

Module FMIN358 Technologies du Web : Accès à une source de données via une interface Web (architecture LAPP)

1. Introduction

Nous considérons ici les applicatifs Web et leur capacité à interfacier différentes catégories de sources de données. Il s'agit notamment de définir des sites Web dynamiques (à l'inverse des pages statiques) qui vont permettre de tirer parti (traiter, restituer, mettre à jour ...) à la volée de l'information contenue dans une ou plusieurs bases de données.

1.1 Quelques éléments terminologiques

- **Tier et architecture 2-tier, 3-tier, n-tier** : le tier est une couche logique d'application et permet de séparer au sein d'architectures applicatives ce qui relève de la présentation, du traitement et de l'obtention des données. Les couches se veulent indépendantes et l'on peut parler d'architecture distribuée.
 1. architecture 1-tier : traitement de texte, logiciel de dessin
 2. architecture 2-tier : modèle client-serveur (ex. client Web/Serveur Web)
 3. architecture 3-tier : approche la plus répandue (ex. Client Web/Serveur Web/Serveur de données) La figure générale de séparation des trois niveaux d'abstraction de l'architecture trois-tier (présentation, traitements, données) est donnée ci-dessous.

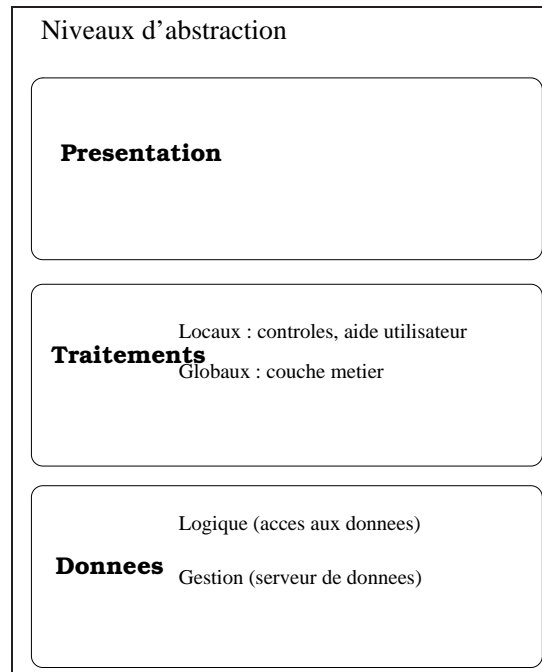


FIG. 1 – Architecture trois-tier

- (a) couche *présentation* : interface homme-machine facilitant l'interaction avec l'utilisateur. Elle peut revêtir différentes formes selon le support considéré.
- (b) couche *logique métier* : traitements qui vont s'opérer sur les données en fonction des requêtes des usagers. Différentes règles de gestion et de contrôle peuvent également être mises en oeuvre à ce niveau.
- (c) couche *données* : les données sont généralement gérées au sein de bases de données. Cette couche s'assure de l'accès comme de la maintenance et de la cohérence des données.

4. architecture n-tier : extension de l'architecture 3-tier : toutes sortes de collaborations sont possibles entre couches comme par exemple entre différents serveurs de données ou entre différents serveurs d'application. Cette architecture multi-tier permet de découpler de manière encore plus fine en répartissant les données ainsi que les traitements sur différents serveurs. ...

– **Acronyme LAPP : Linux Apache PHP Postgresql**

Linux (pour ce qui relève du système d'exploitation), Apache (servir les ressources Web aux clients), PHP (pour créer à la volée en fonction des besoins les pages Web) et Postgresql (pour gérer la persistance des données).

2. La couche logique applicative prise en charge par le serveur HTTP et les scripts serveurs comme ASP ou PHP

L'organisation et le traitement des informations vont s'effectuer du côté du serveur. Le Web dit *dynamique* ne contient pas simplement des pages mais propose aussi des services. **Scripting server** : de manière grossière, le serveur HTTP va traiter les formulaires au format HTML (couche présentation)

et va renvoyer en guise de réponse des pages au format HTML (résultats de traitements, de requêtes sur des sources de données par exemple). Plusieurs techniques sont d'actualité :

- Scripts générant du HTML
 - CGI (Common Gateway Interface) : Perl, C , Tcl, Java , ...
 - NSAPI, ISAPI (APIs de Netscape et Microsoft intégrées au serveur), Servlets (J2EE)
- Scripts insérés dans des pages HTML qui exécutés par le serveur vont se surajouter au source HTML résultat
 - SSS : Server Side Script : ASP (Active Server Page pour IIS (Internet Information Services)), PHP (PHP : pour Personal Home Page ou également Hypertext Preprocessor), JSP (J2EE) (remarque : JavaScript est lui interprété par le client Web) ...
 - SSI : Server Side Include : directives XHTML placées dans du code HTML et évaluées côté serveur (exemple : rajouter la date du jour)


```
<!--#echo var="DATE_LOCAL" -->
```

