CONSIGNES, SÉRIE 3 : LA CLASSE CLIENT, L'INSCRIPTION ET L'AUTHENTIFICATION

Cette série de consignes va vous amener à la programmation de l'inscription du client, autre élément essentiel de votre site. La méthode proposée utilise largement les variables de **session,** augmentant le confort de l'utilisateur, en lui épargnant, à la suite d'une erreur, d'entrer à nouveau les données déjà saisies. On présente ici une technique efficace et répandue de validation côté serveur. Elle demande du soin dans la programmation mais vous aidera à mieux comprendre les traitements côté serveur.

LA TABLE CLIENT

Avant de commencer le code, ouvrez votre BD avec **phpMyAdmin (420.cstj.qc.ca/MySql)** et créez une nouvelle table pour vos **Clients** avec tous les champs nécessaires (voir l'énoncé du TP);

Insérez manuellement deux clients en prenant soin de fournir des **valeurs uniques** dans le champ du nom d'usager (le champ "nom d'usager" n'est pas le champ "nom", ni le champ "prénom"). Forcez la BD à refuser deux noms d'usagers identiques en cliquant sur l'option Unique:

Le mot de passe sera en clair pour le moment, on pourra éventuellement l'encoder. Prévoyez donc un VARCHAR**, 32**. (la méthode **md5** donne toujours un champ de 32 caractères… on réserve l'espace tout de suite).

LA STRUCTURE D'UN PROGRAMME AVEC VALIDATION CÔTÉ SERVEUR

Avant d'aller plus loin, prenez le temps de bien comprendre la structure de programme qui vous est proposée. Une fois assimilé, le modèle est facile à coder.

Supposons qu'on a, dans le **<form>** de la page d'inscription, un bouton **submit** défini ainsi :
<input type='submit' name='valider' value='Valider'/>.

Le contenu de ce formulaire est transmis par la méthode **POST**. Un test **if (isset($_POST['valider']))** nous permettra de savoir si cette page s'affiche pour une première fois ou suite à un clic sur le bouton **Valider**.

Le pseudocode du programme:

en-tête HTML

$valide = vrai; //(On assume que tout est correct au départ)
si isset($_POST['valider']) //(on a cliqué sur le bouton Submit)
    valider la valeur saisie. //(elle est dans $_POST[**'***ceChamp***'**])
    si elle est correcte,
        afficher l'étiquette en normal,
    sinon
        afficher l'étiquette pour signaler une erreur //(en rouge par exemple)
        $valide=faux; //(Une faute est détectée $valide ne sera plus jamais vrai)
    afficher la valeur courante du champ //(elle est dans $_POST[**'***ceChamp***'**])
    Répéter pour chacun des champs // (**PAS UNE BOUCLE** *mais une série de traitements semblables*)
sinon afficher des champs vides

(après avoir traité tous les champs du formulaire)

si isset(**$_POST['valider']) ET $valide**
    passez au programme de confirmation

ÉTAPE 1 : LA CLASSE CLIENT

Les variables membres de la classe correspondront aux données de la table des clients et elles peuvent être **public ou private** je vous laisse le choix**.** Les puristes de l'OO les mettront **private** et créeront des fonctions accesseurs (get et set).

Une fois qu'un dossier d'un client est créé (ou qu'un ancien client s'est authentifié), il demeurera actif tout au long de sa session d'achat par l'entremise des variables de SESSION, tout comme on procédait avec le Panier.

LE CONSTRUCTEUR

Les données de construction sont assez nombreuses et elles proviennent soit de la base de données, ou du formulaire html. On pourra utiliser la méthode de passage des paramètres dans un tableau, tel qu'utilisée pour le constructeur de la classe Produit :

```
$this->noClient = $tableau['noClient'];
$this->nomClient = $tableau['nomClient'];
$this->adresse = $tableau['adresse'];
$this->province = $tableau['province'];
$this->codePostal = $tableau['codePostal'];
...
```

Voyez les lignes suivantes:

```
foreach ($tableau as $cle => $valeur)
   $this->$cle = $valeur;
```

Elles ont exactement le même effet que la série d'affectations présentée plus haut. Voici les explications :

L'instruction **foreach** balaie un tableau. Elle nous évite d'avoir à connaître le nombre d'éléments du tableau pour faire une boucle **for** ordinaire parce qu'elle gère cet aspect automatiquement. Sa syntaxe avec « **=>** » est plus appropriée pour les tableaux associatifs. Comme les indices ne sont pas des nombres mais des valeurs associatives, elle permet de balayer tout le tableau en retournant le nom de l'indice (**$cle** ici) tout autant que la valeur de l'élément indicé (**$valeur**).

