Université de Cergy-Pontoise

Maîtrise d'informatique

2004 - 2005

Jean-Philippe Aumasson <jpnossa@free.fr>

Jean-Baptiste Langlois <jean-baptiste.langlois@wanadoo.fr>

DOCUMENT TECHNIQUE

9 janvier 2005

REALISATION : MODELISATION OBJET AVEC UML, PROGRAMMATION DE L'APPLICATION CLIENT/SERVEUR EN PHP, BASES DE DONNES RELATIONELLES MYSQL.

SUJET : GESTION D'UN ETABLISSEMENT SCOLAIRE DE TYPE UNIVERSITE.

Résumé

Ce document présente les aspects techniques de l'application, et est destiné à quiconque souhaite modifier ou étudier le programme.

Table des matières

Pré-requis

Des connaissances basiques de PHP, HTML et SQL sont nécessaires, ainsi que des notions de conception orientée objet. Pour l'installation et l'utilisation de l'application en local on pourra installer un serveur Apache, MySQL et phpMyAdmin (ce dernier pas indispensable, mais recommandé). Les procédures d'installation et de configuration de ces programmes ne sont pas détaillées ici, vous trouverez ces informations sur leurs sites respectifs, ou par une simple recherche Google. Pour installer l'application sur un serveur distant, un espace d'une dizaine de mégaoctets fournis par un fournisseur d'accès suffit très largement. Un compte MySQL doit aussi être activé, et éventuellement un accès à phpMyAdmin pour l'interaction directe avec la base de donnée (les deux allant souvent de paire). On n'aura pas nécessairement besoin de modifier la base directement dans phpMyAdmin si tout fonctionne bien, mais une consultation facilitera la compréhension du système, et tout ne fonctionnera sans doute pas toujours bien.

1 Présentation

On considèrera que le lecteur connaît les fonctions PHP utilisées, ou est en mesure de consulter la documentation. Une lecture attentive des sources permet de comprendre facilement le fonctionnement du programme, étant donné la simplicité des fonctions utilisées, et la redondance de portions de codes quasi-identiques. Nous détaillons dans ce document les points les plus délicats du programme, la consultation du document d'analyse et du *user's guide* peuvent aussi être utiles à la bonne compréhension du fonctionnement de l'application.

UMS est une application simple de gestion du système éducatif d'une université, sous forme de site web. Les pages sont des fichiers PHP, l'IHM correspond à l'inclusion de code HTML. Chaque utilisateur devra s'authentifier, et disposera d'un niveau de privilèges (au nombre de trois). Un seul compte bénéficie des

privilèges maximum, le niveau 3, dans ce document on nommera le propriétaire de ce compte super-user, root, ou responsable.

L'application a été conçue en suivant le modèle objet, certains fichiers PHP correspondent alors à des classes du système. Reportez-vous au document d'analyse pour la présentation et justification du modèle.

Les informations sont stockées dans une base de données MySQL, on utilise donc des requêtes SQL pour écrire et lire dans la base.

En plus des fonctions de gestion effective du système scolaire, on pourra personnaliser l'interface, et quelques mesures sécuritaires ont été mises en oeuvre.

Après lecture de ce document et de quelques fichiers sources vous devriez être en mesure de modifier l'application, y rajouter des fonctions, ou éventuellement corriger des erreurs de conception ou de programmation. Bonne chance !

2 Installation de l'application

La procédure est détaillée dans le User's Guide, nous n'y reviendrons pas ici.

3 Interface HTML

L'IHM de l'application est totalement conçue en langage HTML, en utilisant les balises les plus simples, notamment :

- < **table** > : pour les tableaux.
- < **form** > : pour les formulaires, l'ajout d'un étudiant par exemple.
- < **input** > , dont l'attribut type pourra être *text* pour un champ de saisie, *password* pour saisir un mot de passe en masquant les caractères, *submit* pour un bouton de "soumission" (au sens de requête et non d'allégance...), *hidden* pour des éléments cachés.

On devra veiller à ne pas inclure de code HTML dans une portion de code destinée à du code PHP, la section devra être fermée à l'aide de la balise `?>`.