3. les principes de base de PHP

PHP est un langage interprété, procédural, et dans une première approche, pauvrement typé, qui fait l'objet d'un module au sein d'Apache. Plus récemment, PHP (depuis sa version 4) a intégré un ensemble de fonctionnalités basées sur le paradigme objet. Un des avantages majeurs de PHP est de permettre l'accès natif à différents SGBD dont PostgreSQL ou MySQL (nul besoin d'API comme ODBC). Une documentation complète du langage PHP est disponible sur <http://fr.php.net/manual/fr/>.

3.0.1 Quelques illustrations

L'extension .php va permettre au serveur de reconnaître les fichiers comprenant du php et donc de les interpréter. A défaut, le code s'affichera alors. Les instructions php vont être comprises entre les balises d'ouverture `<?php` ou `<?` et de fermeture `?>`

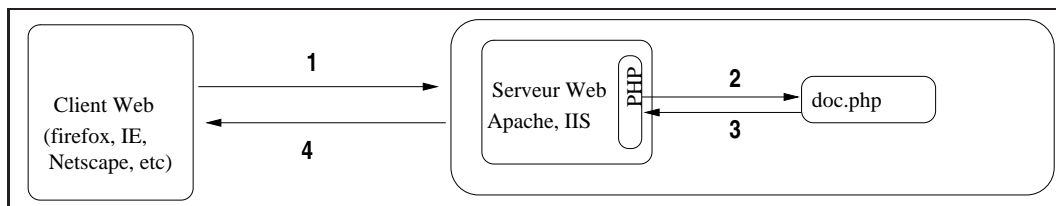


FIG. 2 – PHP et le Web

```
<?php
echo 'Bonjour Monde !!';
?>
```

La fonction echo a pour signature :

```
void echo(string argument1 [, ...string argumentN])
```

et peut être exploitée sous forme simplifiée comme dans l'exemple précédent. Définition d'une variable, affectation d'une valeur et restitution de la valeur

```

<html>
<head> <title> Variable - Affichage </title></head>
<body>
<h1>
<?php
$myvar = "Bonjour Monde";
echo $myvar;
?>
</h1>
</body>
</html>

```

fonction d'affichage print, opération de concaténation et commentaires

```

<html>
<head> <title> Fonctions predefinies - concatenation </title></head>
<body>
<?
// un peu de php
$heure = date("H\hi");
$myvar = "Bonjour Le Monde";
print("<h1>".$myvar." vers ".$heure."</h1>");
?>
</body>
</html>

```

La fonction print a pour signature

```
boolean print(string argument)
```

3.0.2 Variables, constantes et types de données

Les variables débutent toutes par un \$. Si il n'est pas nécessaire de déclarer les variables, il est cependant possible de convertir une variable dans un type de données spécifique (cast) ou d'affecter un type à une variable (settype). PHP admet les types simples : entier (integer), réel (double), alphanumérique (string), booléen (boolean) ainsi que les types complexes tableau (array) et objet et des types spécifiques null (non renseigné) et ressource (par exemple résultat d'une requête). Plusieurs fonctions permettent de tester l'existence d'une variable (isset), de renseigner le type d'une variable (gettype) ou encore de la détruire (unset). Les constantes sont définies à l'aide du mot clé define et, à la différence des variables, ne sont pas précédées par un \$. Il existe des constantes prédéfinies par le système (PHP_VERSION, PHP_OS, _FILE_, _LINE_ par exemple qui renvoient la version de l'interpréteur, le nom du système d'exploitation, le chemin du fichier, ou le numéro de ligne du fichier). Les variables sont interprétées lorsqu'elles sont entre guillemets (avec interpolation), et ne le sont pas lorsqu'elles sont balisées par des apostrophes .

```

<html>
<head> <title> types de variables, tests divers et constantes </title></head>
<body>
<?
define("FACTEUR_CONVERSION",5);
$heures = "mon heure";

