

Exercice 1 : Il s'agit de sécuriser un peu plus le forum vu dans le cours MVC. L'interface est la suivante :

The screenshot shows a web interface for a forum. It is divided into two main sections. The top section, titled 'Poster un message', contains a login form with three input fields: 'Login :', 'Mot de passe :', and 'Message :'. Below these fields is a blue button labeled 'Envoyer votre msg'. The bottom section, titled 'Les messages déjà postés', displays a list of messages. Each message is shown with the user's name in a blue box followed by the message text in a white box. The first message is from 'alain' with the text 'Bien venu dans ce forum'. The second message is from 'david' with the text 'Bonjour'.

On dispose désormais de deux tables : « user » et « forum ».

La première table contient :

- un champs « id » de type varchar(10) qui représente le login de l'utilisateur (c'est la clé primaire de cette table),
- deux champs « mdp » et « nom » de type text qui contiennent le mot de passe et nom de l'utilisateur.

La deuxième table « forum » est composée :

- d'un champ « id » de type int(11) qui est auto-increment et clé primaire afin que chaque message soit identifié à partir d'un entier,
- un champ « nom » de type varchar(10) qui contient le nom de l'émetteur du message
- un champs « msg » qui contient le corps du message.

Créez la BDD avec ces deux tables puis insérez deux utilisateurs dans la table « user » dont les nom sont respectivement :« Dupont », « Alain ». Mettez leur des logins et mots de passe.

Si un utilisateur souhaite poster un message, il saisit son « login », son « mot de passe » ainsi que son message puis clique sur le bouton « envoyer » . Trois possibilités sont à traiter :

- (1) Si un des champs est vide alors on affiche le message « Un des champs est vide ».
- (2) S'il y a une erreur d'authentification alors on affiche « Login ou mot de passe non valide ».

Pour les deux cas (1) et (2) on affichera également un lien « revenir au forum » pour revenir au début et revoir les discussions. Ce forum est à lecture publique c'est-à-dire que tout le monde peut lire les discussions sans s'authentifier. Par contre l'écriture est privée.

- Si tout est correcte on insère dans la table Forum le message ainsi que le « nom » de l'utilisateur (**pas son login mais son nom**) puis affiche tous les messages déjà postés.

Lorsque l'on saisie l'url de notre forum dans un navigateur, les zones de saisie sont vierges mais tous les messages déjà postés doivent s'afficher par ordre chronologique comme présenté dans l'interface ci-dessus au début de l'exercice.

Première partie du travail :

Reprendre toute l'architecture MVC du forum vu en cours et la tester sans CSS. **On rappelle qu'un fichier qui contient que du Php ne doit jamais être fermé par balise ?>.** Supprimez de la requête d'affichage la partie « limit 10 » pour que toutes les discussions soient affichées et non pas les dix dernières seulement.

Deuxième partie du travail :

Ajouter une fonction `checkUser($login,$mdp)` dans le modèle. Celle-ci renvoie un tableau via `fetchall` contenant le scan de la table « user » lorsque la colonne `login=$login` et la colonne `mdp=$mdp`. En d'autre termes elle exécute la requête « `select nom from user where id='$id' and mdp='$mdp'` ». Le tableau renvoyé via `fetchall` est soit null donc le login ou le mdp est faux, soit le tableau contient seulement un case contenant l'individu de la table « user » qui correspond aux login et mot de passe postés.

Modifiez la fonction `CtrlAjouterMessage` du contrôleur pour qu'elle ait un paramètre supplémentaire et qu'elle fasse appelle à la fonction `checkuser` du modèle qui renverra un tableau (par exemple `$nom`) qui est :

- soit null donc on génère une exception avec le message « Login ou mdp non valide » ,
- soit non null donc on récupère le nom de l'émetteur via `$nom[0]->nom` et insère ce nom et le message dans la BDD via la fonction `ajouterMessage` du modèle.

Troisième partie du travail :

Affectez à votre page la feuille de style de l'exercice 1 de la série de CSS et adaptez la pour avoir un affichage similaire à celui donné ci-dessus. Vous notez que les messages sont affichés en tant que « value » des champs « input » et que le nom de la personne est dans entre deux balises label.

