Module : PPE3

|  |
| --- |
| **PROJET BMG**  **Mission 4 : Implémentation du modèle MVC** |

La SSII dans laquelle vous évoluez a en charge le développement d'une application de gestion de la bibliothèque municipale de Groville (BMG). Pour le moment, l’objectif est de fournir aux bibliothécaires les outils nécessaires à la gestion du fonds.

Dans la quatrième mission, l’objectif est de consolider l’utilisation du patron de conception MVC en organisant le code en couches. L’utilisation de la programmation objet est généralisé dans le modèle.

Les développements abordés dans cette mission modifient les choix effectués dans la mission précédente sans ajouter de nouvelles fonctionnalités.

**Pré-requis :**

* Avoir terminé la mission 3.
* Savoir utiliser l’interface d’accès aux données PDO et les requêtes préparées.
* Connaître le patron de conception MVC.
* Savoir interpréter un diagramme de cas d’utilisation et un diagramme de classes.
* Connaître les bases de la programmation objet.

**Feuille de route :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Numéro** | **Tâches** | **Outils/langages** | **Modalités** |
| **1** | Mise en place de la mission 4 | NetBeans, PHP | Professeur/classe |
| **2** | Restructuration du site | NetBeans, PHP | Professeur/classe |
| **3** | Codage du contrôleur secondaire | NetBeans, PHP | Professeur/classe |
| **4** | Codage de la vue permettant d’afficher les négociants | NetBeans, PHP | Professeur/classe |
| **5** | Codage de classe d’accès aux données | NetBeans, PHP, PDO | Professeur/classe |
| **6** | Codage du modèle | NetBeans, PHP | Professeur/classe |
| **7** | Codage du cas d’utilisation « ajouter un négociant » | NetBeans, PHP | Professeur/classe |
| **8** | Codage du cas d’utilisation « consulter un négociant » | NetBeans, PHP | Professeur/classe |
| **9** | Codage du cas d’utilisation « modifier un négociant » | NetBeans, PHP | Professeur/classe |
| **10** | Codage du cas d’utilisation « supprimer un negociant » | NetBeans, PHP | Professeur/classe |

**Tâche n°1**

* Procéder à une sauvegarde de la structure actuelle dans une archive.
* Récupérer le site corrigé de la mission 3.
* Créer une nouvelle entrée dans l’arborescence du site pour la mission 4 et déposer les fichiers du corrigé de la mission 3.
* Créer un nouveau projet PHP dans Netbeans. Créer un dépôt GIT sur NetBeans pour assurer le suivi de la mission. Créer un repository sur GITHUB nommé BMG-4 et « pusher » le code avec le message initial suivant : Mise en place du projet : copie du code de la mission 3

**Tâche n°2**

Nous commençons par adapter le code déjà produit au modèle MVC.

Dans une première phase, nous nous intéressons à la gestion des genres (un des cas d’utilisation décrit en annexe).

* Ajouter un dossier « controleurs » et un dossier « vues ».
* Déplacer les fichiers « header.php », « footer.php », « home.php » et « menu.php » vers le dossier « vues ». Ajouter un préfixe « \_v\_ » pour ces fichiers (il s’agit d’une convention pour distinguer une vue à part entière (externe) des sous-vues utilisées dans d’autres vues).
* Editer le fichier « index.php » (le contrôleur principal) :
  + Adapter la structure « switch » en consultant le diagramme de cas d’utilisation donné en annexe et en testant la valeur de la variable « uc » : « uc » détermine le cas d’utilisation, elle a été transmise par l’URL et sera récupérée en conséquence.
  + Si cette variable n’a pas de valeur, c’est que l’on arrive sur l’application. « uc » prend à ce moment la valeur « home ». Sinon, « uc » peut avoir les valeurs suivantes : gererGenres, gererAuteurs, gererOuvrages.
  + Chaque cas d’utilisation appelle un contrôleur, chaque contrôleur a le nom du cas préfixé par « c\_ ». Ces contrôleurs sont : c\_gererGenres.php, c\_gererAuteurs.php, c\_gererOuvrages.php.
  + La valeur « home » de « uc » provoquera un chargement de la vue « \_v\_home.php ».
* Tester le site (appel à index.php) et l’affichage de la page d’accueil.
* Faire un commit et un push.

**Tâche n°3**

* Modifier le lien présent dans le menu, d’accès à la gestion des genres.

Création du contrôleur secondaire pour gérer le « CRUD » des genres. Dans ce contrôleur, la vue appelée par défaut est la liste des genres. Les autres actions appellent les vues spécialisées dans l’ajout, la modification ou la suppression d’un genre.

