AgoraBo

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 entrainé 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 à pour but d'uniformiser la communication entre les différents intervenants susceptibles de travailler à un projet informatique utilisant une base de données relationelle.

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 :

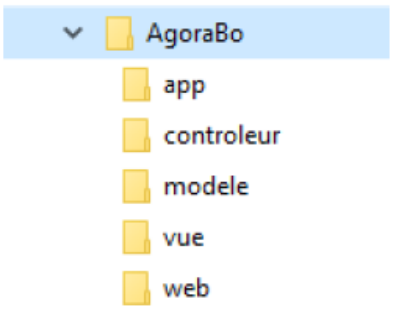
- un modèle (modèle de données),
- une vue (présentation, interface utilisateur)
- un contrôleur (logique de contrôle, gestion des événements, synchronisation)

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

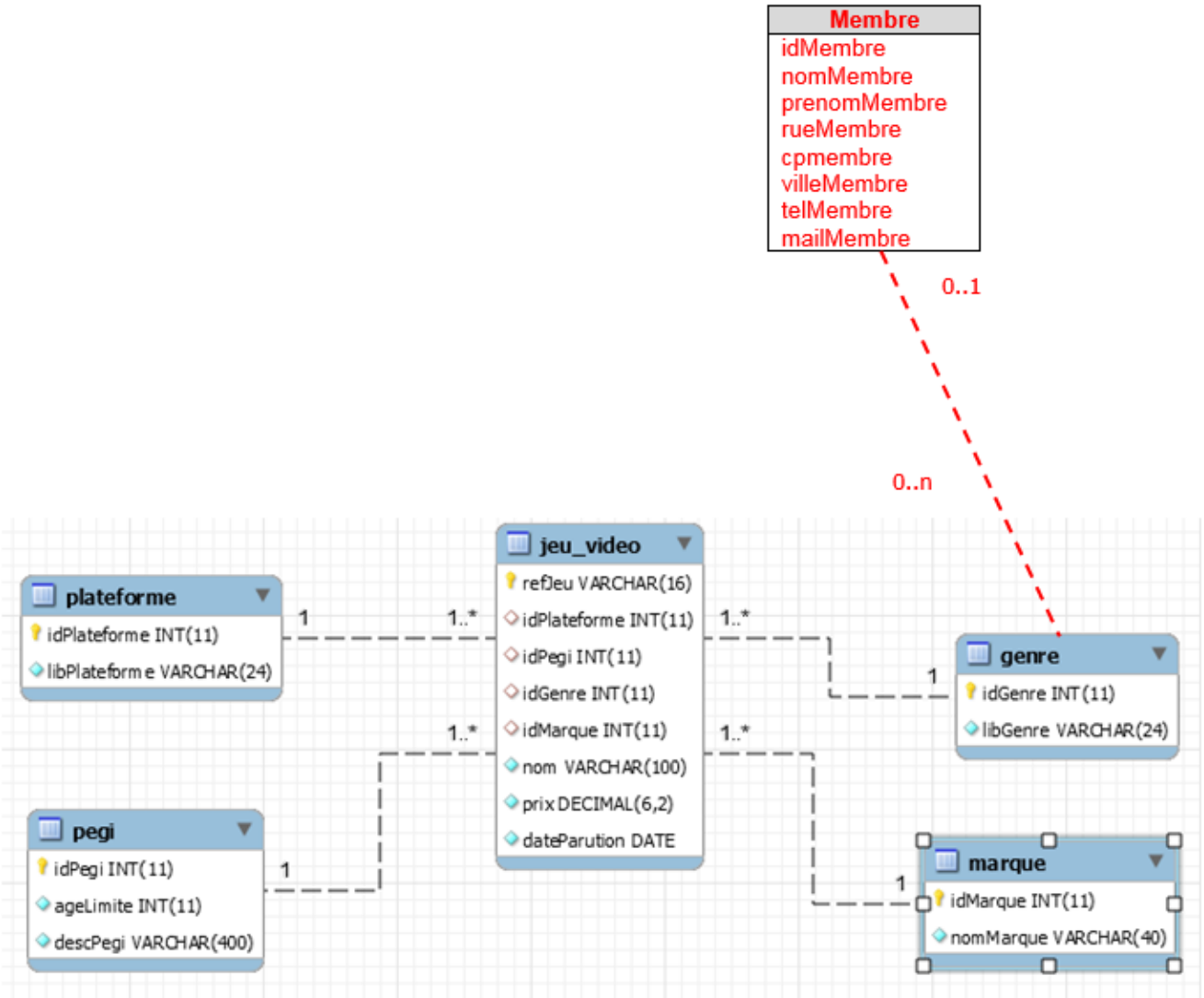


Objectifs des dossiers :

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 àux 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 dePpegi
clé étrangère : idPlateforme en référence à idPlateforme de Plateforme

