par Guillaume Affringue (http://guillaume-affringue.developpez.com)

Date de publication : 20/09/2006

Dernière mise à jour : 11/01/2007

Ce document est une présentation et une aide au déploiement de l'Aedituus, un script de gestion d'espace membre écrit en PHP5 et orienté objet.

I - Présentation générale

L'aedituus est un script permettant de gérer un espace membre. Il est constitué d'un minimum d'interface (inscription, connexion, récupération du mot de passe), le reste est à la charge du programmeur (profil, droits, administration des membres). Cependant plusieurs fonctions permettent de faciliter l'écriture de ces interfaces.

Le coeur est constitué de 2 classes pour le support de l'utilisateur. Nous verrons qu'il est possible de les utiliser indépendemment et ce qu'elles apportent, puis nous verrons comment les utiliser avec l'interface de l'aedituus, c'est-à-dire les fichiers de connexion, inscription et récupération du mot de passe.

Il comporte beaucoup de fonctionnalités lièes à la sécurité. Ces fonctionnalités ont été présentées sur les forums de developpez.com et dans plusieurs articles/cours présents sur developpez.

Le projet avait été initié par Sub0, qui avait publié sur developpez.com deux versions d'un espace membre.

Je l'ai tout d'abord adapté pour mes besoins, puis transformé pour finalement reécrire une version très différente, principalement par l'utilisation de PHP5 et des exceptions, et par certains choix, comme le détachement des sessions natives de PHP.

Une démo de l'aedituus est visible **ici**

Les sources sont **ici**

Le topic sur l'espace membre de Sub0 est **ici** Et **ici**, un topic ouvert par Sub0 pour débattre de la sécurité d'un espace membre en PHP

II - Architecture

A venir

III - Support de l'utilisateur

Pour gérer la session, sont obligatoires les fichiers

- user.class.php
- session.class.php (ou sessionphp.class.php)
- interface.php (définition de l'interface iSession)
- mysql.class.php
- dbmanager.class.php
- commun.php
- fonction.php

et les tables

- membres
- sessions
- session_vars
- membre_vars
- champs
- champs_value

III-A - Classe Csession.class.php

L'explication ci-dessous ne compte que pour session.class.php (le fichier sessionphp.class.php sera traité plus tard)

La class CSession est une implémentation du support d'une session utilisateur alternative à celle de PHP. Elle n'utilise pas ce qui est présent dans PHP (c'est à dire les session_start()...).

Cette classe n'est pas utilisée directement, elle est la classe mère de la classe CUser, qui correspond, elle, au support de l'utilisateur en général. L'héritage s'explique par le fait que le support d'un utilisateur comprend le support de sa session. De plus, la classe CSession n'est ainsi pas instanciée directement, ce qui évite aux données "sensibles" tels que l'id de session d'être accessible depuis le reste du script. Il y a 3 données membres de CSession qui sont accessibles depuis la classe CUser (et que de cette classe, jamais du reste du script)

Données membres protected de la classe CSession

- $session_logged_in qui indique à la classe CUser que l'utilisateur actuel s'est connecté,
- $session_user_id qui correspond à l'Identifiant du membre dans la table membres
- $session_ip qui est à la fois une donnée de session et une donnée de l'utilisateur.

Cette classe utilise un cookie pour identifier l'utilisateur durant sa navigation. Elle n'utilise pas du tout le passage de l'identifiant de session par URL, car ceci est considéré comme une potentielle faille de sécurité. Elle utilise un 2ième identifiant de session, qu'on appelle Token, qui change à chaque page. Ce dispositif rend très difficile le vol de session, mais présente un défaut : si le hacker vole la session alors que l'utilisateur vient de finir sa navigation, sans se déconnecter, le token n'empêchera pas le vol de la session.

La session est initialisé dès la première page, que le visiteur soit connecté et reconnu comme membre, ou non connecté reconnu comme visiteur.

