Électif Technologies Web 2.0 - 2018
Cas d'étude Tiny MVC

Objectif : Appréhender une architecture de type MVC. Développer un site multi-utilisateurs, savoir réaliser le schéma d'architecture d'un site Web, comprendre un système d'information comportant des clés étrangères, réaliser des requêtes SQL comportant des jointures. Comprendre l'originalité et l'intérêt des solutions du Web 2.0.

L'architecture que nous utiliserons est fondée sur le concept « MVC ». Trois fichiers php et des **templates** générent des morceaux d'interface web (éléments de menu, formulaires, etc.)

Le fichier **modele.php** offrira la couche modèle : des fonctions « métier » d'accès à la base de données, en lecture/écriture.

Le fichier **controleur.php** sera notre contrôleur : la page destinataire des soumissions de formulaire. Elle exécutera des appels aux fonctions de la page modele.php pour mettre à jour la base de données, puis redirigera vers la page appelante pour rafraîchir le navigateur.

Le fichier **index.php** s'occupera de générer les **vues**. Un paramètre « view » devra être systématiquement passé en paramètre de façon à sélectionner l'affichage choisi. Selon les valeurs du paramètre view, il faudra parfois également passer d'autres paramètres. La page index s'occupera alors d'afficher la vue désirée, en utilisant des appels aux fonctions de la page modele.php pour récupérer les données.

On utilisera systématiquement des requêtes de type **GET** pour faciliter le débogage des pages.

Exercice 1 : page index, chaînes de requête, maLibUtils

Objectif : Comprendre l'architecture d'une page index, la forme des chaînes de requête et les vérifications à réaliser pour éviter des messages d'alerte

Déplacez le code fourni dans votre répertoire www.
Parcourez son arborescence à la recherche de la page index, et du template « accueil.php », ouvrez ces fichiers et observez leur contenu. Expliquez ce qui se passe lorsque vous affichez la page index sans arguments, et lorsque vous cliquez sur les liens du menu.
Quel paramètre est indispensable à toutes les vues ?
Ajoutez au menu un lien permettant d'accéder à l'affichage des conversations.
Ajouter le logo de Centrale à la bannière

Exercice 2 : (view=users) phpMyAdmin, maLibSQL.pdo.php, modele.php

Objectif : Savoir importer une base de données en utilisant phpMyAdmin, utiliser la librairie maLibSQL pour générer des vues à partir de cette base de données

Une base de données est fournie avec le sujet de TP. Importez-là sur votre serveur de base de données à l'aide de l'interface phpmyadmin. Définissez quelques utilisateurs, blacklistés ou non.
Editer le fichier modele.php. Inclure la librairie maLibSQL.pdo.php. Paramétrer le fichier config pour vous connecter à la base.
Compléter les fonctions proposées dans la première partie du fichier modele.php en rédigeant les requêtes SQL et en utilisant les fonctions adéquates de la librairie maLibSQL.pdo.php.

- Quel mécanisme permet d'éviter les injections SQL ?

Tester ces requêtes à l'aide du fichier users.php. Vérifier que les résultats sont corrects en modifiant les données depuis phpmyadmin et en réactualisant la page.
Considérons le formulaire présent dans la page users.php

1. A quelle page sont envoyés les données du formulaire ?
2. Quelles données sont envoyées ? Pourquoi y-a-t-il deux boutons submit ?
3. Compléter les labels du menu déroulant pour y afficher les statuts des utilisateurs ('bl' ou 'nbl') en plus de leurs noms.

Exercice 3 : page contrôleur

Objectif : Utiliser une page de centralisation de données nommée « controleur.php » pour implémenter des traitements administratifs sur les utilisateurs. Cette page respecte les contraintes suivantes :

Les données nécessaires à chaque commande (idUser, type d'action, paramètres, etc...) seront toutes envoyées à la page controleur.php
La page controleur.php vérifie les données envoyées par l'utilisateur à l'aide de la fonction valider, pour se protéger des injections SQL
La page controleur.php exécute les actions demandées
La page controleur.php redirige ensuite vers la page appelante, en lui renvoyant les mêmes données ainsi qu'un message de feedback permettant d'afficher des commentaires éventuels dans le formulaire initial.

1. Implémenter la fonctionnalité permettant de changer l'état d'un utilisateur sélectionné (blacklisté ou non). Vérifier que l'exercice 2 est pleinement opérationnel.
2. Améliorer le menu déroulant de manière qu'il sélectionne automatiquement l'utilisateur qui vient d'être édité. Pour cela, faire renvoyer par la page controleur.php l'identifiant de l'utilisateur au moment de la redirection.

