp').value =
hex_sha512(document.getElementById('txtMdp').value);document.getElementById('txtMdp').value = '';">

 <i class="fa fa-lock"></i> Se connecter</button>
  <!-- champ caché pour le mot de passe haché -->
  <input type="hidden" name="hdMdp" id="hdMdp" />

 <hr>
 </div>

 <!-- la fenêtre modale -->
 <div aria-hidden="true" aria-labelledby="myModalLabel" role="dialog" tabindex="-1"
```

Utilisation de la méthode de hachage SHA512 dans le code :

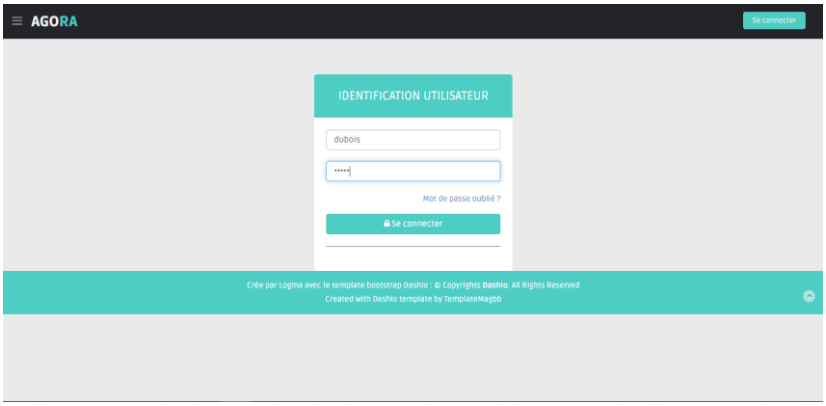
```
<!-- pour la connexion -->
<script src="web/libAgora/sha512.js"></script>
```

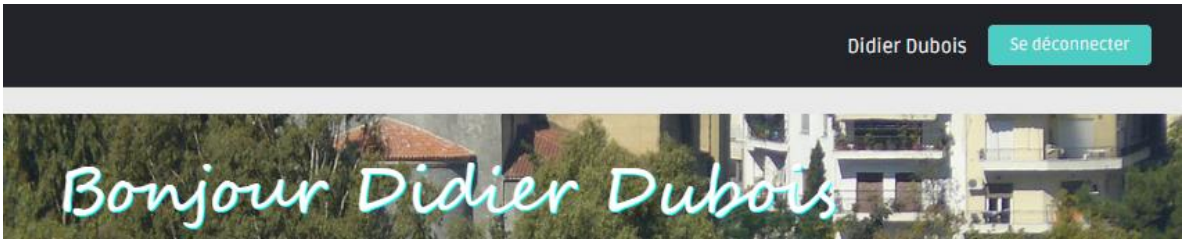
Gestion de connexion et déconnexion :

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

switch ($action) {

    case 'demanderConnexion': {
        require
        break;
    }

 case 'validerConnexion': {
    // vérifier si l'utilisateur existe avec ce mot de passe
    if $utilisateur = {

        // si l'utilisateur n'existe pas
        // positionner le message d'erreur $erreur
    } else {
        echo $erreur;
    }

// inclure la vue correspondant au formulaire d'authentification

    } else {
        // créer trois variables de session pour id utilisateur, nom et prénom
        $_SESSION['idUtilisateur'] = $utilisateur->idPersonne;
        $_SESSION['nom'] = $utilisateur->nom;
        $_SESSION['prenom'] = $utilisateur->prenom;

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

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

Résultat de la page de connexion :



**Mission 3 :**

La mission 3 consiste à utiliser le moteur de Template TWIG pour le langage de programmation PHP. Twig est utilisé par le Framework Symfony.
Un moteur de template est un outil de modèle structurel qui simplifie la syntaxe pour assurer une bonne maintenabilité d'une application web. Il permet de dissocier la partie présentation (HTML) de la partie programmation. Les fichiers templates sont des fichiers qui comportent la mise en page des pages, (et aussi des e-mails, flux RSS, …).

**But** :
Le but de cette mission est d'améliorer la lisibilité et la maintenabilité du code des vues en utilisant twig.