Enfin, pour finir, une surcharge des opérateurs d'affectation (setter) et les accesseurs (getter) permet une utilisation de la classe comme support de stockage des variables de session.

En PHP, ces deux rôles sont confondus dans l'appellation session, mais en réalité, elles sont composées de la partie liaison entre le client et le serveur pour le maintient de la « connexion », et du stockage des nombreuses variables qu'on souhaite attribuer à cette session.

On aurait pu ici externaliser cette fonction de stockage dans une autre classe, mais ça aurait compliqué encore plus.

Pour utiliser le stockage de variable, il faut procéder de cette façon.

- Déclaration d'un objet de CUser (CSession ne doit pas être utilisée directement)
- Pour affecter :$user->ma_variable = 'Hello world';
- Pour récupérer : echo $user->ma_variable ; // Hello world

Ces variables sont stockées dans la table session_vars. La classe redéfinit les opérateurs isset et unset pour ce type de variable, mais ceci est transparent pour le développeur.

Ne pas utiliser le tableau $_SESSION

liste des méthodes publiques

```
interface iSession
{
    public function session_begin();
    public function connectUser( $id_user );
    public function deconnectUser();
    public function isConnected();
    public function identifieUser( $id_user );
    public function getSessionVars();
    public function getIP();
};
```

III-B - Classe Csessionphp.class.php

La classe CsessionPHP implémente l'interface iSession ci-dessus et utilise les sessions PHP, La classe est plus simple à comprendre mais moins souple et contrôlable, car liée à l'implémentation des sessions dans PHP. Elle permet de garder la compatibilité avec d'éventuels scripts existants utilisant les sessions PHP. L'utilisation du tableau $_SESSION est donc possible, mais $user->ma_variable reste utilisable. Les 2 étant reliés, par exemple

Les 2 méthodes sont liées

```
$_SESSION['hello'] = 'world';
echo $user->hello;    // world
```

et inversement.

III-C - Classe CUser.class.php

La classe Cuser permet au développeur de manipuler les données de l'utilisateur. Elle est le lien entre la gestion de la session utilisateur et l'interface (le site dans sa globalité).

Elle permet d'accéder aux données du membre

- Donnée de session à travers les méthodes de la classe Csession
- Données permanentes provenant de la table membre_vars

Ces dernières ont 2 moyens d'être créées, d'une part lors de l'inscription, si le développeur a ajouté des champs personnalisés (cf.page inscription), d'autre part directement dans le code à l'aide des fonctions :

- setPermanentVar permettant de créer/modifier une donnée permanente
- getPermanentVar permettant de la récupérer
- delPermanentVar permettant de l'effacer.

Il est possible grâce à ces fonctions de modifier les données saisies par l'utilisateur sur le formulaire d'inscription. Pour cette raison, attention au choix des noms lors de la création de celle-ci par les fonctions afin de ne pas écraser les données saisies par l'utilisateur, si ce n'est pas explicitement votre but.

Ces données sont permanentes, ce ne sont pas des variables de session, elles survivent après une déconnexion et sont de nouveau accessibles lors d'une nouvelle connexion.

III-D - Class CUserList.class.php

III-E - Exemple d'utilisation simple

Nous allons voir ici comment créer rapidement un espace membre exploitable en utilisant les classes présentées précédemment, session.class.php, user.class.php et cuserlist.class.php (+ quelques dépendances)

Tout d'abord, dans un répertoire indépendant, créer un répertoire include et placer y les fichiers

Liste des fichiers à mettre dans le répertoire include

- dbmanager.class.php
- mysql.class.php
- interface.php
- session.class.php
- user.class.php
- userlist.class.php
- fonction.php

dbmanager.class.php et mysql.class.php permettent la connexion à une base de donnée MySQL

interface.php, session.class.php, user.class.php et userlist.class.php sont les classes dont nous avons parlées ci-dessus et sont au coeur de la gestion de l'utilisateur

fonction.php contient quelques fonctions simples

Créons maintenant à la racine 4 fichiers common.php, inscript.php, connect.php et index.php

Le fichier common.php contiendra les includes, la définition de constantes, la connexion à la bdd et la récupération en base de la configuration de l'aedituus.