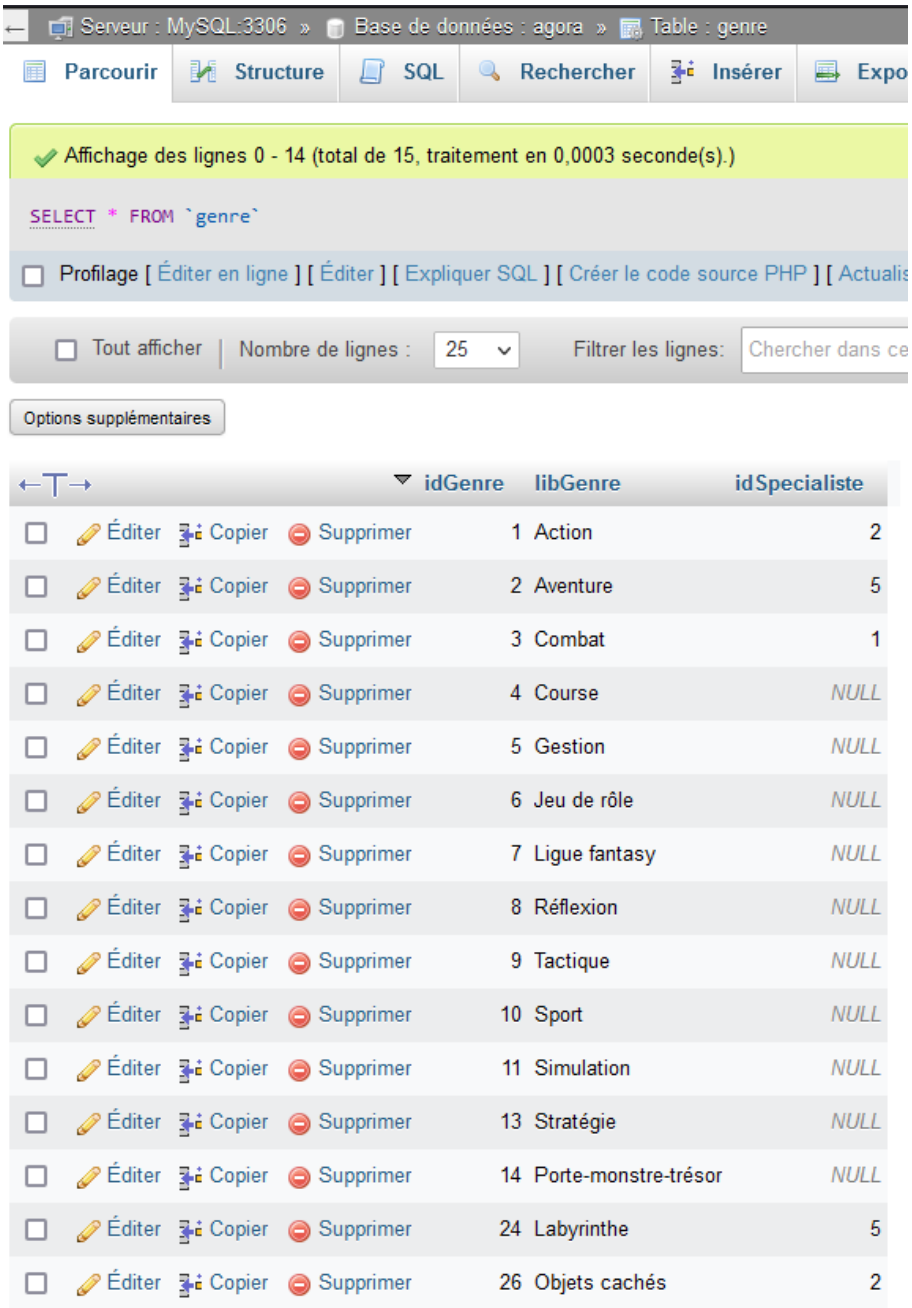
Gestion des genres

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.