Le moteur de Template Twig permet de simplifier le code php, de bien distinguer le code html du php, et rend les pages de code plus visible.
Sa syntaxe est plus concise et plus claire :
En PHP on affiche avec < ? php echo $uneVariable ; ?>
Avec twig on écrit {{ uneVariable }}

Exemple de php qui passe en twig :
```
<?php
foreach ($tbGenres as $genre) {
?>
```
```
{% for key, genre in tbGenres %}
```

Il faut aussi ajouter les blocks d'instructions en twig afin d'organiser le code :
```
{% block central %}
```
```
{% endblock %}
```

Les instructions conditionnelles if, else se font de cette manière :
```
{% if genre.identifiant != idGenreModif %}
    {{ genre.libelle }}</td>
```
```
{% else %}
```

Une balise endif est écrite afin de marquer la fin du if :
```
{% endif %}
```

En php, pour récuper une colonne d'une table, on écrit $table->colonne, mais en twig ce sera juste un point comme montré ci-dessus.

Toutes les parties en php des vues, seront transformées en twig, donc les genres, les plateformes, les jeux, les pegis, les marques verront leurs code passé en twig à la place du php.
Donc effectuer les modifications comme ci-dessus. Et reproduire ces modifications pour chaque vue indiquée.

Ces vues, vont faire un require du layout. Le layout est le template père. Le template père est celui qui définit les templates qui seront réutilisés dans les fichiers.
Le template père comprend 4 grandes parties :
- l'entête qui était dans v_header.php
- un bloc menu qui inclut un template menu uniquement si l'utilisateur est connecté
- un bloc central que les templates fils rempliront
- le pied de page qui était avant dans v_footer.html

Quasiment toutes les vues vont être transformer en twig (accueil, connexion, menu)
Les changements sont assez minimes, il suffit de mettre les blocks d'instructions, ainsi que modifier les if, else etc.

En ce qui concerne l'index, le require du menu n'est plus necessaire car il se fera dans le controleur de chaque page (marque, genre, etc..)

Dans le controleur, il faut rajouter ceci :