```

```

echo gettype($heures)."<br>";
$heuref = 17.15;
echo gettype($heuref)."<br>";
$heurei = 17;
echo gettype($heurei)."<br>";
$heureb = false;
echo gettype($heureb)."<br>";
$tabHeures[] = "une heure";
$tabHeures[] = "deux heures";
$tabHeures[] = "trois heures";
echo gettype($tabHeures)."<br>";
echo $tabHeures[1];
// renvoie deux heures, l'indice commence a 0
$tabMin =array("une minute","deux minutes","trois minutes","quatre minutes","cinq minutes");
// renvoie tous les elements du tableau
print_r($tabMin);
echo sizeof($tabMin);
// boucle pour afficher les elements du tableau
foreach($tabMin as $min)
{
    echo "$min<br>";
}
// convertir une variable
echo (int)$heuref.'<br>';
echo (string)$heuref.'<br>';
// voir si une variable existe ou non
echo isset($existe);
// isset ne renvoie rien si la variable n'existe pas
$existe = true;
echo isset($existe). '<br>';
// renvoie 1
// destruction de la variable existe
unset($existe);
// idem avec gettype qui renvoie null
echo gettype($existe). '<br> <hr>';
$existe = true;
echo gettype($existe)
?>
</body>
</html>

```

3.0.3 Boucles et conditionnelles

Structure conditionnelle if Elle se manipule de manière assez classique. Il est possible d'avoir un moyen simple d'exprimer le if imbriqué. La structure case (choix parmi un ensemble) est également disponible mais n'est pas abordée ici.

```

<?
// if then else simple

```

```

$heures = 17;
if ($heures>16)
echo 'oups il est '.$heures.' heures';
else
echo 'pas de retard';
// if imbriquées avec plusieurs instructions
// date(d) retourne le numero du mois
$mois=date(m);
  if (isset($mois)) {
echo "<center>mes commentaires sur le mois : "
      ."<tt> $mois </tt><br>";

if ($mois==1) {
echo 'mois des soldes <br>';
  echo '<h3><font color=\"blue\"> et il fait froid </h3>';
    } elseif ($mois==2) {
echo 'mois des soldes <br>';
  echo '<h3><font color=\"red\"> et il fait toujours froid </h3>';

    } elseif ($mois==3) {
echo 'mois de mars <br>';
  echo '<h3><font color=\"blue\"> et il fait moins froid </h3>';
    } else {
  echo ' peut etre avril mai ou juin ou ....';
    }
  echo "\n</center>\n";
}
?>

```

Les structures de boucle for et while les boucles for et while peuvent être utilisées sous la forme

```

for (initialisation compteur; test arret; incrementer compteur)
{ serie d'instructions;}

while (test)
{ serie d'instructions; }

```

Un exemple simple vous est donné :

```

<html>
<head> <title> Structures for et while </title></head>
<body>
<?
// boucle for
for ($i; $i<10; $i++)
{ echo date(d).' '.date(m).' '.date(y).'<br>';
}
echo '<br><hr><br>';

```

```
// boucle while
$j=1;
while ($j<10)
{ echo date(d).' ' .date(m).' ' .date(y).'<br>';
$j++;
}
?>
</body>
</html>
```

3.0.4 Les fonctions

Elles revêtent une importance toute particulière en PHP en permettant notamment de *modulariser* le code.

Ecrire une fonction Des exemples d'écriture et d'illustrations vous sont donnés autour de la fonction factorielle. Afin d'assurer modularité et réutilisabilité, il est conseillé de séparer les spécifications des fonctions dans des fichiers externes qu'il conviendra d'appeler au travers des fonctions *require*, *require_once*, *include* et *include_once* qui permettent de faire de la copie d'un fichier dans un autre. Ainsi le fichier dans lequel est spécifiée une fonction peut être appelé dans un autre fichier php dans lequel la fonction va être utilisée. Les différences entre *require* et *include* portent sur la manière de gérer les erreurs : *include()* produit un message d'alerte (warning) en cas d'absence de fichier à inclure (copier-collé) tandis que *require()* retourne une erreur fatale ; et sur la possibilité de copie conditionnelle (test if) pour ce qui concerne *include()*. Il est à préférer les structures *require_once()* et *include_once* qui n'ajoutent le code qu'une seule fois.

```
function nom_fonction (liste arguments eventuellement vide)
{ corps fonction avec une instruction return
}
```

Fichier nommé factorielle.php

```
<?php
function fact($n) {
$fact=1; // initialisation de la variable locale
for ($i=1;$i<=$n;$i++) {
$fact *= $i;
}
return $fact;
}
?>
```

Fichier nommé principal.php

```
<html><body>
    <?php
    include 'factorielle.php';

    for ($x=0;$x<=10;$x++) {
```

```

        echo "$x != ".fact($x)."<BR>\n";
    }
    ?>
</body></html>

```

3.0.5 PHP et la programmation objet

Il est possible d'exploiter certains principes du paradigme objet (héritage, encapsulation, polymorphisme, etc) au travers de PHP et de construire notamment des classes d'objets qui vont posséder des états et des comportements.