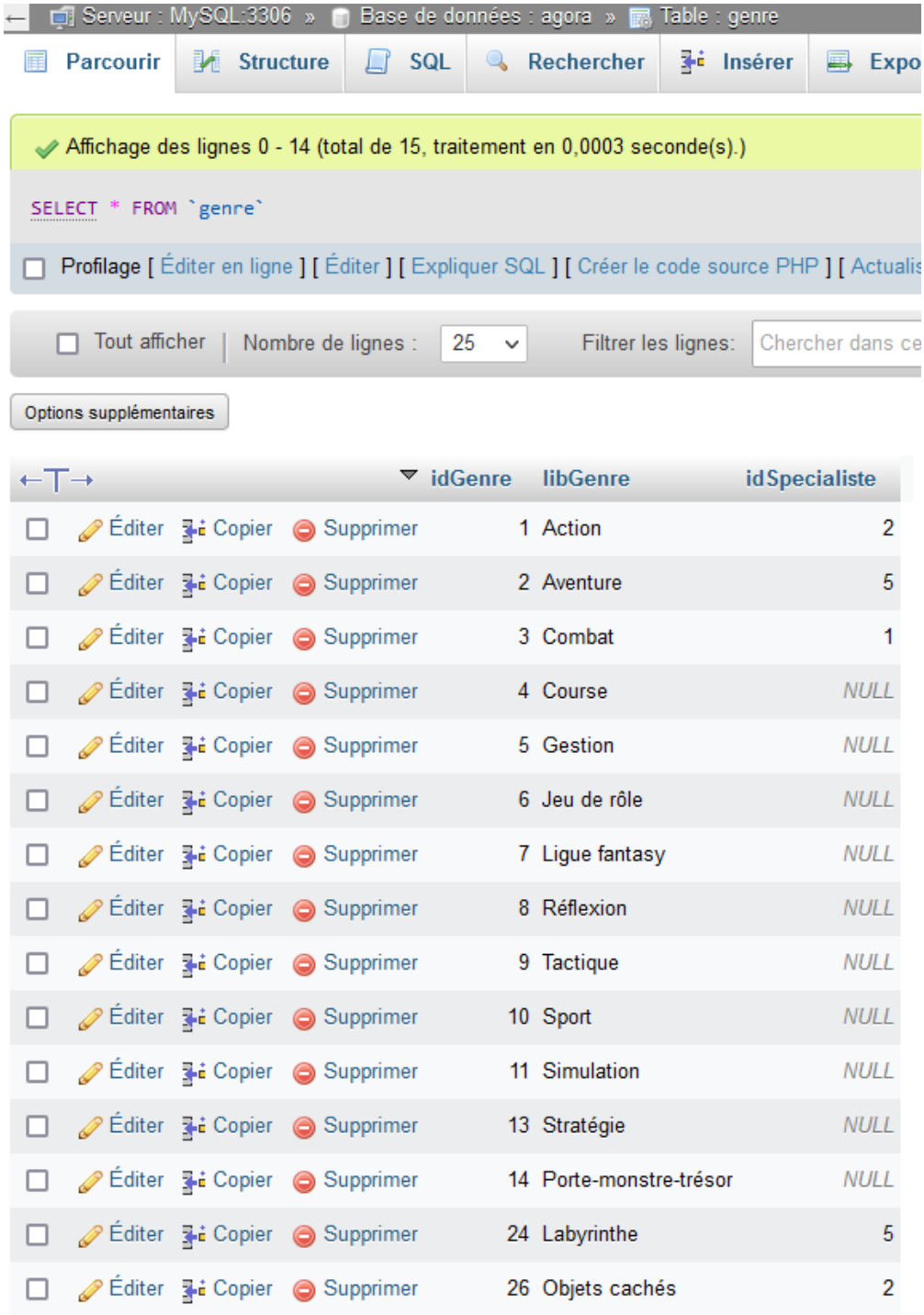
Tutoriel de mise en application de la gestion des genres

1. Modification de la base de données

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



1.2 Modification de quelques genres pour indiquer leur spécialiste



	idGenre	libGenre	idSpecialiste
Éditer Copier Supprimer	1	Action	2
Éditer Copier Supprimer	2	Aventure	5
Éditer Copier Supprimer	3	Combat	1
Éditer Copier Supprimer	4	Course	NULL
Éditer Copier Supprimer	5	Gestion	NULL
Éditer Copier Supprimer	6	Jeu de rôle	NULL
Éditer Copier Supprimer	7	Ligue fantasy	NULL
Éditer Copier Supprimer	8	Réflexion	NULL
Éditer Copier Supprimer	9	Tactique	NULL
Éditer Copier Supprimer	10	Sport	NULL
Éditer Copier Supprimer	11	Simulation	NULL
Éditer Copier Supprimer	13	Stratégie	NULL
Éditer Copier Supprimer	14	Porte-monstre-trésor	NULL
Éditer Copier Supprimer	24	Labyrinthe	5
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)

```php
/**
 * 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

```php
/**
 * 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

```php
/**
 * 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***