4 Fichiers PHP

Ces fichiers se trouvent dans le répertoire *ums/,* sont commentés, vous pourrez générer des pages des documentation avec des logiciels comme phpdocumentor (*http ://www.phpdoc.org/*).

4.1 Classes

Fichiers : *career.php, course.php, diploma.php, linkedlist.php, person.php, privileges.php, sid.php, student.php, teacher.php, tid.php.*

Chaque classe du système correspond à une classe du modèle objet conçu, exceptée la classe "utilitaire" *linkedlist*, réalisée par nécessité pour la gestion de l'ordre des diplômes. Mais elle n'entre pas dans le cadre de la conception orientée objet du système universitaire, et ne se trouve donc pas dans le diagramme de classes. Cette classe permet d'instancier une liste chaînée, éventuellement ré-utilisable, n'étant pas propre aux diplômes.

4.2 Pages

4.2.1 Affichage de l'interface

Un point délicat du programme : l'affichage d'une fenêtre de l'application n'est pas simplement une page PHP décrivant tout le contenu graphique affiché à l'écran, mais un ensemble de fichiers et certaines variables communes. Nous allons décrire le principe en développant le rôle de chaque fichier.

index2.php

C'est le conteneur le plus large, qui contient les balises `<html>` et `</html>`. Ce fichier inclut tous les fichiers de classes, pour avoir accès à leurs méthodes. Il vérifie aussi si le *timeout* a été atteint, et si l'authentification est valide :

```
if ($oldtime!="") { // test le timeout
 if ((time()-$oldtime)>$timeout) {
  die("</head><body><h1>Session timeout exceeded ($timeout seconds)</h1>") ;
 }
}
$oldtime=time() ; // si test ok met à jour l'heure
// teste la validité du password
if (($plg->getPassword()!=$passw) || ($passw=="")) {
 echo "</head><body><h2><p align=\"center\">Incorrect password.<br>Please <a href=index
}
```

Si ces tests ont été passés, il affecte la variable *$priv* au niveau de privilège de l'utilisateur, applique la personnalisation de l'environnement (langue et *skin*), en incluant le fichier de langue et en choisissant le fichier *css*, et inclut le fichier *index3.php* :

```
else { // si les tests de timeout et pwd sont ok
 $priv=$plg->getLevel() ; // $priv <- le niveau de droits de l'user
 // on inclut le lien vers le css dans la page HTML
```

```
 echo "<link REL=\"stylesheet\" TYPE=\"text/css\" href=\"skin/" ;
if ($newskin !="") {
  echo $newskin ;
 } else {
  echo $plg->getSkin() ;
 }
 echo "\"></head><body>" ; // fin de l'entête HTML
 // si la langue a été modifié on inclut le bon fichier
 if ($newlang=="") {
  include "lang/lang-".$plg->getLang().".php" ;
 } else { // sinon on conserve le même
  include "lang/lang-".$newlang.".php" ;
 } // et on inclut index3.php
 include "index3.php" ;
}
```

index3.php

C'est dans ce fichier qu'on insèrera la barre supérieure de menu :

```
include "menu.php" ;
```

Puis, si la variable *$page* est non nulle, on affichera la page *$page.php*, sinon on testera successivement, dans cet ordre, la nullité des variables *$student*, *$teacher*, *$config*, *$valid*, *$pref*, correspondant chacune à la page de l'élément selectionné respectivement dans les menus *Students*, *Teachers*, *Studies*, *Validation*, et *Preferences*. A la première occurence de variable non nulle, appellons là *$myvar*, on incluera la page *$myvar.php*.
On établit donc une hiérarchie arbitraire entre les menus, au cas où l'utilisateur aurait l'idée de sélectionner deux fonctions à la fois. On notera qu'on teste en tout premier la variable *$page*, qui se trouve donc avec la plus grande priorité.

Un exemple d'affectation indirecte de la variable *$page*, dans le fichier *remstudent.php* :

```
<form
action="index2.php" method="POST"><p class="success"><input type="hidden" value="remstu
name="page"><input type="hidden" name="presentid" value="<? echo $chosenid ;?>"><?
$chosenid="" ;
include "hiddenstuff.php" ;
?><input type="submit" name="btn_rm" value="<?
echo $TXT_BTNREMOVE ;?>"></p></form>
```

Ainsi quand on appuieras sur le bouton d'intitulé *$TXT_BTNREMOVE*, la variable *$page* sera affectée à la valeur "remstudent2", la page *remstudent2.php* sera donc chargée, via *index2.php* et *index3.php*.