Exercice 2 : lisez l'énoncé entièrement jusqu'à la fin avant de commencer à programmer. Le but final de cet exercice est de concevoir une page web de gestion de clientèle pour le magasin « cod » dont nous avons créé la charte graphique en CSS dans le TD CSS exercice 1. L'interface d'accueil est la suivante :

The screenshot shows the 'cod' website interface. The header is blue with the 'cod' logo and navigation links: Catalogues des produits, Promotions du moments, Service après vente, Contact, and social media icons. The main content area is divided into two columns. The left column contains three sections: 'Ajouter un client' with form fields for Nom, Prenom, Date de naissance, and Tel, and buttons 'Ajouter client' and 'Tout effacer'; 'Recherche d'un client' with a 'Nom du client' field and buttons 'Chercher' and 'Tout effacer'; and 'Afficher tous les clients' with an 'Afficher' button. The right column contains three promotional banners: 'Promotion Apple !' featuring an Apple MacBook Pro, 'Arrivage disques durs' featuring Western Digital, Lacie, and Maxtor hard drives, and 'Destockage clé usb' featuring a USB key promotion.

Pour enregistrer des clients dans notre base il suffit de saisir le nom, prénom, date de naissance et numéro de téléphone puis de cliquer sur « ajouter client ».

On peut visualiser nos clients en cliquant sur le bouton « afficher les clients » et obtenir par exemple :

The screenshot shows the 'Afficher tous les clients' section. It features a large 'Afficher' button. Below it, the 'Les clients de la base' section displays a list of three clients. Each client entry consists of a checkbox, a label 'Num CL X :', and the client's details. The clients are: Alain Bernard (ne le 1980-04-15 joignable sur le 060102), Thomas Dutronc (ne le 1975-12-20 joignable sur le 0608), and Anne Michel (ne le 1960-10-02 joignable sur le 0677284). At the bottom of the list is a 'Supprimer client' button.

Vous remarquez qu'un bouton « Supprimer client » est apparu à la fin de l'affichage des clients et que chaque ligne est précédée d'un check-box. Si on veut supprimer un ou plusieurs clients il suffit alors de cliquer sur leur checkbox puis de cliquer sur « Supprimer client » ce qui aura pour effet de supprimer de la base tous les clients sélectionnés via les check-box.

Dans le cas où il n'existe aucun client dans la base, lors du clic sur le bouton « afficher les clients », la phrase « Aucune ligne ne répond à votre requête » s'affiche dans un fieldset nommé « Les clients de la base » comme ci-dessous :

Recherche d un client

Nom du client :

Chercher Tout effacer

Afficher tous les clients

Afficher

Les clients de la base

Aucune ligne ne répond à votre requête

On peut également rechercher des clients par nom. Pour cela, il suffit de saisir ce nom dans le champ « Nom du client » du fieldset « Recherche d'un client ». On clique ensuite sur le bouton « Chercher ». Par exemple si l'on cherche les clients qui s'appellent « Anne » alors on aura le rendu suivant :

Les clients de la base

☐ Num CL 3 : Anne Michel ne le 1960-10-02 joignable sur le 0677284

☐ Num CL 54 : Anne Roux ne le 1972-03-30 joignable sur le 06050485

Supprimer client

On remarque que la aussi un bouton « supprimer client » est apparu ainsi que des check-box dans chaque ligne d'affichage. Ceci permettra de faire des suppressions, comme nous l'avons déjà illustré précédemment. Notons que si l'on recherche une personne qui n'existe pas dans la base (par exemple Mr Patrick) alors on aura l'affichage de la phrase « Aucune ligne ne répond à votre requête »

Avant d'arriver à une telle présentation, nous allons passer par plusieurs étapes :

Première partie du travail

Nous allons d'abord concevoir notre interface graphique sans CSS sur une table simplifiée « clientsimple » contenant uniquement un identifiant et un nom. Pour cela :

- Créez une BDD nommé ex1

- Lui ajouter une table « clientsimple » avec : un champ « id » numérique, clé primaire en auto incrémente et un champ « nom » de type texte.

