**Mise en place de l'aedituus sur un script existant**

## 1. 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.
Il est écrit en PHP5, est orienté objet.

## 2. Architecture

L'aedituus est composé de 2 parties distinctes, d'une part les interfaces avec l'utilisateur pour la connexion, l'inscription... d'autre part un ensemble de scripts pour la gestion et le support de l'utilisateur et de sa session lors de sa navigation.

### 2.1. 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
- commun.php
- fonction.php

et les tables

- membres
- sessions
- session_vars
- membre_vars

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

#### 2.1.1. 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),

- $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.

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.

**NB : Ne pas utiliser le tableau $_SESSION**

liste des méthodes publiques

```php
<?php
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();
};
?>
```

### 2.1.2.Class 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

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

et inversement.

### 2.1.3.Class CUser.class.php

La classe Cuser permet au développeur de manipuler les donnée 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 membres

- 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 ajouter 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 l'autre d'une nouvelle connexion.

### 2.1.4.Conclusion de la partie support de l'utilisateur

Le duo CSession (ou CsessionPHP) et CUser peut-être utilisé indépendamment du reste et permettent un support d'une session utilisateur. Au niveau architecture, la classe Cuser augmente la maintenabilité car toutes les fonctions liées à un utilisateur peuvent être rajoutées dans cette classe. Pour fonctionner, il n'y a besoin que des fichiers et tables précisées au 2.1

## 2.2. 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.

### 2.2.1.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 à été envoyé.

Parmi les fonctionnalités additionnelle de la page inscription, on trouve

- L'ajout dynamique de champ de formulaire
- L'encryptage par l'algorythme RSA du mot de passe coté client, décryptage coté serveur

Pour utiliser l'ajout dynamique de champ de formulaire à l'inscription, il faut procéder comme ceci.

- aller sur votre gestionnaire de BDD (style PHPMyAdmin)
- Selectionner la table 'champs'
- Pour ajouter un champ, insérer une ligne
  - 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

- 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.

### 2.2.2.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.

### 2.2.3.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

### 2.2.4.Conclusion partie Interface

Chaque élément de l'interface peut-être utilisé séparement. Dans chacun d'eux, la partie traitement est clairement séparée de la partie affichage du formulaire. L'utilisation de formulaires extérieurs déjà existants est tout à fait possible, mais de manière à conserver tous les ajouts de sécurité, il est conseillé de garder le même modèle que les formulaires présents dans les fichiers templates de l'aedituus.

## 2.3. 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

```php
<?php
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;
    }
}
?>
```

- 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.

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

```php
<?php
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)

```
<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)

```
    <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

```php
<?php
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();
?>
```

Ceci crée la session (ou la récupère).

**Principales fonctions**
Pour avoir les infos utilisateurs, utiliser
$userdata = $user->getUserData();

Pour utiliser les sessions, utiliser

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

Pour stocker des données permanentes propre à un utilisateur, utiliser

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

Pour empecher l'acces à une page, utiliser

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