4.2.2 Pages de saisie et validation

Pour l'entrée de donnée, on aura généralement deux fichiers, *page.php* et *page2.php*, le premier pour la saisie des informations, le second pour leur écriture effective dans la base, après validation dans *page.php*. En cas d'erreur, elle sera signalée sur *page2.php*, et on aura un lien pour revenir à la saisie, sans modification des champs remplis.

Un exemple simple, l'ajout d'une filière, avec les fichiers *addcareer.php* et *addcareer2.php*. Tout d'abord le premier, auquel on a ajouté des commentaires :

```
if (($config=="") || ($login=="") || ($passw=="")) {
 die("<h1>HTTP Error 401 : Authorization Required</h1>") ;
} // on teste si l'utilisateur a le droit de visualiser cette page
?>
// le titre de la page, affiché en haut, selon le selecteur title
<h2 class="title"><? echo $TXT_ADDCAREER ; ?></h2>
<p></p><p></p><p // pour laisser un espace suffisant
align="center"> // centrer l'affichage
 // début du formulaire, traité par index2.php
<form action="index2.php" method="POST">
// élément caché : affectation de $page à addcareer2
<input type="hidden" value="addcareer2" name="page">
// début d'un tableau
<p align="center"><table width="400" border="0"
valign="top"><tr><td width="50%" border="0">
// le label indiquant la nature de l'entrée à saisir
<p class="entryl"><? echo $TXT_TITLE ; ?> :</p></td><td width="50%"
border="0"><p class="entryr">
// le champ, un input de type text, l'entrée sera la variable $title
<input type="text" name="title" value=""
size="25"></p></td></tr></table> // fin du tableau
<br><br><?
include "hiddenstuff.php" ;
?>
// le bouton de validation, la saisie sera traitée par $page.php
<input type="submit" name="button" value="<? echo $TXT_BTNSUBMIT ; ?>"></p></form>
```

la variable *$page* ayant été affectée à "addcareer2", on atteindra la page *addcareer2.php* en cliquant sur le bouton de soumission. Le code de cette page :

```
if (($page=="") || ($login=="") || ($passw=="")) {
 die("<h1>HTTP Error 401 : Authorization Required</h1>") ;
} // test le droit d'accès à la page
?>
<?
// instancie une nouvelle filière, d'identificateur 1 (valeur indifférente)
```

```
$ca = new Career(1) ;
// on ajoute une career, lui affectant son vrai ID (et plus 1)
$ca->addCareer($title) ;
// message de succès de l'opération
?><h2 class="success"><? echo $TXT_CAREERADDED ; ?></h2><p></p><br><br><p
class="success"><? echo $TXT_THECAREER ; ?>  ;<b><? echo $title ; ?></b>  ;<?
echo $TXT_HASBEENADDED ; ?>
```

4.3 Langues

Le choix de la langue s'effectue par l'utilisateur dans le menu *Preferences*, et correspond au choix d'un fichier *lang-<2 lettres>.php*, dans le répertoire *ums/lang/*. Les pages affichées sont *chglang.php* et *chglang2.php*.

Les fichiers de langue contiennent les affectations des variables de la forme *$TXT_< chaîne en majuscules, en anglais>*, utilisées comme éléments textuels de l'interface. La langue par défaut est l'anglais, on utilise donc le fichier lang-en.php, commençant par les lignes :

Pour ajouter une langue on ajoutera un fichier dans ce répertoire nommé comme spécifié ci-dessus, et définissant les mêmes variables que les autres fichiers. Dans le cas de l'ajout d'un élément textuel à l'interface on devra évidemment ajouter la variable correspondante dans tous les fichiers de langue. Si l'administrateur n'est pas polyglotte, il pourra utiliser les services de traduction disponibles sur Internet.

Les langues suivantes sont disponibles : anglais (par défaut), espagnol, français, japonais (encodage JIS), et portuguais.