- Créez le formulaire composé d'une zone de texte pour le nom et de deux boutons : un submit dont le nom visuel est « Ajouter client » et un reset dont le nom visuel est « Tout effacer ». Bien entendu le champ id (étant en auto-increment) n'a pas besoin d'être saisi dans le formulaire. Ajoutez à votre formulaire un fieldset « Ajouter client » pour obtenir alors une interface très simple de la forme :

Ajouter un client

Nom :

Ajouter client Tout effacer

- Programmez en Php le bouton submit pour qu'il enregistre dans la table clientsimple chaque ajout de nouveau client saisi dans la zone de texte ci dessus.
- Ajoutez le client « durand » et allez voir dans phpmyadmin si l'élément a bien été inséré dans la table clientsimple.
- Créez le formulaire composé uniquement d'un bouton submit dont le nom visuel est « Afficher les clients ». Ajoutez à votre formulaire un fieldset « Recherche de tous les clients » pour obtenir alors une interface très simple de la forme :

- Programmez en Php le bouton submit pour qu'il affiche le contenu de la table clientsimple. On doit alors avoir un rendu de la forme :

On note que le numéro affiché après la phrase « Client numéro » correspond à l'id du client. La chaine qui est après les « : » est le nom du client. Modifiez votre script pour qu'il vous affiche la phrase « aucune ligne ne répond à votre requête » si aucun client n'est présent dans la table clientsimple.

-On souhaite maintenant que :

- Un fieldset « Liste des clients » s'affiche avec à l'intérieur la liste des clients.
- Chaque ligne soit précédée d'une check-box
- Les noms des clients apparaissent dans des zones de texte.
- Un bouton « Supprimer client » s'affiche à la fin de l'affichage des clients

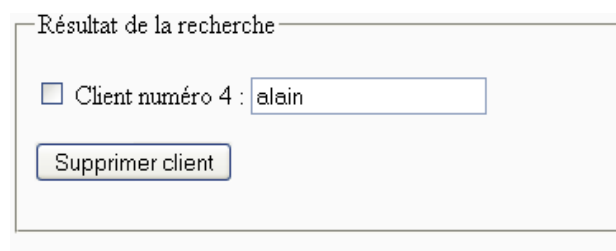
On doit alors obtenir une interface de la forme :

-Pour empêcher les modifications des zones de textes nous allons ajouter des « readonly="readonly" » à l'intérieur des balises concernées.

- Pour supprimer un ou plusieurs clients, on affiche la liste des clients, puis on active la check-box du client ou des clients que l'on souhaite supprimer, puis on clique sur le bouton « Supprimer client ». Programmez ce script en Php puis testez le.

Indication : pour connaître quels sont les clients qui ont été cochés il suffit de nommer les check-box avec l'id du client correspondant puis parcourir le tableau \$_POST pour voir ce qui a été posté...

- Nous allons maintenant faire un travail similaire pour la recherche d'un client avec son nom. Pour cela : créez le formulaire composé d'une zone de texte pour le nom du client et d'un bouton submit dont le nom visuel est « Chercher client », et enfin un bouton reset dont le nom visuel est « Tout effacer ». Ajoutez à votre formulaire un fieldset « Recherche d'un client » pour obtenir alors une interface totalement similaire à celle de l'affichage de tous les clients.



- Si aucune personne de notre table ne correspond à notre recherche alors on affiche le message « aucune ligne ne correspond à votre requête ».

- Implémentez le bouton « Supprimer client » qui fonctionne exactement de la même manière que celui implémenté dans le cas d'un affichage complet de la liste des clients. On suppose qu'il peut exister plusieurs personnes avec le même nom. On ne supprime donc que la personne dont on a coché le check-box et non pas toute les personnes qui ont le même nom que celle de la personne recherchée.

Deuxième partie du travail

-Il est temps de faire évoluer notre table clientsimple. Nous allons alors lui ajouter 3 champs : un prénom de type texte, une date de naissance de type date et un numéro de téléphone de type numérique. Modifier votre page pour intégrer tous ces changements. Notons que pour afficher le résultat d'une recherche ou l'ensemble des clients, les zone de textes contiendront des messages de la forme :

durand dominique né le 1977-01-31 joignable sur le 0234652365

Troisième partie du travail

Nous allons mettre au propre notre code. Pour cela, on passe par une architecture MVC similaire à celle donnée en cours pour l'exemple du forum. On rappelle qu'un fichier qui contient que du Php ne doit jamais être fermé par balise ?>. On va organiser notre arborescence de la manière suivante :

- un contrôleur frontal site.php
- un répertoire « controleur » contenant le fichier contrôleur.php qui contient les fonctions suivantes :

ctlAccueil() qui est appelée par le contrôleur frontal si le client vient d'arriver sur la page. Cette fonction appelle la fonction afficheraccueil() de la vue

ctlAfficherClient() qui est appelée par le contrôleur frontal si le bouton « afficher les client » est posté. Cette fonction charge dans une variable \$client le résultat de la fonction chercherTousLesClients() du modèle puis appelle la fonction afficherClient() de la vue en lui donnant en paramètre \$client.