Dans le cas présent, parce qu'il nous arrivera d'instancier le client à partir d'un tableau vide quand on traite un nouveau client, seule la méthode du **foreach** sera acceptable. Utilisez-la plutôt que la méthode qui énumère toutes les variables membres.

Règle de bonne programmation : pour que la méthode fonctionne correctement, il sera important de donner à vos champs HTML le même nom que ce qui leur correspond dans la BD, question d'avoir une équivalence directe entre **$ligne['noClient']** et **$_POST['noClient']** par exemple.

LES AUTRES MÉTHODES

Ajoutez à votre classe une méthode pour afficher le nom du client, une pour obtenir son mot de passe (getMotDePasse()…) et une autre pour en changer la valeur (setMotDePasse()…) qui serviront quand le client voudra changer son mot de passe.

LES PROGRAMMES

Avec cette préparation et la compréhension du précédent sujet, vous pouvez commencer à coder les programmes suivants :

- **inscription.php** : formulaire de saisie des données et **validation complète côté serveur**;
- **confirmation.php** : afficher à l'utilisateur les données saisies validées et demander de confirmer le tout. L'utilisateur peut retourner à sa page d'inscription s'il détecte une erreur;
- **creationDossier.php** : Réception de la réponse du serveur quant à la création du compte. Ça peut être un succès ou un message indiquant que le nom d'usager existait déjà en BD. Dans ce cas, il retourne à **inscription.php**.

Chacun de ces programmes devra commencer par le code spécifique aux sessions : **session_start();**

ÉTAPE 2 : LE PROGRAMME INSCRIPTION.PHP

Un client doit se **créer un dossier** pour pouvoir acheter. Par la suite, on lui permettra de **modifier** son dossier. Plutôt que de vous faire créer une page pour chacune de ces deux fonctions, les consignes vont vous diriger vers la création d'un programme unique qui combinera ces deux aspects. Pour ce travail, cette méthode vous est obligatoire. Vous l'apprécierez grandement quand vous serez rendu à coder la mise à jour du dossier de l'utilisateur.

Selon le contexte, le programme **inscription.php** sera donc appelé de trois manières :

pour créer un nouveau compte utilisateur — les champs de saisie seront tous vides;

1. pour poursuivre la création d'un compte utilisateur sur lequel des problèmes de validation ont été détectés— les champs de saisie contiendront ce qu'on y avait mis dans la même session de travail;
2. pour modifier un dossier existant — les champs de saisie contiendront les valeurs stockées dans la base de données.

Pour tenir compte de ces trois possibilités, **en début du programme, avant l'affichage du premier champ**, mettez une structure logique s'inspirant du pseudo code suivant:

```
si (on arrive par le bouton Valider)
{
        // On soumet les informations d'inscription d'un nouveau client au serveur
        On transfère dans un tableau le contenu pertinent du $_POST
        On instancie un nouveau client en fournissant ce tableau au constructeur
```

```
}
sinon si (le client est déjà authentifié)
{
        // il y a un contenu dans la variable S_SESSION['client']
        On instancie un nouveau client en se servant de la variable S_SESSION['client']
}
else
{
        // Situation initiale : un nouveau client désire s'inscrire
        On instancie un nouveau client en fournissant au constructeur un tableau vide
}
```

Suite à ce prélude, on est certain qu'on a un objet **Client** qui correspond à la situation :