```php
/**
 * 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'ajoût, 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>
```

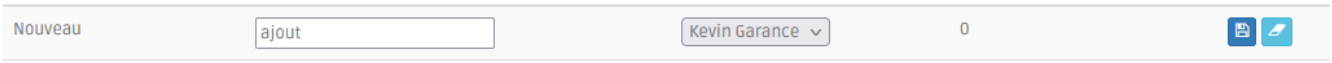
2.3 Contrôleurs :

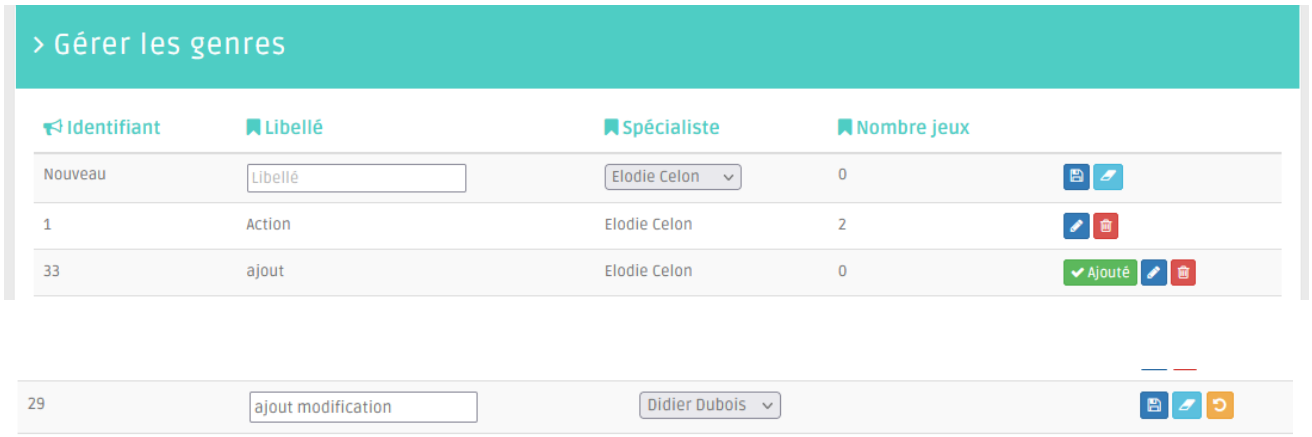
2.3.1 Adaptation du contrôleur secondaire ***c_gererGenres.php*** : il faut traiter également le spécialiste pour les cas d'ajout, de modification et de consultation

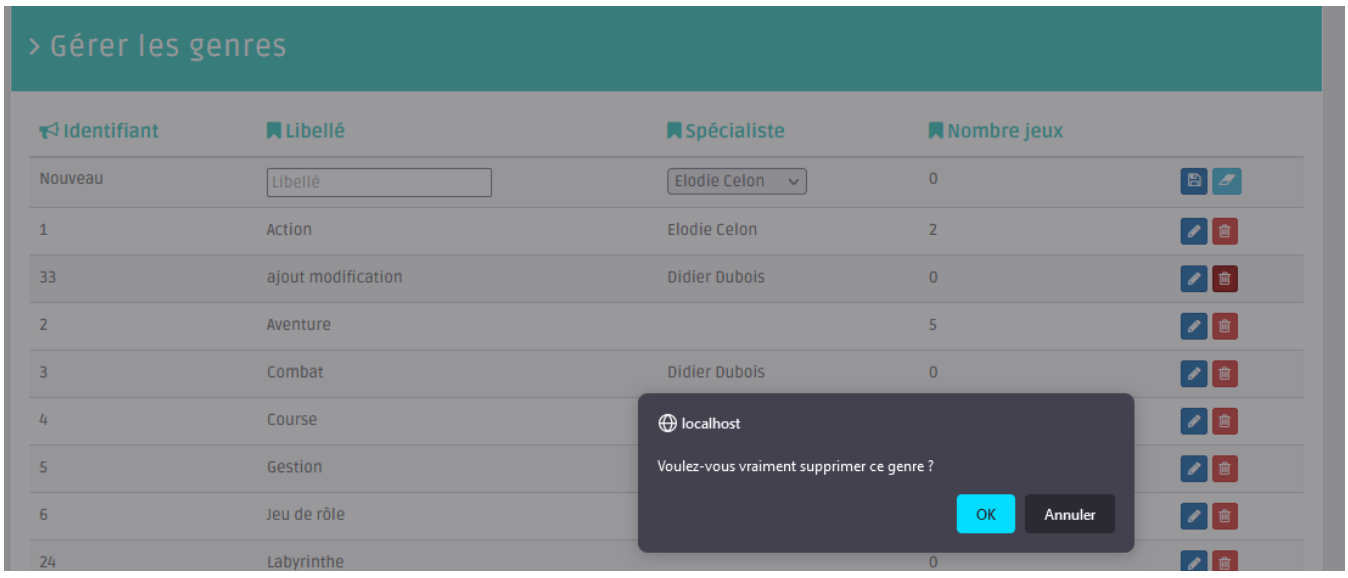
```php
<?php
// si le paramètre action n'est pas positionné alors
//      si aucun bouton "action" n'a été envoyé alors par défaut on affiche les genres
//      sinon l'action est celle indiquée par le bouton