ctlAjouterClient(\$nom,\$prenom,\$date,\$tel) qui est appelée par le contrôleur frontal si le bouton « ajouter client » est posté. Les paramètres correspondent aux informations client qui ont été postés. Cette fonction teste si un des paramètres est vide et dans ce cas renvoie une exception avec un message « un des champs est invalide » sinon elle appelle la fonction ajouterClient(\$nom,\$prenom,\$date,\$tel) du modèle puis la fonction afficheraccueil() de la vue.

ctlChercherClient(\$nom) qui est appelée par le contrôleur frontal si le bouton « chercher client » est posté. Le paramètre représente le nom du client posté. Si le paramètre est vide elle renvoie une exception avec un message « Le champs nom est vide » sinon elle charge dans une variable \$client le résultat de la fonction chercherNomClient(\$nom) du modèle puis appelle la fonction afficherClient() de la vue en lui donnant en paramètre la variable \$client.

CtlSupprimerClient() qui est appelée par le contrôleur frontal si le bouton « supprimer client » est posté. Elle parcourt le tableau \$_POST via foreach(\$_POST as \$key => \$val) et appelle pour chaque itération la fonction supprimerClient(\$key) du modèle. A la fin de la boucle elle appelle la fonction afficherAccueil() de la vue.

CtlErreur(\$erreur) qui est appelée par le contrôleur frontal dans le catch(Exception \$e). Elle appelle la fonction afficherErreur() de la vue en lui donnant en paramètre \$erreur qui correspond au message d'erreur.

- un répertoire « modele » contenant le fichier connect.php qui contient les constantes de connexion ainsi qu'un fichier modele.php qui contient les fonctions suivantes :

getConnect() qui crée une variable \$connexion qui contient un nouvel objet PDO de connexion à notre BDD, affecte une valeur à \$connexion->setAttribute et \$connexion->query puis renvoie l'objet \$connexion.

chercherTousLesClients() qui appelle getConnect() puis lance une requête qui cherche tous les client de la base et renvoie le résultat dans un tableau via fetchall avant de fermer le curseur.

chercherNomClient(\$nom) qui appelle getConnect et lance une requête qui cherche les client dont le nom est \$nom puis renvoie le résultat dans un tableau via fetchall avant de fermer le curseur.

ajouterClient(\$nom,\$prenom,\$date,\$tel) qui appelle getConnect et lance une requête qui ajoute le client donné en paramètre dans la BDD

supprimerClient(\$id) qui appelle getConnect et lance une requête qui supprime le client dont l'id est donné en paramètre.

- un répertoire « vue » qui contient un répertoire « image » pour les images du CSS, un répertoire « style » pour les feuilles de style CSS, et enfin deux fichier gabarit.php et vue.php

Le fichier gabarit.php contient toute la partie html et une unique partie Php composé de <?php echo \$contenuAffichage; ?> . Cette variable \$contenuAffichage sera rempli dans le fichier vue.php qui contient deux fonctions :

afficherClient(\$client) qui construit le contenu de la variable \$contenuAffichage en utilisant le tableau \$client donné en paramètre. Ce tableau contient le résultat du fetchall des fonctions du modèle de recherche de client ou d'affichage de tous les clients. A la fin de cette fonction on appelle le gabarit via require_once('gabarit.php');

afficherAccueil() qui met la chaîne vide dans \$contenuAffichage puis appelle le gabarit via require_once('gabarit.php');

afficherErreur(\$erreur) qui construit le contenu de la variable \$contenuAffichage en utilisant le message d'erreur \$erreur. Elle ajoute également dans la variable \$contenuAffichage le code Html d'un lien qui renvoie vers le contrôleur frontale. A la fin on appelle le gabarit via require_once('gabarit.php');

Quatrième partie du travail

-On souhaite maintenant avoir un CSS correspondant exactement à l'affichage illustré au tout début de cette série d'exercices. On utilisera alors le fichier CSS de l'exercice 1 de la série de CSS (avec une résolution de 1366x766) et on adapte le fichier HTML pour qu'il corresponde à l'affichage souhaité.