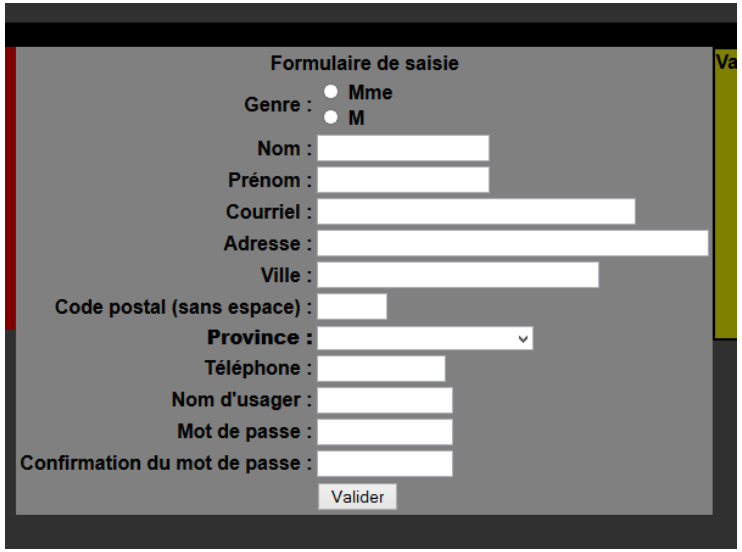
- non vide si c'est un dossier en train d'être validé.
- non vide si c'est un dossier déjà existant
- vide si c'est une création de dossier.

PARTIE 1 : LA PAGE DE L'INSCRIPTION, LE CODAGE DE BASE

On passe maintenant à l'affichage de la fiche d'inscription. Dans un premier temps, **travaillez sur deux ou trois champs**, assurez-vous que tout fonctionne, puis généralisez pour l'ensemble des champs. Vous traiterez les exceptions à mesure qu'elles se présenteront.

Vous implantez ici le pseudocode présenté sur la page 2.

Créez un formulaire en PHP. Le paramètre **action** de la balise **<form>** appelle **le programme lui-même** par la méthode **POST**.



Vous vous aiderez en faisant en sorte que l'attribut "name" de la balise input et celui de l'attribut de la classe **Client** soient identiques. Ainsi, au moment requis, les valeurs déjà saisies apparaîtront pour éviter de les taper à nouveau et vous permettra de coder plus efficacement.

Après avoir codé quelques champs, vous verrez qu'il y a de nombreuses répétitions. Vous voudrez probablement coder une fonction avec quelques paramètres, ce qui raccourcira considérablement votre travail.

Une fois le code de base écrit, sans les validations, testez la présentation de votre page. Lors d'un premier affichage, les variables de session étant vides, rien n'apparaîtra dans les champs. Forcez la variable **$valide** à **false** avant le test final (celui qui ressemble à : **if (isset($_POST['valider']) && $valide==true)** et vous

devriez constater que votre page s'affiche à nouveau avec les champs remplis des valeurs tapées avant le clic sur le bouton **Valider.**

PARTIE 2, LA VALIDATION CÔTÉ SERVEUR

Le canevas est créé, vous devriez avoir écrit une fonction d'affichage des champs qui sont de même type (par exemple, tous les champs texte).

LE CHAMP PROVINCE ET L'UTILISATION D'UNE LISTE DÉROULANTE.

Pour le champ **Province**, plutôt que de faire une validation, vous allez utiliser une liste déroulante. Une liste allège la validation en ne fournissant que des valeurs correctes tout en simplifiant la vie de vos clients.

Lors de l'inscription, aucune valeur n'est affectée au paramètre et la liste affiche simplement le premier élément. Au moment de la mise à jour de la fiche des clients, vous devez faire en sorte que cette liste affiche la province déjà saisie pour chacun des clients.

```
function afficherProvince($provParam)
{
       Construire un tableau des provinces
       Afficher l'étiquette "Province"
       echo "<select name='province'>"
       pour chaque province du tableau des provinces
              echo "<option value='$cleDuTableau'";
              si ($valDuTableau == $provParam)
                 echo "selected";
              echo $valeurDuTableau . "</option>";
       echo "</select>";
}
```

L'appel de la fonction ressemblera à: afficherProvince($client->province);

LES AUTRES CHAMPS

Pour le champ **Genre**, un bouton radio ou une liste déroulante éviteront également une validation.