4.4 Apparences (*skins*)

L'utilisateur peut changer le *skin* de l'application, dans le menu Preferences, un extrait du code de menu.php s'y reportant :

```
<td align="center" border="0"><select
name="pref"><option selected><?
/* Pref Menu */
echo $TXT_PREFMENU." -" ;
?><option
value="chgpwd"><? echo $TXT_CHGPW ; ?><option
value="chgskin"><? echo $TXT_CHGSKIN ; ?><option
value="chglang"><? echo $TXT_CHGLANG ;
?></select></td>
```

Le choix d'un *skin* correspond au choix d'un fichier de style *css*. Ces fichiers sont dans le répertoire *ums/skin/*. Les pages de l'interface utilisées sont *chgskin.php* et *chgskin2.php*.

On ne détaillera pas le contenu de ces fichiers, une lecture du fichier fournit une description de chaque sélecteur. On utilise respectivement les suffixes de sélecteurs 'l', 'c' et 'r' pour gauche, centre, et droite.

Pour ajouter un *skin* il suffit d'ajouter un fichier *css* dans ce répertoire.

4.5 Autres fichiers

4.5.1 functions.php

Ce fichier contient les définitions de fonctions propres au système, et non à une classe particulière. Pour les utiliser dans une page donnée, on devra inclure la ligne suivante :

```
include "functions.php" ;
```

On a notamment des fonctions pour déterminer la période courante, le nom d'une ville à partir de son code postal, le nombre de diplômes et de filières de l'université.

4.5.2 hiddenstuff.php

Contient des variables d'environnement du système cachées, comme l'identificateur de l'utilisateur, son mot de passe, l'heure d'accès à la page (pour déterminer le timeout). On l'incluera donc pour toute utilisation ou modification de ces variables (par exemple pour la modification du mot de passe, voir *chgpwd2.php*).

4.5.3 config.php

Ce fichier est crée à l'installation, et contient les informations de configuration du système, il est de la forme :

```
<?
 $servername="sql.free.fr" ;
 $username="jpnossa" ;
 $password="XXXXXX" ;
 $database="jpnossa" ;
 $timeout="300" ;
?>
```

Ce fichier est crée à l'installation de l'application, à partir des informations entrées par l'utilisateur. La signification de chaque variable :

- `$servername` : le nom du serveur, localhost pour une utilisation locale.
- `$username` : le nom du propriétaire de la base.

– `$password` : le mot de passe associé au *$username*.
– `$database` : le nom de la base.
– `$timeout` : la durée limite de veille du programme sans utilisation. Si cette durée est dépassée l'utilisateur devra se réidentifier.

4.5.4 log.php

Ce fichier contient une seule fonction, *log_ action*, appellée pour ajouter un évènement au fichier de log du jour. Reportez-vous à la section Sécurité pour le fonctionnement du système de logs, et aux commentaires du fichier source.

5 Base de données

5.1 Tables et clés

On se reportera à l'analyse pour les détails sur la normalisation des tables et les dépendances fonctionnelles.

5.2 Requêtes

On utilise les fonctions PHP d'interaction avec une base de données, leur nombre est très limité, et leur utilisation varie peu d'une interaction à une autre. On détaillera l'exemple de la méthode *getIDFromName*, de la classe *Person*, qui, à partir du nom et du prénom (surname et first name) d'un utilisateur, renvoie son identificateur.

```
function getIDFromName($fname,$sname) {
 include "config.php" ;
 $dblink=mysql_connect($servername,$username,$password) ;
 if ( !$dblink)
 {
  die("Connection error :".mysql_error()) ;
 }
 mysql_select_db($database,$dblink) ;
 $query=mysql_query("SELECT ID FROM ‘Person‘ WHERE sname=\"".$sname."\" AND   fname=\""
 $result=mysql_fetch_array($query) ;
 mysql_close($dblink) ;
 $this->nip=$result[0] ;
 return $this->nip ;
}
```

La première ligne est la déclaration de la fonction, spécifiant son nom et ses paramètres, ici `$fname` et `$sname`, respectivement pour le prénom et le nom.