```
echo $twig->render('lesMarques.html.twig', array(
    'menuActif' => 'Jeux',
    'tbMarques' => $tbMarques,
    'idMarqueMotif' => $idMarqueModif,
    'idMarqueNotif' => $idMarqueNotif,
    'notification' => $notification
));
```

IdMarqueNotif aura été ajouté au préalable plus haut dans le fichier.
Le require du menu est fait ici, au niveau du menuActif.

Toutes les modifications sont assez simples dans l'ensemble, il faut juste adapter le code php en twig. La vue pour les genres a été la plus compliqué à mettre en place car la colonne idSpecialiste ne se synchronisait pas lorsqu'un nouveau genre était ajouté. Il a fallut faire la clé étrangère entre le genre et les membres, puis rajouter dans le modèle, un select pour l'idSpecialiste et rajouter le paramètre. Rajouter ce paramètre dans le controleur des genres.

```
 $requete = 'SELECT G.idGenre as identifiant, G.libGenre as libelle, G.idSpecialiste AS
idSpecialiste, CONCAT(P.prenomMembre, " ", P.nomMembre) AS nomSpecialiste,
 (SELECT COUNT(refJeu) FROM jeu_video AS J WHERE J.idGenre = G.idGenre) AS nbJeux
 FROM genre AS G
 LEFT OUTER JOIN membre AS P ON G.idSpecialiste = P.idMembre
 ORDER BY G.libGenre';
$requete_prepare->bindParam(':lstMembre', $idSpecialiste, PDO::PARAM_INT);
```

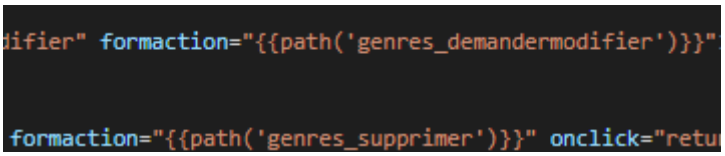
**Mission 4 :**

La mission 4 est de transféré notre projet Agora sur le framework symfony.
Symfony est un ensemble de composants PHP, un framework d'application web. Ce framework permet d'être plus efficace, et rapide car il permet aux développeurs de gagner du temps en réutilisant des modules génériques. Il dispose d'énorméments de spécificités afin d'alléger le travail des développeurs. Il permet aussi de maintenir la stabilité des applications à moindre coût. Ce framework est très recherché dans le monde du travail ce qui est bénéfique pour ceux le maitrisant.

Dans cette mission, nous avons installer composer et sympfony. Grâce à symfony nous avons créer un dossier AgoraBoS qui servira pour la mission 4. Pour utiliser symfony il faut lancer le serveur à partir de l'invite de commandes.

En codage, on va reprendre les fichiers de la mission 3, le modèle et les vues. On fera des changements minimes dans le modèle.

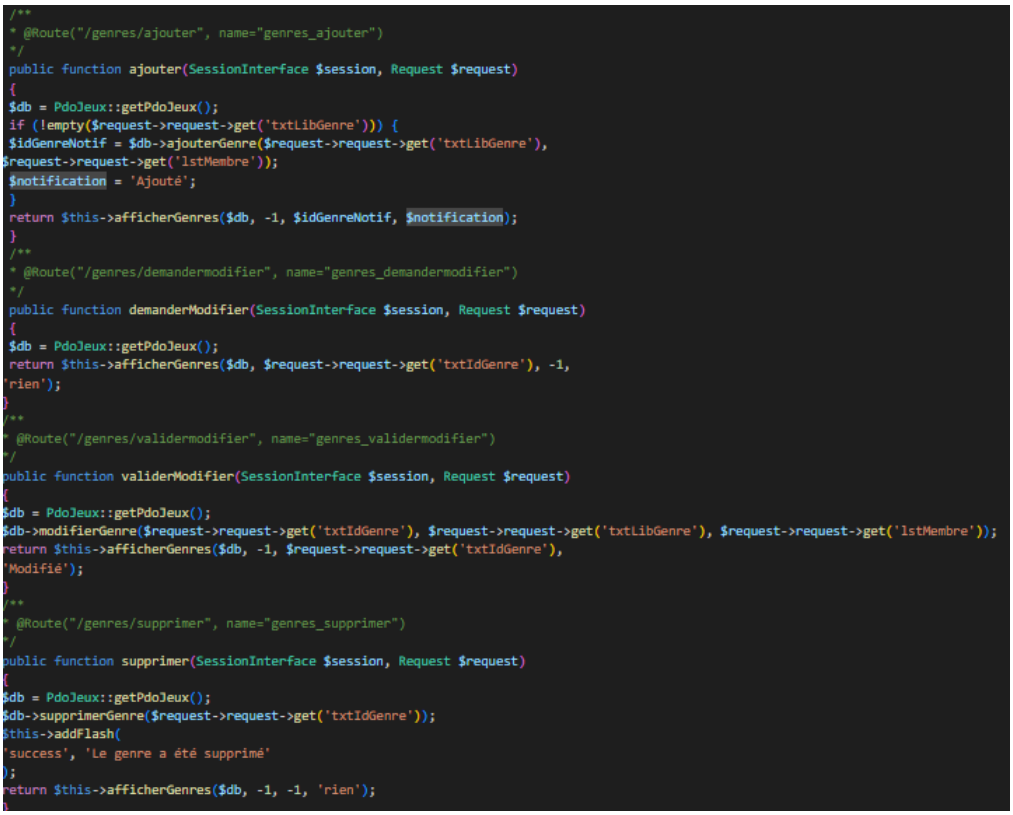
Pour les vues, les changements seront toujours les mêmes pour toutes.
`{{ include('message.html.twig') }}` cette include sera dans toutes les vues des menus.
`{{path('genres_ajouter')}}` pour ceci, on adapte en fonction du menu en question



ce sera pareil pour ces deux formaction

Quant aux controleurs, ils changent complètement.