Pour tout le reste, il faut prévoir deux types d'affichage des valeurs: un mode d'affichage correct, lorsque la donnée est valide et un mode d'affichage erroné lorsque la validation a détecté une erreur. L'affichage en noir lorsque tout est correct et en rouge lorsqu'une erreur est présente est une possibilité simple.

Les **expressions régulières** peuvent vous simplifier considérablement la vie dans certaines validations.

Si tout est valide, il vous reste à transférer le contenu de l'objet **Client** vers une variable de session: **$_SESSION['client'] =** $client.

Finalement, **une fois tous les champs validés**, utilisez la balise <meta> pour automatiser le passage à la page de confirmation.

```
echo "<meta http-equiv='Refresh' content='0;url=confirmation.php' />";
```

Testez avec des erreurs dans chacun de vos champs, assurez-vous que tout fonctionne correctement, c'est-à-dire que toute erreur est signalée, que la page demeure la même tant qu'il reste au moins une erreur et que, dès que tout est correct, on passe à la page de confirmation.

PARTIE 3 : LA PAGE DE CONFIRMATION

Cette page permet à l'utilisateur de vérifier ses données une dernière fois avant de les transmettre au serveur. S'il détecte une faute il peut retourner à la page d'inscription.

Le bouton **Confirmer** soumet un formulaire qui appelle **creationDossier.php**. Le reste des données n'a pas à être dans ce formulaire puisque les valeurs sont dans des variables de session.

Vous n'avez donc qu'à transférer les données du tableau **$_SESSION** vers une instance de la classe client et les afficher. Votre programme débutera donc par

```
$client = $_SESSION['client'];
```

Testez le retour à la page d'inscription, toutes les données saisies devraient y être.

PARTIE 4: LA PAGE DE CRÉATION DU DOSSIER

Il faut ajouter deux méthodes à la classe BDService: une pour l'insertion d'un nouveau client et une pour la mise à jour de sa fiche

INSERTION EN BD

La méthode d'insertion devra lancer une exception ("throw") si une erreur se produit lors du "insert". L'erreur de violation d'unicité (nom d'usager déjà en BD), se traite différemment des autres erreurs, car il faut en aviser l'utilisateur et l'application doit continuer de fonctionner. On peut vérifier le code d'erreur d'une requête par l'attribut **errno** de l'objet MySQLi l'ayant exécutée. Le code de l'erreur doublon est 1062.

MISE À JOUR EN BD

La méthode de mise à jour (update) retournera le nombre de lignes modifiées par la requête. Si une erreur se produit elle lancera une exception.

Il arrive que des programmeurs envoient un énoncé UPDATE ou pire, DELETE, sans mettre de condition WHERE dans l'énoncé SQL. Ceci cause une mise à jour de tous les enregistrements de la table ou encore la suppression complète de ceux-ci. On peut ajouter la protection suivante:

```
        ...
        if (!stripos($sql, 'WHERE'))
           throw new Exception ('Danger!!! pas de WHERE dans l\'énoncé UPDATE');
    ...
```

LA PAGE DE CRÉATION DU DOSSIER

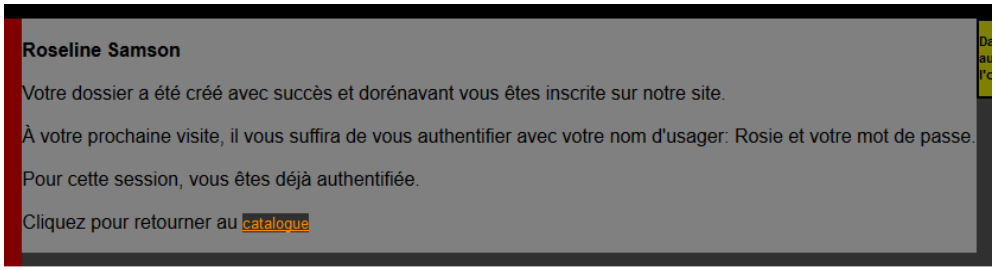
Instanciez un **Client** avec le contenu de la variable de session :

```
$client = $_SESSION['client'];
```

Exécutez l'insertion dans la BD en invoquant la méthode d'insertion de votre objet BDService, invocation enrobée d'un **try{} catch(){}.**

Si le catch(){} attrape l'exception "**doublon**" retournez à la page **inscription.php** avec un paramètre **GET** (dans l'url) spécialement conçu à cet effet. Le retour à la page d'inscription doit **obligatoirement** (je vous l'impose) passer par la fonction **header()** de php. Si le catch attrape une autre exception, le programme se termine. Attention: faites suivre l'appel de header() d'un **exit();** Cela semble superflu mais c'est nécessaire. Ça garantit que la suite du code ne sera pas exécutée. Logiquement, le code passe à une autre page mais la documentation de PHP insiste qu'il faut jouer ainsi de sécurité en portant à la fois bretelles et ceinture.