Code du fichier common.php

```
<?php
define ('PIWARE_ROOT', './');
define ('SESS_NAME', 'piware');
define ('SESS_TIME', 7200);

require_once 'include/dbmanager.class.php';
require_once 'include/mysql.class.php';
require_once 'include/interface.php';
require_once 'include/session.class.php';
require_once 'include/user.class.php';
require_once 'include/userlist.class.php';
require_once 'include/fonction.php';

$cfg['prefix']          = 'pw_';

// Connexion à la bdd
$db = db::getInstance('Mysql');
$db->setParam( 'localhost', 'root', '', 'test_aedituus');
$db->connect();

// On récupère la config générale
$result = $db->get_results( "SELECT co_key, co_value FROM ".$cfg['prefix']."config" );
foreach ($result as $config)
{
    $cfg[$config->co_key] = $config->co_value;
}
?>
```

La constante PIWARE_ROOT indique le chemin vers la racine et sert à vérifier que les fichiers inclus ne le sont pas par un script externe.

Les constantes SESS_NAME et SESS_TIME définissent respectivement le nom et la durée de la session (en secondes).

III-E-1 - L'inscription

Attaquons maintenant la page d'inscription.

Code de la page d'inscription

```
<?php
require_once 'common.php';
$user = CUser::getInstance();
$user->session_begin();

if ( ! empty($_POST['formSend']))
{
    if ( (empty($_POST['login'])) || (empty($_POST['mdp'])) || ($_POST['mdp'] != $_POST['mdp2']) )
    {
        $msg = 'Erreur de login ou mot de passe';
    }
    else
    {
        $user->login = $_POST['login'];
        $user->mdp = $_POST['mdp'];
        $id_membre = CUserList::addUser($user);

        if ( is_null($id_membre) )
        {
            $msg = 'Echec inscription';
        }
```

Code de la page d'inscription

```
            else
            {
                $msg = 'inscription réussis';
                $user->connectUser( $id_membre );
                $user->setPermanentVar('eye_color', $_POST['eye_color']);
            }
        }
    }


    echo '<html>
    <head></head>
    <body>'.$msg.'<br /><form action="" method="post">
        Pseudo : <input type="text" name="login" value="'.$user->login.'" /><br />
        Mot de passe : <input type="password" name="mdp" /><br />
        Confirmation : <input type="password" name="mdp2" /><br />
        La couleur de vos yeux : <input type="text" name="eye_color" value="" /><br /><br />
        <input type="submit" name="formSend" value="Valider" />
    </form>
    </body>
    </html>';
    ?>
```

Beaucoup de choses à dire ici, nous allons étudier du haut vers le bas.

On commence par inclure notre fichier common.php définit ci-dessus.

Initialisation de l'utilisateur

On récupère l'objet $user courant par la technique du singleton.

On démarre la session par $user->session_begin();

(Toutes les pages du site commenceront de cette manière)

Traitement du formulaire

Nous avons ensuite le traitement du formulaire. On commence par vérifier que les champs ne sont pas vides et que le mot de passe et sa confirmation coïncident.

Si tout est ok, on commence l'enregistrement de notre utilisateur. Pour cela, on rempli notre objet $user avec les attributs login et mdp (il y a aussi email, mais je ne l'ai pas mis ici pour ne pas surcharger).

Une fois notre objet $user rempli, on l'ajoute à la liste des utilisateurs par le service CUserList::addUser auquel on passe notre objet $user en paramètre.

Ce service va enregistrer pour nous l'utilisateur dans la base de donnée et retourne l'identifiant de celui-ci en cas de succès.