* Ajouter un fichier « c\_gererGenres.php » dans le dossier « controleurs ».
* Placer le code permettant la récupération de l’action dans l’url et l’appel des vues en fonction de l’action reçue (s’inspirer du fichier « index.php »). La variable « action » pourra prendre les valeurs suivantes :
  + listerGenres,
  + consulterGenre
  + ajouterGenre,
  + modifierGenre,
  + supprimerGenre.

* Tester différentes url pour le use case «c\_gererGenre . Le système affiche une page vide. Illustration :



Faire un commit et un push.

**Tâche n°4**

Nous allons à présent transformer la page « listerGenres.php » en vue « v\_listeGenres.php ».

* Déplacer le fichier « listerGenres.php » dans le dossier « vues ».
* Le renommer « v\_listeGenres.php».
* Editer le fichier et enlever tout ce qui ne concerne pas la présentation : garder toutefois pour le moment la partie d’interrogation de la base de données : $cnx = connectDB() … disconnectDB($cnx), code qui, vous le savez n’a rien à faire dans la vue et que nous déplacerons ultérieurement.
* Modifier les liens présents dans la liste (Ajouter un genre, lien sur l’ID du genre). Ils ne fonctionneront pas (page vide) car cela n’a pas encore été codé.
* Tester, l’affichage des genres est fonctionnel.
* Faire un commit et un push.

**Tâche n°5**

Avant de coder la couche métier, nous allons améliorer la couche d’accès aux données. Actuellement, cette couche est gérée via des fonctions placées dans le fichier « \_data.lib.php ». Nous allons transformer ces fonctions en les encapsulant dans une classe.

* Effacer le fichier « \_data.lib.php » du dossier « include ».
* Créer un dossier modèle
* Y placer le fichier PdoDao.class.php fourni en annexe et étudier son contenu.

**Tâche n°6**

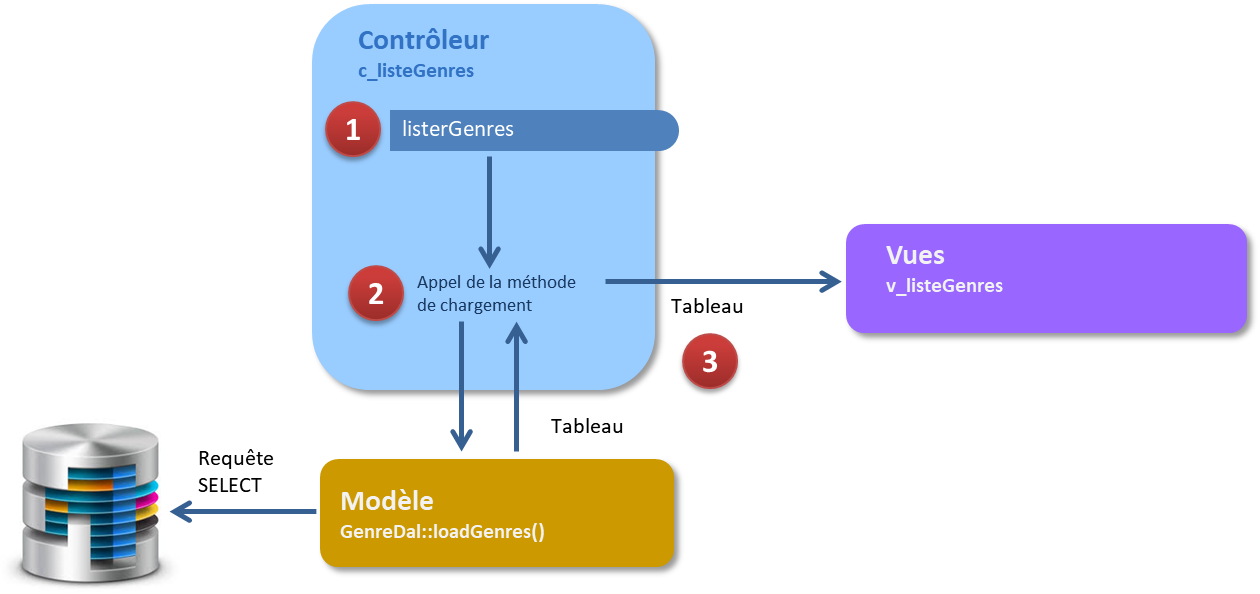
Après le contrôleur et la vue, il nous reste à implémenter le modèle.