Si tout semble correct, assurez-vous que le dossier a bel et bien été inséré dans la BD. Complétez par du code qui affiche quelque chose de similaire à la page ci-dessous:



Créez une nouvelle variable de session qui deviendra très importante : **`$_SESSION['authentification']`**et mettez-y le nom d'usager comme valeur. C'est cette variable qui vous indiquera par la suite si le client est authentifié ou non.

Revenez ensuite dans le programme **inscription.php** et ajoutez les lignes de code nécessaires pour afficher un message d'erreur si jamais la page a été appelé à cause de l'erreur de **doublon**.

L'indication d'erreur étant dans le get (envoyé par la fonction **header()**), cette page peut servir à afficher le formulaire de saisie des données ET à afficher un message d'erreur.

À ce moment-ci, soit le nom d'usager n'existe pas dans la BD et le programme crée le nouveau client, soit il existe déjà et le contrôle revient à la page d'inscription qui affiche, outre l'erreur de doublon, les valeurs déjà saisies.

DES CAS SPÉCIAUX QUI DOIVENT ÊTRE TRAITÉS POUR UN SITE PROFESSIONNEL :

LA FONCTION HTMLENTITIES()

Vos champs de formulaires sont une porte d'entrée pour l'injection HTML (xss). Pour vous protéger utilisez la fonction php **htmlEntities()**. Cette fonction converti tous les caractères spéciaux HTML sous leur forme texte : '**<**' devient '**<**' '**>**' devient '**>**' etc. Par défaut, htmlEntities ne convertit pas les apostrophes et guillements. Pour ce faire on doit lui fournir le paramètre ENT_QUOTES.

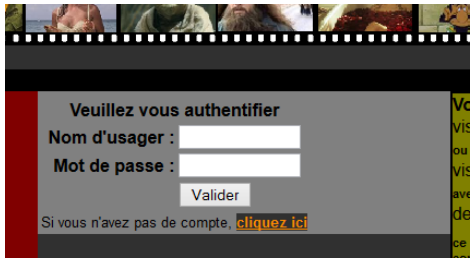
LA METHODE BDService::NEUTRALISE()

Autre cas: imaginez un nom de famille **Prud'homme** ou encore **L'Heureux.** Au moment de l'insertion vous frapperiez une erreur. En effet, le nom Prud'homme donne un énoncé SQL du genre VALUES('Prud'homme',…). Les apostrophes ne balançant pas; le programme plante.

La méthode mysqli::**real_escape_string()** résout ce problème. Avant de construire vos énoncés d'insertion vous convertissez les informations à insérer avec cette méthode. Real_escape_string() étant une méthode de la classe mysqli, on peut ajouter une méthode simple à notre classe BDService, nommée neutralise comme suit:

```
public function neutralise($info) {
     return $this->BD_Interne->real_escape_string($info);  }
```

ÉTAPE 3 : LA PAGE D'AUTHENTIFICATION



Vous devez offrir au client la possibilité de s'identifier. Soit vous le faites dans une page du genre de celle qui est à gauche, soit vous l'incluez dans la page du catalogue. Les consignes décrivent la première approche.

Cette page est simple : elle contient un formulaire de deux champs dont la balise **action** rappelle le programme lui-même, par exemple, **authentification.php**. La méthode sera **POST**. Ajoutez à votre page un lien vers la page d'inscription pour les nouveaux clients.

Commencez par ouvrir une session (session_start();). La suite va être située dans un *gros* **if (isset($_POST['valider']))** dont l'accolade fermante sera située juste avant le HTML de la page.