La dernière partie est optionnelle : si l'utilisateur est bien enregistré, outre un message l'avertissant, on le connecte directement et on ajoute une variable supplémentaire demandée dans le formulaire.

Enregistrement de variables permanentes

Je vais présenter plus en détail cette dernière fonctionnalité. Il s'agit d'un service d'enregistrement rapide de variables liées à un utilisateur. Vous pouvez définir toutes les variables qui vous semblent utiles, mais **vous ne devez pas utiliser les mots-clés login, email, mdp et instance** qui sont déjà utilisées par la classe CUser.

Ces variables peuvent provenir de formulaire comme ici ou directement depuis le code. Elles s'enregistrent par la méthode $user->setPermanentVar($key, $value), se récupèrent par $user->getPermanentVar($key) et s'effacent par $user->delPermanentVar($key).

Il faut bien comprendre, comme le nom des méthodes l'indiquent, que ces variables sont permanentes et survivent après une déconnexion.

Il est important de noter aussi que l'utilisateur doit être soit connecté ($user->connectUser($id_user)) soit identifié ($user->identifieUser($id_user)) pour utiliser ces 2 méthodes. L'identification permet de reconnaître un utilisateur, mais le laisse en mode **non connecté**.

Affichage du formulaire

Au niveau du formulaire, vous remarquerez que l'attribut value du login contient $user->login. En effet, après envoi du formulaire, la valeur dans le POST est placée dans l'objet $user. Ceci nous permet donc, lors d'un échec à l'inscription, de remplir les champs avec ce qu'on a déjà enregistré. Seul est exclu le mot de passe, car il est préférable qu'il soit ressaisi à chaque tentative.

Nous verrons plus loin que ceci peut être fait avec les champs "optionnels" tel que le eye_color à certaines conditions (nous verrons en même temps une méthode plus simple de les enregistrer).

III-E-2 - La connexion

Notre utilisateur est enregistré en base de donnée et nous allons voir maintenant comment réaliser la page de connexion.

Code de la page de connxion

```
<?php
require_once 'common.php';
$user = CUser::getInstance();
$user->session_begin();

if ( ! empty($_POST['formSend']))
{
    if ( (empty($_POST['login'])) || (empty($_POST['mdp'])) )
    {
        $msg = 'Veuillez renseigner tous les champs';
    }
    else
    {
        $user->login = $_POST['login'];
        $baseUser = CUserList::getUserByLogin( $user->login );
        $md5mdp = md5( CUserList::getUserGDS( $user->login ) . stripslashes($_POST['mdp']) .
 CUserList::getUserGDS($user->login));
        if ($baseUser->mdp != $md5mdp)
        {
            $msg = 'Mauvais mot de passe';
        }
```

Code de la page de connxion

```
            else
            {
                $user->connectUser( $baseUser->getUserId() );
                $msg = 'Vous êtes connecté';
            }
        }
    }

    echo '<html>
    <head></head>
    <body>'.$msg.'<br /><form action="" method="post">
        Pseudo : <input type="text" name="login" value="'.$user->login.'" /><br />
        Mot de passe : <input type="password" name="mdp" /><br />
        <input type="submit" name="formSend" value="Valider" />
    </form>
    </body>
    </html>';
    ?>
```

Comme pour la page d'inscription, je vais expliquer du haut vers le bas.

On commence par inclure notre fichier common.php définit ci-dessus.

Initialisation de l'utilisateur

On récupère l'objet $user courant par la technique du singleton.

On démarre la session par $user->session_begin();

Traitement du formulaire

Nous avons ensuite le traitement du formulaire. On commence par vérifier que les champs ne sont pas vides. Si tout est ok, on commence la procédure de connexion.

On récupère un objet user correspondant au login que le visiteur a saisis. Cet objet nous servira de base pour la comparaison (nous ne comparons ici que les mots de passe, mais on aurait pu comparer les dates de naissance, les emails....). Cet objet est appelé $baseUser, on le récupère par le service CUserList::getUserByLogin ($login).