* Dans un premier temps, nous allons implémenter la couche « référence » qui contient la classe « Genre » : Placer le fichier « \_reference.lib.php » fourni dans le dossier « include ».
* Créer un fichier GenreDal.class.php (Dal : Data Access Layer) dans le dossier modèle. Ce fichier contiendra toutes les fonctions d’accès à la base de données pour les genres et utilisera les méthodes écrites dans le PdoDa 🡪
  + Ajouter ce code dans le fichier : require\_once ('PdoDao.class.php');
* Revenons à la vue v\_liste\_genres.php. Dans la mission 3, cette vue crée, valorise puis manipule une variable $lesGenres (un tableau). On va déplacer le code de récupération de cette variable dans des fichiers afin d’adapter la structure au modèle MVC.

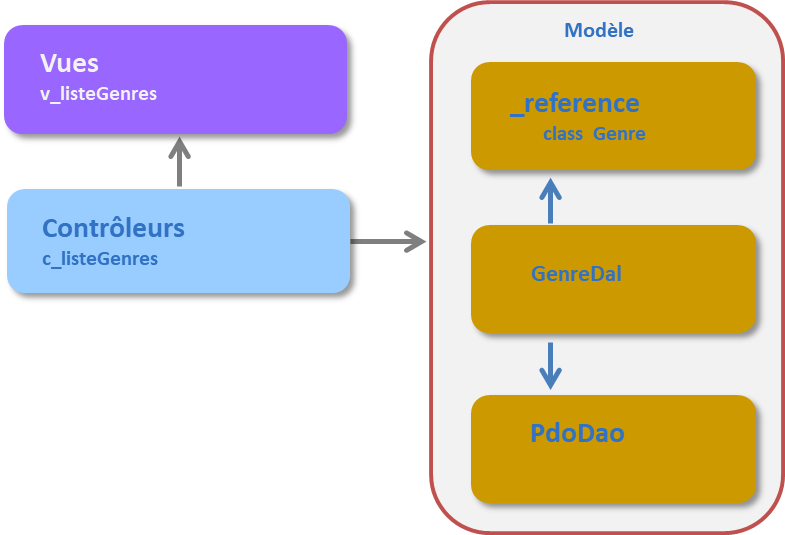
|  |  |
| --- | --- |
| **Modèle** | **Dans le fichier GenreDal.class.php :** |
| L’interrogation de la base de données se fait dans le modèle :  Petite nouveauté : nous allons traiter des objets. |  |
| **Contrôleur** | **Dans le fichier c\_gererGenres.php** |
| Le contrôleur fait le lien entre le modèle et la vue. |  |

* Tester !
* Faire un commit et un push ; message : *Tâche 6 : liste des genres avec modèle, vue et contrôleur*

**Synthèse :**



*Cinématique de la liste des genres*



**Tâche n°7**

Dans cette étape nous allons nous attaquer à l’ajout d’un genre.

*[voir annexe : cinématique de l’ajout d’un genre]*

* Déplacer le fichier « ajouterGenre » dans le dossier « vues » et le renommer « v\_ajouterGenre.php ».
* Editer le fichier et enlever les parties HTML au début et à la fin du fichier que nous avons placées dans les sous-vues. Enlever également l’inclusion des sous-vues ; elles sont incluses dans le fichier index.php.

Le formulaire de saisie et la gestion des erreurs ne sont pas modifiés. En revanche, les instructions réalisant les contrôles et l’ajout en base sont obsolètes.

Comme dans la mission précédente, nous allons opter pour une autre répartition du code dans notre architecture MVC. Il n’est pas souhaitable que la vue accède directement au modèle. C’est donc le contrôleur qui doit gérer l’ajout en sollicitant une méthode du modèle.

Quelle composante du modèle doit s’occuper de l’ajout ? La classe « Genre » ne peut pas se charger elle-même de la création d’une de ses instances. C’est donc la classe « GenreDal » qui va prendre en charge cette opération.

Sur le plan de l’affichage, l’opération d’ajout sera décomposée en deux phases : la saisie (formulaire et affichage des erreurs éventuelles) et la validation de l’ajout (retour après la saisie) qui vérifie les données puis affiche un message de bonne fin d’exécution ou non. Ces deux phases seront distinguées via une nouvelle variable : option. Il y aura donc deux vues et deux cas dans le contrôleur (option = saisirGenre ou option = validerGenre).