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;
        }
```

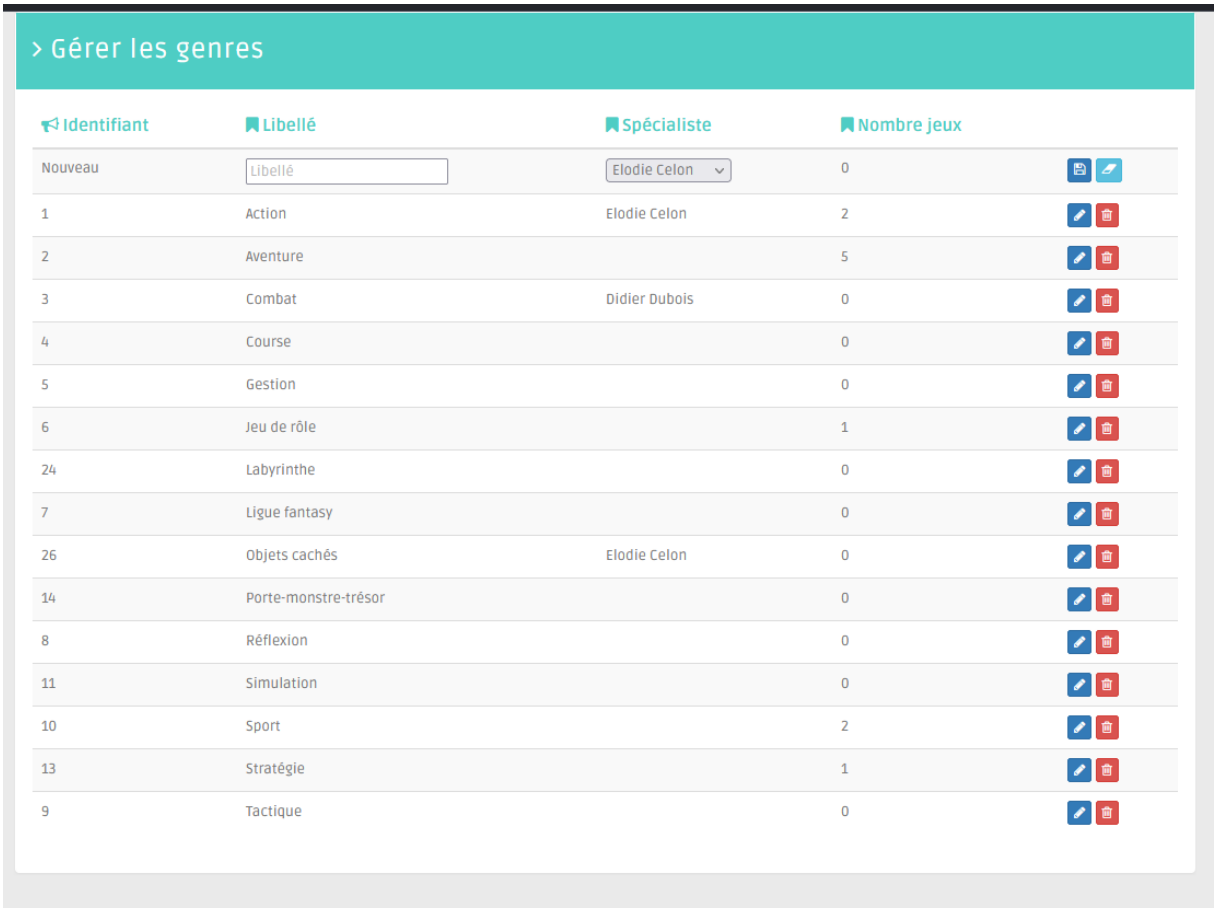
```php
    case 'supprimerGenre': {
            $idGenre = $_POST['txtIdGenre'];
            $db->supprimerGenre($idGenre);
            //  à compléter, voir quelle méthode appeler dans le modèle
            break;
        }
}

// l' affichage des genres se fait dans tous les cas
$tbMembres  = $db->getLesMembres();
$tbGenres  = $db->getLesGenresComplet();
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)







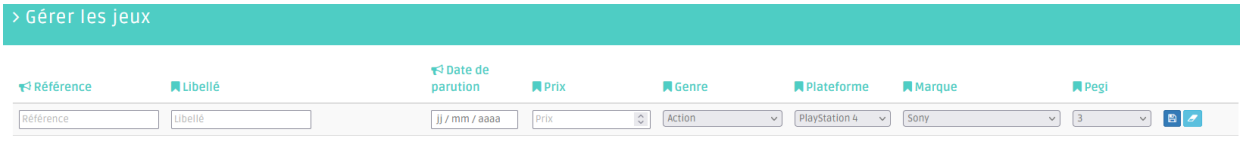


Gestion des marques, pegis, plateformes

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 necesssite la selection de ces attributs via des listes déroulantes.



Aperçu

```php
<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

```php
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;
    }
```

Controleur

```php
// 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.