La ligne suivante, l'inclusion du fichier *config.php*, est indispensable à toute requête. En effet, ce fichier config.php est très court, mais fondamental ; il contient les informations de connexion, son rôle est spécifié plus haut dans ce document dans la partie *Autres fichiers.*

Mais revenons à notre exemple.

```
$dblink=mysql_connect($servername,$username,$password) ;
```

La fonction `mysql_connect( hôte, utilisateur, mot de passe)` établit une connexion vers la base MySQL hébergée sur la machine spécifiée selon les arguments spécifiés, et renvoie l'identifiant de connexion, affecté à la variable `$dblink`. Si la tentative de connexion échoue, la valeur `false` est renvoyée.

```
if (!$dblink)
 {
  die("Connection error :".mysql_error()) ;
 }
```

Si la connexion a échoué on renvoie une erreur.

```
mysql_select_db($database,$dblink) ;
```

On sélectionne la base de données à partir de son nom et de l'identifiant de connexion.

```
$query=mysql_query("SELECT ID FROM ‘Person‘ WHERE sname=\"".$sname."\" AND   fname=\"".
```

La requête envoyée à la base de données, où on doit être particulièrement attentif à la syntaxe et à la ponctuation, notamment à l'insertion de variables PHP.

```
 $result=mysql_fetch_array($query) ;
```

Renvoie le tableau de tous les champs du résultat.

```
 mysql_close($dblink) ;
```

Ferme la connexion.

```
 $this->nip=$result[0] ;
```

Affecte la variable de classe `$nip` au premier champ du résultat dans `$result`.

```
 return $this->nip ;
}
```

Retourne la valeur et termine la fonction.

6 Sécurité

Nous nous sommes limités au strict nécessaire, sans entrer dans la prévention et la protection d'attaques de pirates, en formulant l'hypothèse que les étudiants de l'université ne sont pas des génies en informatique.

6.1 Identification

C'est un point particulièrement important, dans la mesure où un utilisateur devra s'identifier à chaque connexion. L'identification sert à reconnaître l'utilisateur comme une entité particulière et lui afficher l'interface associée à cette entité : les éléments de personnalisation, et surtout un ensemble de privilèges. On utilise la procédure la plus courante, en obligeant l'utilisateur à saisir un identificateur et un mot de passe.

L'identificateur d'un utilisateur sera 0 pour le root, et l'ID des autres utilisateurs : 1XXXXXXX pour un étudiant, 2XXXXXXX pour un enseignant.
Le mot de passe sera crée pour le responsable par l'adminisatrateur, à l'installation de l'application. Etudiants et enseignants se voient attribué un mot de passe à huit caractères à leur ajout dans la base, les lignes de codes correspondant sont respectivement dans *addstudent2.php* et *addteacher2.php*, par exemple, pour le *addstudent2.php* :

```
$pass=crypt($sname.$fname,$stu->getID()) ;
$pass=substr($pass,0,8) ;
$pri->setPassword($pass) ;
```

La fonction *crypt* renvoie une chaîne correspondant au chiffrement de la chaîne concaténée du nom et du prénom de l'utilisateur, avec son ID comme "sel". On prend ensuite les huits premiers caractères de cette chaîne pour créer le mot de passe.
Pour modifier la taille du mot de passe, par exemple l'étendre à 16 caractères, on modifiera la deuxième ligne en :

```
$pass=substr($pass,0,16) ;
```

Cependant seuls les nouveaux inscrits auront un mot de passe à seize caractères, les comptes existants conserveront leur mot de passe.

A l'identification (page *index.php*), on recherche si le couple login/mot de passe entré correspond à une entrée de la base de données, ce traitement étant effectué dans *index2.php*, la vérification des variables d'authentification étant effectuée à chaque chargement de page :

```
// $login et $passw sont le login et password entrés dans index.php
```

```
$stu = new Person($login) ; // instanciation d'une personne de login $login
$plg = new Privileges($stu) ; // instanciation des privilèges associés
(...)
// si le mot de passe de cette personne et différent de $passw...
// ... ou si $passw est nul
if (($plg->getPassword() !=$passw) || ($passw=="")) {
// alors on affiche un message d'erreur
 echo "</head><body><h2><p align=\"center\">Incorrect password.<br>Please <a  href=inde
}
```

6.2 Droits d'accès

Les droits d'accès correspondent au champ *level* de la table *Privileges*, qui est soit 1, 2, ou 3, par ordre croissant de privilège, des droits très restreints de l'étudiant aux droits *root*.