Dans cet énoncé **if**, vous allez

- Ouvrir la BD et lire (**select**) un dossier complet d'utilisateur de manière à pouvoir instancier un objet **Client**. Les critères de l'énoncé "**select**" seront reçus par le **$_POST.**
- Si le client existe et que le mot de passe est le bon
  - vous instancier le client. Mettez ensuite ce client dans la variable de session, comme vous l'avez déjà fait par **$_SESSION['client']= $client;**
  - Il vous reste à « noter » que ce client est authentifié. Vous utilisez la même variable que lors de la création du dossier (possiblement **`$_SESSION['authentification']=$_POST['usager']`**.
  - Enfin, pour le moment, vous retournez l'utilisateur à la page d'accueil (votre catalogue).
- Sinon, retournez à la page d'authentification (par header() ou autre) mais avec une indication de l'erreur dans le **GET.**
  - Dans la partie « formulaire » de votre programme, ajoutez, au bon endroit, le code **if (isset($_GET['message']))**… pour afficher le message d'erreur.

ÉTAPE 4: GESTION DE LA CIRCULATION

Pour compléter votre programme d'authentification, il vous reste à le modifier un peu pour qu'il fasse une bonne « gestion de la circulation ».

Les cas possibles sont les suivants :

- le client clique sur le **lien d'authentification** : on lui permet de s'identifier et on le ramène au catalogue;
- le client clique sur le lien **Mon dossier** : s'il est authentifié, on l'amène à sa fiche sinon, on lui demande de s'authentifier **puis on l'amène à sa fiche**;
- le client clique sur **Commander** : s'il est authentifié, on passe à la page de confirmation du contenu de son panier sinon, on lui demande de s'authentifier **puis on l'amène à la page de confirmation du contenu du panier**. Ce dernier cas n'étant pas encore programmé, on pourra coder le lien mais on devra le vérifier plus tard.

Voici comment gérer tout ça.

Dans une partie commune à toutes les pages (en-tête, colonne de gauche ou autre) mettez un lien (Mon Dossier) sur lequel le client cliquera pour accéder à sa fiche. Ce lien appellera l'url suivante: **authentification.php?prov=dossier**

Si le client est déjà authentifié, on veut l'envoyer directement à sa fiche. Sinon, on « note » qu'il est arrivé par le lien **Mon dossier** avant de lui demander de s'authentifier.

```
if (isset($_SESSION['authentification'])) {
  header("location:inscription.php");
  exit();
}
if (isset($_GET['provenance']))  //Si on arrive par le lien Mon dossier
    $_SESSION['provenance'] = 'dossier';
```

Voilà pour le début du programme.

Ensuite, **après** le code qui gère l'authentification, au lieu de l'énoncé qui renvoie directement au catalogue, ajoutez un aiguillage sur la valeur de $_SESSION['provenance'.]. Ainsi, si l'on provient de "Mon dossier" on va vers inscription.php (avec header() suivi d'un exit()) , si on provient de "Commander le panier" (pas encore développé) on ira vers commande.php. Par défaut on retourne au catalogue (index.php). n'oubliez pas d'effacer la valeur de $_SESSION['provenance'] une fois l'aiguillage accompli.

ÉTAPE 5 : LA MISE À JOUR DE LA FICHE DU CLIENT

Un client authentifié, pourra mettre les infos de sa fiche à jour **sauf son nom d'usager qui doit demeurer le même et son mot de passe qu'on traitera dans une autre page.**

Vous avez déjà un formulaire et quelques programmes qui permettent la saisie et l'enregistrement des données du client. Vous utiliserez ce même formulaire et les programmes reliés en les modifiant plutôt que de "réinventer la roue".

MODIFICATIONS À APPORTER À INSCRIPTION.PHP

Le client authentifié peut modifier toutes ses informations de la fiche **sauf son nom d'usager**. Le mot de passe sera modifiable, mais dans une page à part. Si le client est authentifié, le nom d'usager apparaît en **lecture seule** et le mot de passe, lui, n'est pas affiché.

Voici une suggestion : dans le cas de la mise à jour du dossier, affichez les champs non désirés du **<form>** en **type='hidden'** plutôt que **text** ou **password**. Ils ne seront pas visibles mais seront tout de même dans le **POST**. Le cas du champ de confirmation doit être traité à part.