Mission 2

Authentification

Dans le contexte Agora l'identité numérique est stockée dans la base de données. Un formulaire d'authentification permet aux utilisateurs de saisir leurs données d'identification pour accéder à l'application.

Dans cette mission, nous transmettons le hash ou empreinte du mot de passe, en utilisant l'algorithme de hachage SHA512 qui produit un hash de 512 bits soit 128 caractères hexadécimaux. Ce n'est donc pas le mot de passe en clair qui transitera via le réseau mais son empreinte numérique.

Méthodes de sécurisation des mots de passe :

- Fonction de hachage (SHA256, SHA512, bcrypt, …)

Une fonction de hachage convertit un texte en une suite de caractères assez courte nommé l'empreinte (hash) (ou condensé, haché, message digest).
Il est impossible de déchiffrer l'empreinte pour revenir au texte d'origine

- Technique du (grain de) sel

Il s'agit de compliquer le piratage en concaténant des caractères supplémentaires (le grain de sel, de préférence une longue chaine de caractères) au mot de passe à hacher.
Le grain de sel peut lui-même être haché, aléatoire, regénéré …
Dans le cadre d'une authentification, il est recommandé d'utiliser un grain de sel propre à chaque utilisateur et stocké dans la base de données.
Tests de grain de sel en PHP : http://info.crypto.free.fr/graindesel.php

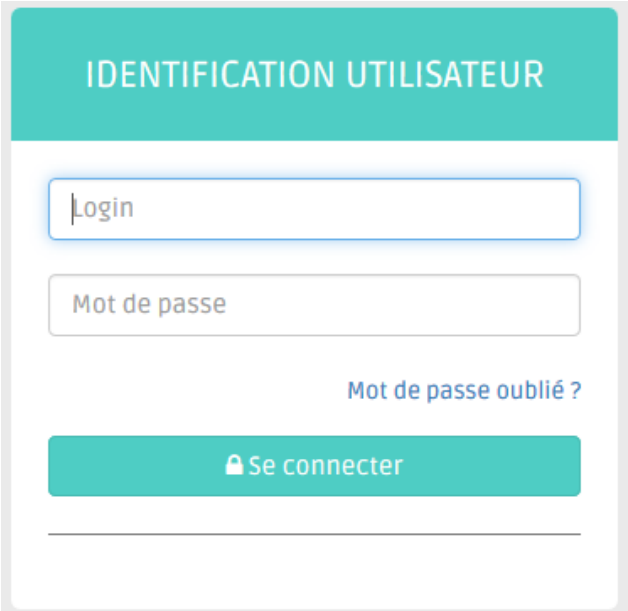
- Captcha

Le captcha est un système couramment utilisé sur les sites Web pour vérifier qu'un humain est bien ce qu'il prétend être, et non une machine. Il peut ainsi arrêter les attaques par force brute en cours.

Principales étapes de la mise en œuvre de l'authentification

- Création d'un utilisateur avec des droits restreints sur la base
- Adaptation de la base de données
- Développement du modèle (MVC)
- Développement de la vue : le formulaire d'authentification
- Hachez le mot de passe
- Développement du contrôleur
- Transmission de données via les sessions
- Adaptation du contrôleur principal : index.php
- Création du contrôleur secondaire : c_connexion.php
- Adaptation des vues v_header et v_accueil
- Déconnexion
- Test

Formulaire d'authentification créé



Mission 3

Moteur de template Twig

Dans le contexte de la MJC AGORA, la troisième mission consiste à améliorer la lisibilité et la maintenabilité du code des vues en utilisant le moteur de template Twig.

Présentation

Twig est un moteur de template qui a été créé par SensioLabs, les créateurs du framework Symfony.
On le retrouve nativement dans les frameworks Symfony et Drupal8, mais il peut être installé sur la majorité des frameworks ainsi que dans un environnement PHP.

 https://twig.symfony.com/

Notions

- **Template**

Le terme Template peut se traduire par modèle, gabarit ou patron. Les templates nécessitent un moteur (script) traitant un modèle de mise en page (le template).

- **Moteur de template**

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, …).

Principales étapes de la mise en œuvre de Twig

Préambule

*Pour installer Twig, nous utiliserons l'outil **Composer***

***Composer** est un gestionnaire de dépendances PHP, dont la première version est sortie en 2012.*
Les dépendances, dans un projet, ce sont toutes les bibliothèques (dans notre cas, ce sera twig) dont le projet dépend pour fonctionner.
Ce système de gestion fonctionne grâce à la centralisation des informations de chaque bibliothèque.