On doit notamment différencier les utilisateurs pour l'affichage des fonctions dans les menus, un exemple de différenciation, dans le fichier *menu.php* :

```
if ($priv==3) {
?><option
 value="chprivtea"><? echo $TXT_CHGPRIV;
}
if ($priv==2) {
?><option
 value="listhimself2"><? echo $TXT_INFOLIST;
}
if ($priv==3) {
?><option
 value="remteacher"><? echo $TXT_REMTEACHER;
}
```

Ce qui correspond littéralement à tenir compte du code entre accolade pour chaque type de type de privilège, et concrètement à afficher les options `$TXT_CHGPRIV` (changement de privilèges) et `$TXT_REMTEACHER` (supprimer un enseignant) pour le super-user, et `$TXT_INFOLIST` (informations personnelles) pour un enseignant. On rappelle qu'une chaîne de caractère précédées du signe dollar ('$') constitue une variable PHP, et qu'en l'occurence les variables *$TXT_<suite de majuscules>* sont définies dans les fichiers de langue, dans *ums/lang/*.

6.3 Prévention

Pour tenter d'éviter une utilisation frauduleuse quelques précautions simples ont été mises en oeuvre.

6.3.1 Limitation des droits

La justification des choix de limitations de droits est définie dans le document d'analyse, nous nous interesserons ici à leur mise en place au niveau du code du programme.

Les droits correspondent à un niveau de privilèges, le champ level de la table Privileges.

Le principe est relativement simple : avant l'affichage d'une fonction soumise à condition sur les droits de l'utilisateur, on teste le niveau de privilège de l'utilisateur. Un exemple, l'affichage des fonctions dans le menu Studies, avec un extrait du code du fichier *menu.php* :

```
<td align="center" border="0"><select
name="config"><option selected><?
/* Config Menu */
echo $TXT_CONFIGMENU." -";?>
<option
value="listcourses"><? echo $TXT_LISTCOURSES;?>
<?
if ((getPeriod($rightnow)=="Re") && ($priv==3))
{
?> <option
value="addcourse"><? echo $TXT_ADDCOURSE;?><option
value="remcourse"><? echo $TXT_REMCOURSE;?><option
value="adddiploma"><? echo $TXT_ADDDIPLOMA;?><option
value="remdiploma"><? echo $TXT_REMDIPLOMA;?><option
value="addcareer"><? echo $TXT_ADDCAREER;?><option
value="remcareer"><? echo $TXT_REMCAREER;
}
?>
</select></td>
```

Le test correspond à la ligne :

```
if ((getPeriod($rightnow)=="Re") && ($priv==3))
```

Ici on conditionne l'affichage des options du menu d'une part en fonction du niveau de privilège :

$priv==3 <=> l'utilisateur possède le niveau 3 de privilège, soit le maximum, les droits root,

d'autre part en fonction de la période courante :

getPeriod($rightnow)=="Re" <=> la période courante est "Re", soit la rentrée (Start of year).

6.3.2 Timeout

Un timeout a été mise en place pour déconnecter l'utilisateur après une certaine durée sans utiliser le programme. Nous avons arbitrairement fixé le délai à dix minutes, soit 600 secondes. Dans le fichier *index2.php*, appellé à chaque action sur le système on aura :

```
if ($oldtime!="") {
 if ((time()-$oldtime)>$timeout) {
  die("</head><body><h1>Session timeout exceeded ($timeout seconds)</h1>") ;
 }
}
```

Ce code affichera alors le message d'erreur à toute tentative d'action sur le système ultérieure au délai spécifié. La variable *$timeout*, fixant la durée limite, en seconde, est affectée dans le fichier *config.php*.