MODIFICATIONS À APPORTER À CREATIONDOSSIER.PHP

Si le client n'est pas authentifié, on crée son dossier — ce que vous faites déjà — mais s'il existe, il ne faut pas le créer mais **mettre à jour son dossier.** Au niveau BD ce sera un UPDATE plutôt qu'un INSERT.

La syntaxe SQL pour la mise à jour est la suivante : **UPDATE *table* SET *champ1* = '*valeur1*', *champ2* = '*valeur2*',… WHERE *nomUsager='valeur'.*** Construisez l'énoncé qui mettra à jour **tous** les champs. N'oubliez surtout pas la condition **WHERE** ...

Dans un bloc **try… catch** appelez la mise à jour ainsi :

```
$maj = $bd->MiseAJour($strSQL);
```

Suite au bloc, donc s'il n'y a pas eu d'erreur de SQL, vous pouvez écrire quelque chose d'équivalent à ces énoncés :

```
if ($maj == 0)
     $message = 'Vous n'avez fait aucun changement';
```

et récupérer le texte du message plus loin dans la partie HTML pour informer le client. S'il n'y a pas d'erreur, informez-le que son dossier a été mis à jour.

ÉTAPE 6 : LE CHANGEMENT DU MOT DE PASSE

Cette page exige du client qu'il tape le mot de passe actuel, son nouveau mot de passe et sa confirmation. Pour chacun des trois champs, vous devez évidemment utiliser ***type='password'*** pour ne pas afficher les caractères réels.

On a encore affaire à un cas de programme qui va s'appeler lui-même. Une partie construit le formulaire et fait les validations possibles **côté client** en **JavaScript:**

- obligation de taper des données dans tous les champs
- le nouveau mot de passe différent de l'ancien
- minimum de 5 caractères
- confirmation égale au nouveau mot de passe.

Prévoyez l'affichage d'un message d'erreur si le programme de validation côté serveur détecte une erreur.

Sur le clic du bouton **Valider**, vous tentez une lecture de la table client avec **une condition WHERE basée sur le nom d'usager et le mot de passe**. Si la requête ne retourne rien, c'est que le client n'a pas tapé le bon mot de passe. Rappelez la page en transmettant un message d'erreur. Autrement, mettez à jour son mot de passe (UPDATE *Clients* SET *MotPasse*= etc.) et affichez lui un message de confirmation du changement.

ÉTAPE 7 : LA VALIDATION ASYNCHRONE DE L'UNICITÉ DE L'USAGER

Une fois les fonctionnalités centrales établies, on peut augmenter l'ergonomie de l'utilisateur s'inscrivant à votre site. En effet, un utilisateur qui, par inadvertance, choisit un nom d'usager déjà existant dans la BD ne sera mis au courant de cette erreur que lors de la soumission du formulaire d'inscription. Or il est possible de l'aviser de cet erreur bien avant.

Pour ce faire vous introduirez ici une validation AJAX qui permettra de savoir, avant la soumission du formulaire d'inscription, si le nom d'usager est déjà existant en BD. Pour déployer cette validation ergonomique on créera deux fonctions Javascript et une variable booléenne Javascript.

La première fonction JS sera déclenchée lors de l'évènement **onBlur** du champ **usager**. Elle fera une requête asynchrone au serveur par l'entremise d'un mini script, **VerifierUniciteUsager.php**, à qui elle fournira par GET la valeur souhaitée par l'usager. VerifierUniciteUsager.php interrogera la BD et retournera (echo) un statut binaire du genre : 'Existant' ou 'Inexistant'. Si l'usager est inexistant une variable booléenne JS sera initialisée à true.

La deuxième fonction sera déclenchée lors de la soumission du formulaire (onSubmit) et elle retournera la valeur de la variable booléenne JS. Le formulaire sera soumis seulement si cette variable est vraie.

Notez que cette validation asynchrone côté client n'existe que pour l'ergonomie de l'utilisateur, **elle ne remplace aucunement la validation finale** côté serveur de l'unicité de l'usager (voir étape 2, partie 4, Insertion en BD).