Etapes :

- Installation de Twig
- Installation de Composer
- Ajout de Twig à l'application
- Création du template de base
- Adaptation du fichier index.php
- Création du template de la page d'accueil
- Création du template de la page de connexion
- Adaptation du contrôleur c_connexion.php
- Création du template de la page de gestion des genres
- Création du template de l'affichage d'une liste déroulante
- Adaptation du contrôleur c_gererGenres.php
- Test

Mission 4

Utilisation du framework Symfony

Les besoins de la MJC AGORA s'accroissant, l'application AgoraBo sera progressivement complétée avec de nouvelles fonctionnalités. Afin d'en faciliter la maintenabilité et l'évolutivité, il a été décidé de refondre l'application pour utiliser le framework Symfony.

Présentation

Symfony est un ensemble de composants PHP, un framework d'application Web, une philosophie et une communauté - tous travaillant ensemble en harmonie.

Symfony est un des framework les plus prisés du web, qui, plus qu'un simple support de travail, est une technologie alliant rapidité de développement, sécurité, modularité et évolutivité du code. C'est un framework PHP open Source avec une architecture MVC (Model, View, Controller). Il permet aux développeurs de créer des applications web sans codage monotone et étendu. Grâce à ce framework, ils peuvent se servir d'outils pour réaliser un travail normé et organisé, poser les bases d'un projet web, suivre une architecture et une logique de travail strictes et rigoureuses. Cet applicatif est source de productivité pour un développeur car il dispose de "composants" qu'il est libre d'utiliser ou non dans son travail de développement web.

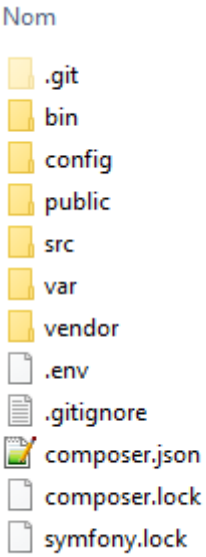
Principales étapes de la mise en œuvre de Symfony

Création du squelette de l'application Symfony

Dans une fenêtre de commande, positionné sur la racine web www, il faut saisir la commande : symfony new AgoraBoS

```
C:\wamp64\www>symfony new AgoraBoS

INFO A new Symfony CLI version is available (5.4.19, currently running 5.4.18).

     If you installed the Symfony CLI via a package manager, updates are going to be automatic.
     If not, upgrade by downloading the new version at https://github.com/symfony-cli/symfony-cli/releases
     And replace the current binary (symfony.exe) by the new one.

* Creating a new Symfony project with Composer
  (running C:\ProgramData\ComposerSetup\bin\composer.phar create-project symfony/skeleton C:\wamp64\www\AgoraBoS  --no-interaction)

* Setting up the project under Git version control
  (running git init C:\wamp64\www\AgoraBoS)

[OK] Your project is now ready in C:\wamp64\www\AgoraBoS


C:\wamp64\www>
```

Un dossier AgoraBoS a été ajouté avec le contenu ci-dessous.

