TP3

L'objectif de ce TP est de :
o Mettre en œuvre le motif de conception MVC (Modèle Vue Contrôleur)

Rappel de cours : l'architecture MVC

Développer selon une structuration MVC consiste à diviser son code en 3 parties:

- Le **modèle (M)** est la partie de code correspondant à la gestion des données. Le modèle retourne les données demandées, met à jour les données, réalise les calculs. Le modèle ne connaît pas la vue.
- La **vue (V)** est la partie de code correspondant à la partie visible de l'application, à l'affichage des données.
- Le **contrôleur (C)** est la partie de code qui prend en compte les actions de l'utilisateur appelle le modèle et déclenche l'affichage des vues. Dans la variante MVC que nous utilisons ici, le contrôleur récupère les données par appel du modèle et passe les données aux vues pour affichage. Il assure la communication entre le modèle et les vues qui n'accèdent donc pas directement au modèle.

Cette architecture fonctionne de la manière suivante :
1. L'utilisateur envoie une requête (une demande d'action) au contrôleur.
2. Le contrôleur effectue l'action demandée par l'utilisateur en appelant le modèle.
3. Le contrôleur informe la vue d'un changement d'état du modèle.
4. La vue met à jour son affichage.
5. L'utilisateur voit le résultat de sa requête.

Le but de ce TP est de transformer une application intégrant uniquement la notion de modèle, en une application qui suit la structuration MVC. Vous pouvez vous inspirer du code de l'application blog présentée en cours et fournie pour ce TP.

A FAIRE :

1) Compréhension de la version MVC de l'application 'blog'

1. Récupérez la version MVC de blog sur l'intranet et testez la. Vérifiez que le comportement est identique à la version classique.

2. Exécutez l'application et en focalisant sur ce qui s'affiche dans la barre d'url (page appelée et paramètres).

3. Notez les répertoires et fichiers composant l'application

3. Identifiez bien le rôle de chaque fichier

4. Analysez de façon détaillée le code des actions du contrôleur display (Display.php). Peut-on décrire de façon générale la suite d'opérations accomplies par toute méthode correspondant à une action du contrôleur?

5. Identifiez les variables d'état (qui se propagent de page en page et qui correspondent à un état de l'application) et les paramètres des actions (qui ne se propagent pas).

2) Passage de l'application images de la version Modèle vers la version MVC

1. Créez une copie de la version Modèle de l'application image. Vous garderez donc une version Modèle en état de fonctionnement qui vous permettra de vérifier à tout moment le comportement attendu de l'application et vous modifierez petit à petit la copie pour arriver à une structuration MVC du code.

2. Reproduisez la structure de répertoires de blog (**controller**, **model**, **view**,...).

3. Recopiez le contrôleur frontal de blog (**index.php**) et adaptez le.

4. Nous préconisons d'utiliser 3 contrôleurs pour cette application. Des contrôleurs nommés par exemple **home.php**, **photo.php** et **photoMatrix.php**. Ils seront respectivement spécialisés dans le traitement des actions relatives à l'accueil (**apropos**), aux photos simples (**first**, **next**, **prev**, **random**,

zoom) et aux photos multiples (**first**. **next**, **prev**, **random**, **more**, **less**). Créez les 3 classes correspondantes et écrivez les en-têtes des fonctions correspondant à chaque action en laissant leurs corps vides. On ajoutera en plus l'action par défaut **index** à chaque contrôleur, correspondant à l'action effectuée lorsque aucune action particulière n'est mentionnée (voir blog).

5. Nous préconisons d'utiliser une vue globale **mainView.php** affichant la globalité (menu et contenu) et des vues spécialisées pour les différents contenus (**homeView.php**, **aproposView.php**, **photoView.php**, **photoMatrixView.php**). Le contrôleur sélectionnera la vue spécialisée en rangeant le nom du script correspondant dans un objet $data (par exemple $data->content="photoview.php") et appellera la vue globale **mainView.php** qui contiendra l'instruction "include(data->content)". Sans pour l'instant les remplir, créez ces différents fichiers y compris le fichier correspondant à une simple classe data sans attribut ni méthode (**data.php**).

6. Nous allons commencer par écrire le contrôleur **home.php**. Il comporte 2 actions: l'action par défaut **index** et l'action **apropos**. Pour ces 2 actions, il n'y a pas de paramètre à prendre en compte. Le contrôleur n'a finalement qu'à sélectionner les vues **homeView.php** ou **aproposView.php** (pur HTML) ($data-> content="homeView.php" ou $data->content="aproposView.php"), préparer le menu ($data->menu['Home']="index.php" etc...) et appeler **mainView.php**. Ecrivez le contrôleur home.php consistant en ces 2 actions. Ecrivez les 3 vues **mainView.php**, **homeView.php** (pur html) et **aproposView.php** (pur html) et testez leur fonctionnement.

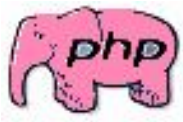
7. Enfin, nous allons écrire l'action **index**, l'action **first** et l'action **next** du contrôleur **photo.** Contrairement au contrôleur **home**, ce contrôleur doit connaître le modèle. Dans son constructeur, comme pour le contrôleur Display de blog, vous devez donc créer une instance du DAO.

- L'action **first** a besoin de récupérer l'url de la première photo auprès du modèle. Il prépare le menu ($data->menu['Home']="index.php" etc...), sélectionne le contenu correspondant à photoView ($data->content="photoView.php"), prépare l'image à afficher ($data->imageURL=...) et appelle la vue MainView.php.
- L'action **index** se résume à appeler "this->first()".
- L'action **next** utilise le paramètre indiquant l'image courante (imgId), appelle le modèle pour récupérer l'image suivante et, de la même façon que first, prépare les données à afficher pour la vue en les rangeant dans $data.

Tester ces actions.

8. Il faudra aussi prendre garde à ne pas oublier de propager les paramètres d'état (imgId, size, nbImg), Aucune information d'état ne doit être conservée en dehors des variables dans l'URL. En particulier, vous n'avez pas besoin de fichiers, de sessions PHP, de coockies, de variables Javascript, etc.

9. Complétez toutes les actions jusqu'à ce que le comportement de la version MVC de l'application soit identique à celle d'origine.



Non, vous ne voyez pas des éléphants roses, la structuration MVC rend bien votre code plus lisible et plus maintenable. Bon courage.