*[voir annexe : cinématique de l’ajout d’un genre]*

* Ajouter une méthode « addGenre() » dans la classe « GenreDal ». Cette fonction reçoit en paramètre un tableau contenant les valeurs correspondant au genre à ajouter. Elle procède à l’ajout et renvoie le code du genre ajouté. En voici le code :

/\*\*

\* ajoute un genre

\* @param string $code : le code du genre à ajouter

\* @param string $libelle : le libellé du genre à ajouter

\* @return le code du Genre

\*/

public static function addGenre($code, $libelle) {

$cnx = new PdoDao();

$qry = 'INSERT INTO genre VALUES (?,?)';

$res = $cnx->execSQL($qry, array(// nb de lignes affectées

$code,

$libelle

)

);

if (is\_a($res, 'PDOException')) {

return PDO\_EXCEPTION\_VALUE;

}

return $code;

}

* Déplacer le code réalisant les contrôles de saisie dans le contrôleur :
  + Avant tout, déclarer les variables qui nous seront utiles dans tout le contrôleur ; à placer donc en tête du contrôleur, avant d’entrer dans le 1er switch (action)

// variables pour la gestion des messages

$titrePage = 'Gestion des genres';

// variables pour la gestion des erreurs

$tabErreurs = array();

$hasErrors = false;

* + Dans le switch ajouterGenre, déclaration des variables pour la saisie.

// initialisation des variables

$hasErrors = false;

$strCode = '';

$strLibelle = '';

* + Récupérer l’option dans l’url (saisir ou valider ?).

// traitement de l'option : saisie ou validation ?

if (isset($\_GET["option"])) {

$option = htmlentities($\_GET["option"]);

} else {

$option = 'saisirGenre';

}

* + Dans le cas d’une saisie, appeler la vue contenant le formulaire.

switch ($option) {

case 'saisirGenre' : {

include 'vues/v\_ajouterGenre.php';

} break;

* + Dans le cas d’une validation,
    - Réaliser les contrôles de saisie.

…

* + - On teste les cas d’erreurs :
      * La zone obligatoire txtCode n’est pas saisie ou correcte
      * La zone obligatoire txtLibelle n’est pas saisie ou correcte
      * Si les 2 zones sont correctes mais que le code existe déjà (\*)

Alors dans ces cas :

* on positionne la variable $hasErrors à true,
* on ajoute au tableau d’erreur un message d’erreur explicite.

(\*) : comment tester le code d’un genre déjà existant : on fait appel au modèle

$doublon = GenreDal::loadGenreByID($strCode);

if ($doublon != NULL) {

…

}

* + - S’il n’y a pas d’erreur, appeler la méthode d’ajout (\*). S’il n’y a pas eu d’erreur d’insertion, ajouter un message de réussite dans la variable $msg et appeler la vue : v\_afficherMessage.php. Ajouter le commentaire suivant :

// include 'vues/v\_consulterGenre.php'; Ce sera décommenté par la suite.

* + - S’il y a une des erreurs :

$msg = "L'opération d'ajout n'a pas pu être menée à terme en raison des erreurs suivantes :";

$lien = '<a href="index.php?uc=gererGenres&action=ajouterGenre">Retour à la saisie</a>';

include 'vues/v\_afficherErreurs.php';



* Dans la vue « v\_ajouterGenre.php », supprimer le code inutile.
* Les deux méthodes utilisées **dans GenreDal.php** :

/\*\*

\* charge un objet de la classe Genre à partir de son code

\* @param $id : le code du genre

\* @return un objet de la classe Genre

\*/

public static function **loadGenreByID($id)** {

$cnx = new PdoDao();

// requête

$qry = 'SELECT code\_genre, lib\_genre FROM genre WHERE code\_genre = ?';

$res = $cnx->getRows($qry, array($id), 1);

if (is\_a($res, 'PDOException')) {

return PDO\_EXCEPTION\_VALUE;

}

if (!empty($res)) {

// le genre existe

$code = $res[0]->code\_genre;

$libelle = $res[0]->lib\_genre;

return new Genre($code, $libelle);

} else {

return NULL;

}

}

/\*\*

\* ajoute un genre

\* @param string $code : le code du genre à ajouter

\* @param string $libelle : le libellé du genre à ajouter

\* @return le code du Genre

\*/

public static function **addGenre($code, $libelle)** {

$cnx = new PdoDao();

$qry = 'INSERT INTO genre VALUES (?,?)';

$res = $cnx->execSQL($qry, array(// nb de lignes affectées

$code,

$libelle

)

);

if (is\_a($res, 'PDOException')) {

return PDO\_EXCEPTION\_VALUE;

}

return $code;

}

**Les erreurs :**

|  |  |
| --- | --- |
|  |  |
|  |  |

**Annexes**

**Cinématique de l’ajout d’un genre**