Exercice 4 : (view=login) identification et administration des utilisateurs

Objectif : Utiliser les variables de session et des redirections pour sécuriser un site

Utilisez la page contrôleur pour compléter la procédure de connexion :
Constatez ce qui se passe lorsque vous activez le formulaire de connexion et que vous saisissez des identifiants. Quelles fonctions sont mobilisées dans les pages controleur et footer ?
Développez ces fonctions dans le fichier « maLibSecurisation.php ».
Vérifiez que la procédure de login fonctionne à l'aide du pied de page.
Développez la fonctionnalité de déconnexion.

Exercice 5 : (view=conversations) Librairie de création de formulaires, maLibForm.php

Objectif : Développer une librairie de fonctions permettant de faciliter la création de formulaires et de tableaux complexes

Le code précédent réalise l'affichage d'un tableau associatif et d'une liste d'options en utilisant du code qui pourrait être mieux écrit puis capitalisé. Dans cet exercice, nous développons une librairie de fonctions à cet effet. Les fonctions à développer sont listées dans le fichier maLibForms.php. Complétez le code de ces fonctions et utilisez les pour faire fonctionner la page d'affichage des conversations.

1. Complétez la page modele.php pour gérer les conversations
2. Complétez la page conversations.php pour afficher uniquement les champs 'id' et 'theme' dans cet ordre, à l'aide de la fonction mkTable
3. Complétez la fonction mkTable pour que les entêtes des colonnes s'affichent dans des balises <th> et non <td>
4. Complétez la fonction mkLiens dans le fichier maLibForms et remplacez l'appel à mkTable par un appel à mkLiens dans conversations.php
5. Complétez le formulaire en bas de la page conversations.php en utilisant les fonctions mkForm, mkSelect et mkInput, et mettez à jour la page controleur.php pour que les demandes de création et de changement d'état des conversations soient effectivement réalisées
6. Améliorer le menu déroulant de manière qu'il sélectionne automatiquement la conversation qui vient d'être éditée ou créée.
7. Améliorer votre formulaire pour qu'il soit possible de supprimer une conversation.
8. Améliorer votre formulaire pour qu'il soit possible d'ajouter une conversation.
9. **Comparez le code des exercices 2 et 5 : grâce à nos fonctions, le travail de développement est réduit de moitié, le code est plus lisible... Vive la modularité !**

Exercice 6 : (view=chat) messagerie instantanée

Objectif : Implémenter l'affichage des messages d'une conversation en réactualisant la page régulièrement. Mettre en œuvre une démarche de tests multi-utilisateurs. Constater les problèmes posés par une solution « Web 1.0 ».

NB : La vue chat est appelée par la page des conversations lorsque l'on clique sur l'une des conversations disponibles. On lui fournit l'identifiant de la conversation à afficher grâce au paramètre « idConv ».

1. **Page modele.php** : Compléter les fonctions permettant de lister les messages d'une conversation et d'en enregistrer un nouveau
2. **Page chat.php** :
   Afficher le thème de la conversation courante
   Afficher la liste des messages de la conversation courante, au format :
   [Auteur] Message (dans la couleur de l'auteur)
   [Auteur] Message (dans la couleur de l'auteur)
   Ajouter le formulaire permettant de poster un nouveau message, uniquement si la conversation est active ET que l'utilisateur est connecté.
   Le contenu du formulaire sera automatiquement sélectionné lorsque l'on cliquera dedans.
3. **Page controleur.php** : ajouter le code permettant de poster un nouveau message

Pour tester le bon fonctionnement de cette page dans un environnement multi-utilisateurs, on pourra utiliser plusieurs navigateurs de types différents (IE, chrome, ffox) ou tester en binôme à condition de configurer correctement le serveur apache (paramètre listen) pour le rendre accessible sur le réseau de la salle de cours.

4. Pour éviter de devoir appuyer sur F5 régulièrement pour récupérer les nouveaux messages, nous souhaitons rafraîchir la page de conversations toutes les 10 secondes. Proposer une solution.

   Cette méthode a pour désavantage de faire perdre les messages en cours d'écriture par l'utilisateur. Une solution serait de ne plus changer de page pour récupérer les messages, mais plutôt de déclencher des requêtes asynchrones en javascript pour récupérer uniquement les messages qui ne sont pas encore affichés sur la page. C'est l'objet du prochain module !