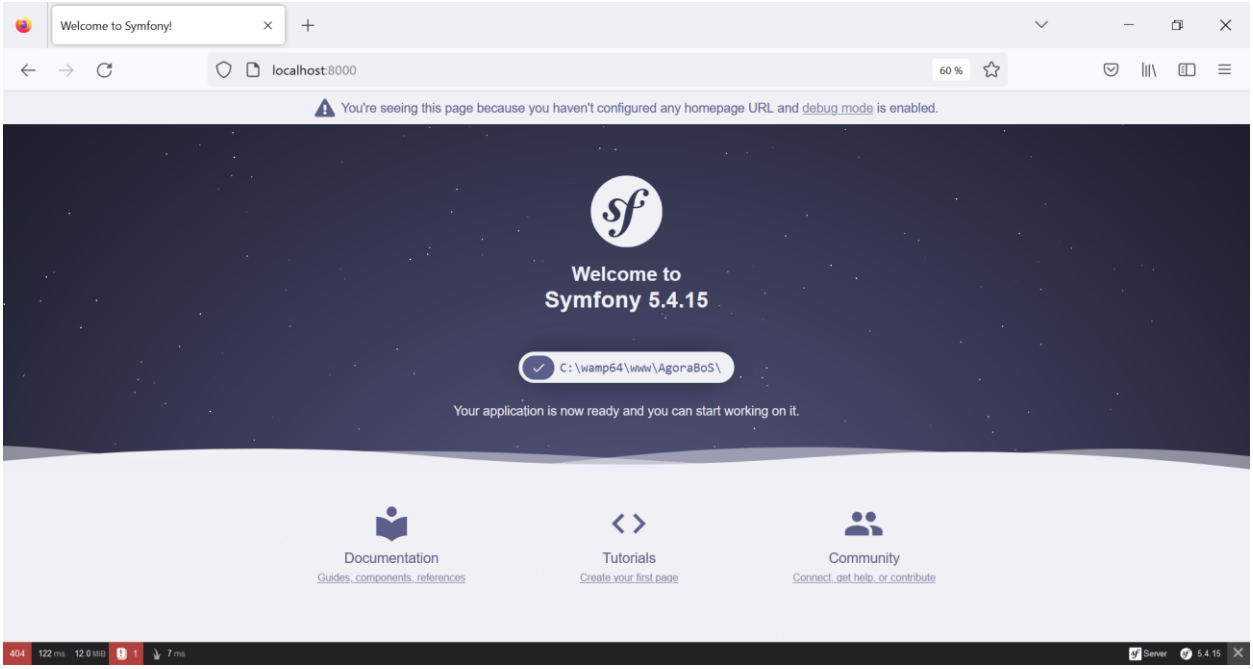


Démarrage du serveur et de l'application

Un serveur web est intégré à Symfony depuis la version 5.4 de PHP. Ainsi on est sûr que le serveur utilisera la même version de PHP que celle utilisée en ligne de commande. Pour le démarrer, il faut lancer la commande : symfony serve –no-tls

```
C:\wamp64\www\AgoraBoS>symfony serve --no-tls

INFO A new Symfony CLI version is available (5.4.19, currently running 5.4.18).

      If you installed the Symfony CLI via a package manager, updates are going to be automatic.
      If not, upgrade by downloading the new version at https://github.com/symfony-cli/symfony-cli/releases
      And replace the current binary (symfony.exe) by the new one.

Following Web Server log file (C:\Users\delio\.symfony5\log\e22671e67880eeecfbf1f81dec052965ca49763e.log)
Following PHP-CGI log file (C:\Users\delio\.symfony5\log\e22671e67880eeecfbf1f81dec052965ca49763e\79ca75f9e90b4126a5955a33ea6a41ec5e854698.log)

 [WARNING] The local web server is optimized for local development and MUST never be used in a production setup.

 [OK] Web server listening
      The Web server is using PHP CGI 7.4.9
      http://127.0.0.1:8000

[Web Server ] Nov 22 08:56:29 |DEBUG  | PHP    Reloading PHP versions
[Web Server ] Nov 22 08:56:30 |DEBUG  | PHP    Using PHP version 7.4.9 (from default version in $PATH)
[Web Server ] Nov 22 08:56:30 |INFO   | PHP    listening path="C:\\wamp64\\bin\\php\\php7.4.9\\php-cgi.exe" php="7.4.9" port=57697
```

Dans un navigateur, on lance l'application en saisissant dans la barre d'adresse : localhost:8000
On obtient la page ci-dessous

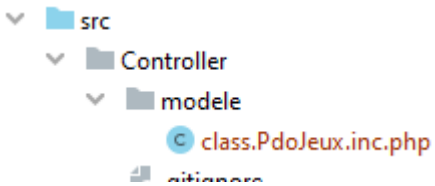


Le modèle

Le framework Symfony est un ensemble de composants. Dans cette version nous utiliserons le modèle de l'application existante, à savoir la classe d'accès aux données PdoJeux.

Dans le dossier src/Controller il faut créer un nouveau dossier **modele**.

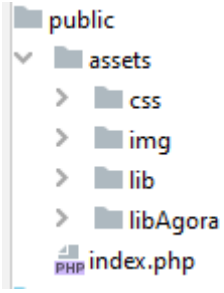
Copier le fichier existant class.PdoJeux.inc.php dans le dossier src/Controller
On obtient :



Dans le fichier .env à la racine du projet, ajouter les paramètres pour l'accès à la base de données. Dans l'ancienne application ces paramètres de configuration figuraient dans le fichier app/ _config.inc.php.

Les vues

On commence par copier le style et les scripts javascript. Création d'un dossier assets dans le dossier public et copie des dossiers se trouvant dans le dossier web de l'ancienne version d'AgoraBo.
On obtient :



Twig est le moteur de template préconisé par le framework Symfony. Comme il a été déjà utilisé dans l'application existante, les vues vont s'intégrer sans trop de modifications.

On supprime le fichier templates/base.html.twig . C'est le template de base (père) et on va le créer à partir de notre existant.

Copier les vues suivantes du projet existant dans le dossier templates de AgoraBoS :
accueil.html.twig
layout.html.twig
connexion.html.twig
lesGenres.html.twig
liste.html.twig
menu.html.twig

Renommer la vue layout.html.twig en base.html.twig . C'est le nom par défaut pour un template père dans Symfony 5.

Création des contrôleurs et des routes

Lorsqu'une requête HTTP arrive au serveur, Symfony essaie de trouver une route qui corresponde au chemin de la requête. Une route est le lien entre le chemin de la requête et une fonction devant créer la réponse HTTP associée à cette requête.
Cette fonction est nommée "contrôleur". Dans Symfony, la plupart des contrôleurs sont implémentés sous la forme de classes PHP. On peut créer ces classes manuellement, mais Symfony permet de nous aider à aller plus vite.