6.3.3 Erreur 401

Par convention l'erreur 401 intervient quand un utilisateur tente d'accéder à une page ou un fichier dont il ne possède pas l'autorisation nécessaire. Pour notre programme, ce cas se présenterait si un étudiant, ne s'étant pas authentifié, tenterait d'accéder directement à une page en entrant son chemin d'accès, par exemple :

http ://jpnossa.free.fr/files/maitriseInfo/Objet/projet_04_05/ums/addmark.php

Par exemple dans l'espoir de modifier ses notes...mais s'affichera le message :

HTTP Error 401 : Authorization Required

Pour restreindre l'accès à une page de cette façon, on insère par exemple le code suivant au début de la page *addstudent.php* :

```
if (($student=="") || ($login=="") || ($passw=="")) {
 die("<h1>HTTP Error 401 : Authorization Required</h1>") ;
}
```

En effet, si la page a correctement été accédée, à partir du menu, la variable *$student* correspondrait à l'entrée en question, les variables *$login* et *$passw* sont affectées respectivement à l'identificateur et au mot de passe de l'utilisateur lors de son authentification, une valeur vide signifie donc que l'identification n'a pas eu lieu.

Le principe est analogue pour toute autre page, on posera toujours la condition sur *$login* et *$passw*, et sur d'autres variables en fonction de la page.

6.4 Logs

On conserve des logs des actions les plus importantes, pour pouvoir savoir *a posteriori* qui a fait quoi et quand, dans le répertoire *logs*, les fichiers étant de la forme *log_ <date>*.

La création d'un log est réalisée à l'aide la fonction *log_ action(<user>, <action>)*, qui sauvegarde les informations suivantes :

- L'horaire de l'action, à la seconde près.
- L'adresse IP du visiteur, correspondant à la clé *REMOTE_ ADDR* du tableau *$_ SERVER*.
- L'action réalisée : chaîne de caractère spécifiée sur la page de l'action, par exemple pour l'ajout d'un étudiant on précisera l'ID de l'étudiant ajouté :

```
log_action($login,"student added : ".$stu->getID());
```

...qui ajoutera dans le fichier *log_ 07-01-05* la ligne :

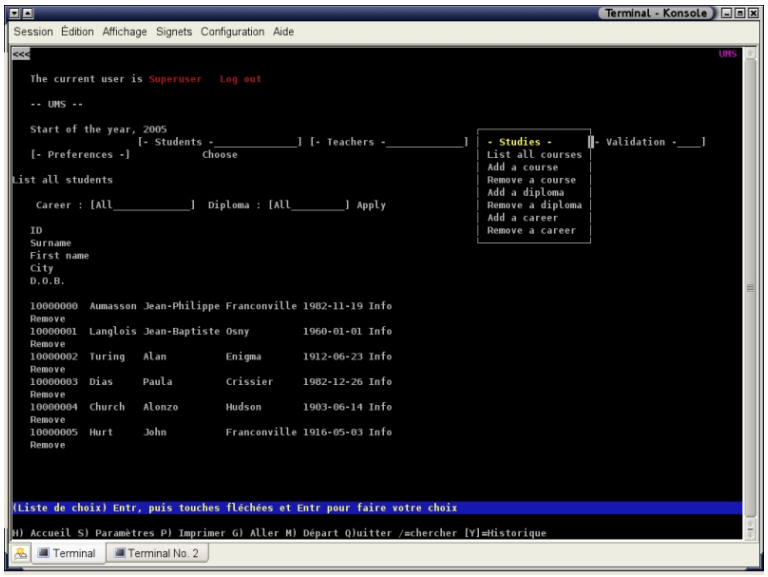
07-01-05 04 :32 :09 // IP :82.229.88.17 // USER : 0 // ACTION : student added : 10000013

...si l'utilisateur d'identificateur 0 (le *super-user*) a ajouté l'étudiant d'identificateur 10000013 à 4h 32min 9sec le 7 janvier 2005.

7 Portabilité

UMS est utilisable sur tous les naviguateurs usuels, sur toute plateforme. Ont notamment été testés Firefox, Konqueror, Safari, Internet Explorer. De plus la non-utilisation de technologies particulière (*javascript...*) permet l'affichage sur des naviguateurs plus basiques, tel Dillo ou même le butineur en mode texte lynx! Dillo ne gérant pas les styles css, les skins ne seront pas gérés, et l'application aura l'apparence du skin *purity* simplifié (pas d'alignements...).

Un exemple d'affichage d'UMS sous lynx :



8 Licence

Le programme est sous licence GPL, dont l'intitulé se trouve dans le fichier COPYING.