On compare ensuite les mots de passe (attention, le mot de passe enregistré en base est de la forme MD5(GDS+mdp+GDS)), il faut donc redonner cette forme au mot de passe provenant du formulaire. Si la compraraison est un succès, on connecte l'utilisateur simplement par $user->connectUser($id_user).

Affichage du formulaire

On note, comme pour le formulaire d'inscription, que l'objet user permet de conserver le login saisit durant les tentatives.

III-E-3 - Page protégée

Nous allons voir ici l'exemple typique d'une page ayant un contenu "public" et un contenu "privé" personnalisé au membre connecté.

Exemple de page public/privée typique

```
<?php
require_once 'common.php';
$user = CUser::getInstance();
$user->session_begin();
$userdata = $user->getUserData();

if ( ! $user->isConnected() )
{
    echo 'Vous devez être connecté pour voir cette page, vous pouvez vous connecter par ce lien :
 <a href="./connect.php">Page de connexion</a>';
}
else
{

    echo 'bonjour '.$userdata['login'].', vous avez de jolis yeux '.$userdata['eye_color'].'<br
 />Pour vous déconnecter, c\'est ici : <a href="./connect.php?a=deco">Page de déconnexion</a>';
}
?>
```

On voit apparaître $userdata. Ce tableau contient toutes les propriétés accessibles de l'utilisateur. N'hésitez pas à faire un print_r($userdata). Ce tableau sera principalement rempli par le développeur, car comme vous pouvez le constater, outre le login, on retrouve ici notre variable eye_color définit à l'inscription.

Vous noterez la fonction $user->isConnected qui se passe de commentaires. Vous remarquerez aussi surement qu'il y a un lien vers une page de déconnexion, qui nous n'avons pas vu encore, et que nous ne verrons pas ;-). A vous de chercher ! (un indice : $user->deconnectUser())

III-F - Conclusion de la partie support de l'utilisateur

Voila, vous avez un espace membre. Bon il est très succint, mais il fonctionne. Il vous restera à écrire la déconnexion, une barre de navigation entre inscription/connexion/déconnexion et un brin de finition...

Ou alors, vous continuez la lecture, et découvrirez que tout est déjà fait et prêt à l'usage. Cet exemple permet aussi et surtout de montrer que l'aedituus n'est pas un bloc figé, mais que chaque brique peut être facilement intégrée à un système existant. Il vous suffira de reproduire ce qui a été fait ci-dessus.

IV - Interface entre le support de l'utilisateur et l'utilisateur lui-même

Les scripts d'interface sont constitués des fichiers inscript, connect, passperdu (et gds, mais il sera traité avec le script de connexion). Ces scripts ont besoin du support de l'utilisateur vu ci-dessus. Ils ne peuvent pas être utilisés indépendamment.

L'ensemble des scripts d'interface est fait de façon à ne générer ni cookie, ni session_start(), ni header (par exemple pour les redirections), ni die(), exit()...., bref, aucune fonction qui empêcherait le script de se poursuivre (afin par exemple d'afficher un pied de page). Ceci est possible grâce au mécanisme des exceptions, appliqué ici aux « erreurs utilisateur ». Ainsi, si le mot de passe est incorrect, si l'email est faux.... ont soulèvera une exception, avec un message d'erreur, cette exception est capturée et on affiche un message d'erreur personnalisé grâce à la fonction affiche_message et éventuellement on réaffiche le formulaire, le script ayant soulevé l'exception est interrompu et on passe à la suite.

Principe

Les 3 pages fonctionnent selon le même principe. Nous définissons une fonction qui affichera le fomulaire souhaité grâce au mécanisme des templates. S'il n'y a pas d'envoi de formulaire, nous affichons simplement celui-ci par appel de la fonction. Si le formulaire est envoyé, nous le traitons. durant le traitement nous testons diverses choses, viabilité des données, mot de passes concordants... Suivant chaque résultat de test, il y a 3 choix.