```php
<?php
// src/Controller/GenresController.php
namespace App\Controller;
use Symfony\Component\HttpFoundation\Response;
require_once 'modele/class.PdoJeux.inc.php';
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Bundle\FrameworkBundle\Controller\RedirectController;
use PdoJeux;
class GenresController extends AbstractController
{
 /**
 * fonction pour afficher la liste des genres
 * @param $db
 * @param $idGenreModif positionné si demande de modification
 * @param $idGenreNotif positionné si mise à jour dans la vue
 * @param $notification pour notifier la mise à jour dans la vue
 */
 private function afficherGenres(PdoJeux $db, int $idGenreModif, int $idGenreNotif,
string $notification ) {
 $tbMembres = $db->getLesMembres();
 $tbGenres = $db->getLesGenresComplet();
 return $this->render('lesGenres.html.twig', array(
 'menuActif' => 'Jeux',
 'tbGenres' => $tbGenres,
 'tbMembres' => $tbMembres,
 'idGenreModif' => $idGenreModif,
 'idGenreNotif' => $idGenreNotif,
 'notification' => $notification
 ));
 }
 /**
 * @Route("/genres", name="genres_afficher")
 */
 public function index(SessionInterface $session)
 {
 if ($session->has('idUtilisateur')) {
 $db = PdoJeux::getPdoJeux();
 return $this->afficherGenres($db, -1, -1, 'rien');
 } else {
 return $this->render('connexion.html.twig');
 }
```



```php
/**
* @Route("/genres/ajouter", name="genres_ajouter")
*/
public function ajouter(SessionInterface $session, Request $request)
{
$db = PdoJeux::getPdoJeux();
if (!empty($request->request->get('txtLibGenre'))) {
$idGenreNotif = $db->ajouterGenre($request->request->get('txtLibGenre'),
$request->request->get('lstMembre'));
$notification = 'Ajouté';
}
return $this->afficherGenres($db, -1, $idGenreNotif, $notification);
}
/**
* @Route("/genres/demandermodifier", name="genres_demandermodifier")
*/
public function demanderModifier(SessionInterface $session, Request $request)
{
$db = PdoJeux::getPdoJeux();
return $this->afficherGenres($db, $request->request->get('txtIdGenre'), -1,
'rien');
}
/**
* @Route("/genres/validermodifier", name="genres_validermodifier")
*/
public function validerModifier(SessionInterface $session, Request $request)
{
$db = PdoJeux::getPdoJeux();
$db->modifierGenre($request->request->get('txtIdGenre'), $request->request->get('txtLibGenre'), $request->request->get('lstMembre'));
return $this->afficherGenres($db, -1, $request->request->get('txtIdGenre'),
'Modifié');
}
/**
* @Route("/genres/supprimer", name="genres_supprimer")
*/
public function supprimer(SessionInterface $session, Request $request)
{
$db = PdoJeux::getPdoJeux();
$db->supprimerGenre($request->request->get('txtIdGenre'));
$this->addFlash(
'success', 'Le genre a été supprimé'
);
return $this->afficherGenres($db, -1, -1, 'rien');
}
```

On reprendra la même base du controleur pour les autres menus.

`<a href="{{path('jeux_afficher')}}` on modfie le lien des menus en fonction de ce qu'on a écrit dans les vues, c'est-à-dire le path qui a été modifier plus haut. On fait cette modification pour tous les menus.

Difficultées rencontrées au niveau des modifications, suppression lors d'un nouvel ajout.