- Le script continue simplement au test suivant en cas de succès.
- Nous levons une exception que nous capturons, puis nous affichons le message grâce à la fonction affiche_message qui prendra la place du formulaire.
- Nous levons une exception que nous capturons, puis nous rappellons la méthode d'affichage du formulaire en lui passant le message d'erreur en paramètre

IV-A - Inscription

Dans le fichier inscript.php, on peut distinguer successivement

- la récupération des POST avec la classe de filtrage (cf 2,3 Sécurité)
- La fonction d'affichage du formulaire
- Le traitement du formulaire si celui-ci a été envoyé.

Parmi les fonctionnalités additionnelle de la page inscription, on trouve

- L'ajout dynamique de champ de formulaire
- Un chiffrage par l'algorythme RSA du mot de passe coté client, déchiffrage coté serveur

IV-A-1 - Champs dynamiques

Liste des colonnes de la table champs

```
ch_nom: nom du champ pour le formulaire (caractères alphanumériques, sans espaces)
ch_texte : Texte affiché à coté du champ de formulaire
ch_type_affichage : R=>radio, T=>text, S=>select, A=>textarea
ch_min_size : taille minimum du texte saisi par l'utilisateur
ch_max_size : taille maximum du texte saisi par l'utilisateur
ch_obligatoire : 1: obligatoire, 0:non obligatoire
ch_ordre : Order d'affichage des champs sur le formulaire d'inscription.
```

Pour les choix, comme les listes select ou les boutons radio, ils faut saisir les différentes valeurs à proposer dans la table champs_value

Liste des colonnes de la table champs_value

```
cv_id_champ : identifiant du champ saisi ci-dessus
cv_nom : le texte à afficher sur la liste select ou à coté du bouton radio
cv_ordre : ordre d'affichage.
```

IV-A-2 - Chiffrage par RSA

IV-B - Connexion

Le script de connexion utilise un GDS pour le transit et le stockage des informations. Ce GDS est créé à l'inscription et est unique à chaque membre. L'unicité êmpèche la constitution d'un dictionnaire global au site. Ce gds est utilisé à chaque connexion pour encoder le mot de p asse coté client. Ainsi, un script AJAX est chargé de chercher un GDS valable au fur et à mesure de la saisie du login sur le formulaire de connexion. Ce script distant est le fichier gds.php.

Pour empêcher les attaques par force brute, un timer est installé, et oblige un intervalle de temps entre 2 connexion. L'interval de temps est configurable dans la table config.

IV-C - Pass perdu

Test la validité dulogin et du mail envoyé par l'utilisateur. Si les informations sont valides, le script génére un mot de passe de 9 caractères alpha numériques et l'envoie par mail

IV-D - Conclusion partie Interface

V - Deploiement

Il est préférable de ne pas essayer de fusionner l'aedituus avec un script existant, mais plutôt de le laisser « à coté ». C'est-à-dire créer une page dédiée à l'affichage des formulaires de l'aedituus.

Le coeur de chaque page du formulaire est écrite dans les fichiers templates (que vous modifierez pour les adapter au design de votre site).

Ainsi, l'intégration se déroule dans cet ordre.

- Création d'une page aedituus.php (appeler la comme vous voulez) au design de votre site, prévoyant un emplacement pour les formulaires de l'aedituus.
- Sur cette page, saisir à l'endroit ou vous souhaitez voir apparaître les formulaires, le code ci-dessous.
- Modification du design des formulaires (fichier template) pour l'adapter au design du site.Tous ces templates se ressemblent beuacoup. Ils ne contiennent que les formulaires. Seul message.tpl ne contient pas de formulaire, mais un cadre, qui s'affichera à la place des formulaires en cas d'erreur ou d'exception soulevée.

Code d'insertion des formulaire

```
If ( ! empty($page))
{
    switch($page)
    {
        case 'connect':
            include 'connect.'.EXT;
        break;
        case 'inscript':
            include 'inscript.'.EXT;
        break;
        case 'passperdu':
            include 'passperdu.'.EXT;
        break;

        default:
            affiche_message('Impossible de trouver la page');
        break;
    }
}
```

Ce qui signifie que vos pages seront accessibles par

- aedituus.php?page=connect
- aedituus.php?page=inscript
- aedituuq.php?page=passperdu

Enfin, en haut de la page, saisissez le code

Code de gestion de la session utilisateur

```
define ('PIWARE_ROOT', './');
require_once PIWARE_ROOT.'include/extension.inc';
require_once PIWARE_ROOT.'include/commun.'.EXT;

// Récupération des $_GET
$page = (isset($_GET['page']) ? $_GET['page'] : null);

// L'objet user, démarrage de la session utilisateur
$user = new CUser();
$user->session_begin();
```

En réglant la constante PIWARE_ROOT en fonction de l'emplacement de ce fichier par rapport à la racine.

Pour ajouter des champs de connexion sur une autre page du site, il suffit de mettre un code du style (seuls les éléments HTML de formulaire sont réellement obligatoires)

Code du formulaire de connexion

```
<form method="post" action="aedituus.php?page=connect" id="connexion-form" name="connexion-form"
 onsubmit="md5hash(this.mdp, this.md5mdp, this.gds);">

Login <input type="text" name="login" id="login" size="20" value="" autocomplete="off">

<div style="border:solid 1px
 black;height:16px;width:120px;background-color:#F0F0FF;font-weight:bold;" id="info_gds">Grain de
 sel</div>

Mot de passe <input type="password" name="mdp" size="20">

<input type="hidden" name="md5mdp" value="">
<input type="hidden" name="gds" id="gds" value="">
<input value="Valider" id="Valider" type="submit" class="bouton">
</form>
```

Ainsi que ceci dans l'entête pour bénéficier de la protection par GDS coté client (non obligatoire)

Code d'insertion du système de récupération du GDS par ajax

```
<script type="text/javascript">
    var _adresseRecherche = "gds.{EXT}"   // l'adresse à interroger pour trouver le GDS
</script>
<script type="text/javascript" src="./js/MD5Hash.js"></script>
<script type="text/javascript" src="./js/getGDS.js"></script>
<script type="text/javascript">
window.onload = function()
{
    initAutoComplete(
        document.getElementById('connexion-form'),
        document.getElementById('login'),
        document.getElementById('Valider'),
        document.getElementById('gds'),
        document.getElementById('info_gds')
    );
};
</script>
```

Pour le moment, les fichiers de l'aedituus sont présents, on a créer une page à part pour l'affichage des formulaires, mais le site lui-même ne possède encore aucune protection.

Pour intégrer l'aedituus dans toutes les pages, il faut mettre tout en haut de chaque page

Code nécessaire au démarrage de la session dans les pages à protéger

```
define ('PIWARE_ROOT', './');
require_once PIWARE_ROOT.'include/extension.inc';
require_once PIWARE_ROOT.'include/commun.'.EXT;

// L'objet user, démarrage de la session utilisateur
$user = new CUser();
$user->session_begin();
```

VI - Principales fonctions

Pour avoir les infos utilisateurs, utiliser

```
$userdata = $user->getUserData();
```

Pour utiliser les sessions, utiliser

```
$user->une_variable = une_valeur;
echo $user->une_variable; //  une_valeur
```

Pour stocker des données permanentes propre à un utilisateur, utiliser

```
$user->setPermanenteVar( 'une_variable', $une_valeur);
echo $user->getPermanentVar( 'une_variable' );
```

Pour empecher l'acces à une page, utiliser

```
if ( ! $user->isConnected())
{
    header('./aedituus.php?page=connect');    // lien vers page de connexion
}
```