Un exemple est donné qui porte sur un classe abstraite `Animal` et un ensemble de classes filles (`Chien`, `Chat` et `Vache`). Les méthodes préfixées par `_` sont des méthodes prédéfinies par PHP.

```

<?php

abstract class Animal
{
    public $espece;
    public $nom;
    public $cri;

# constructeur
    public function __construct($uneEspece, $unNom, $unCri)
    {
        $this->espece = $uneEspece;
        $this->nom = $unNom;
        $this->cri = $unCri;
    }

    /*
     * trier les animaux par ordre alphabétique sur leur nom
     */
    public static function compare($animal1, $animal2)
    {
        if($animal1->nom < $animal2->nom) return -1;
        else if($animal1->nom == $animal2->nom) return 0;
        else return 1;
    }

# fonction d'affichage predefinie
    public function __toString()
    {
        return "$this->nom a pour espece $this->espece et pour cri $this->cri";
    }
}

class Chien extends Animal{
    public function __construct($nom){

```

```

    parent::__construct("Chien", $nom, "wouah !");
}
}

class Chat extends Animal{
    public function __construct($nom){
        parent::__construct("Chat", $nom, "miaou !");
    }
}

class Vache extends Animal{
    public function __construct($nom){
        parent::__construct("Vache", $nom, "meuh !");
    }
}

# tableau pour instancier les classes concretes
$animaux = array(
    new Chien("Pluto"),
    new Vache("Margueritte"),
    new Chat("Gros Minet"),
    new Chien("Dingo"),
    new Chat("Kiki"),
    new Chat("Felix"),
);

# usort trie un tableau en utilisant une fonction de comparaison definie par l'utilisateur
usort($animaux, array("Animal", "compare"));

# affichage fait appel a la methode __toString
foreach($animaux as $animal) echo "$animal<br>\n";

?>

```

3.0.6 Les éléments des formulaires HTML

Les formulaires apportent un peu de *dynamique* aux pages HTML, en prenant en charge des informations fournies par l'utilisateur et en y apportant une réponse contextualisée. Un formulaire comprend une balise de début `<form>` et de fin `</form>` et va contenir un ensemble d'éléments de contrôle autorisant l'interaction avec l'utilisateur : INPUT, TEXTAREA et SELECT notamment.

- INPUT va avoir pour attributs TYPE (ensemble de valeurs TEXT, RADIO, CHECKBOX, PASSWORD, SUBMIT, RESET, HIDDEN) ainsi que NAME (nom de la variable associée) et VALUE (valeur par défaut, valeur sélectionnée, texte écrit sur le bouton etc.)
- SELECT pour créer des listes à choix (avec le mot-clé OPTION) va également avoir un attribut NAME
- TEXTAREA pour des zones de saisie supérieures à une ligne, pourvu des attributs NAME, ROWS et COLS

`<form>`

```

<b><font color="grey" size="+1"> Identification </font></b> <br><hr><br>
<b><font color="grey" size="+1"> Nom </font></b>
<input type="text" name="nomU" value="Martin">
<br>
<b><font color="grey" size="+1"> Mot de passe </font></b>
<input type="password" name="pwdU" value="anonyme">
<br>
<b><font color="grey" size="+1"> Choix exclusif </font></b> <br><hr><br>
<input type="radio" name="choixU" value="travailler"> Travailler <br>
<input type="radio" name="choixU" value="dormir"> Dormir <br>
<br><hr><br>
<b><font color="grey" size="+1"> Liste deroulante </font></b> <br><hr><br>
<select name="liste">
<option value="travail"> Travailler
<option value="dormir"> Dormir
<option value="nspp"> Ne Se Prononce Pas
</select>
<br><hr><br>
<b><font color="grey" size="+1"> Zone de saisie </font></b> <br><hr><br>
<textarea name="commenter" rows=5 cols=20>Vous pouvez me modifier </textarea>
<br><hr><br>
<input type="submit" value="Envoyer">
<input type="reset" value="Effacer">
</form>
</body></html>

```

3.0.7 Traitement des données d'un formulaire HTML par PHP

Nous allons nous placer dans un premier temps dans le cas de figure le plus simple : le formulaire et le script php sont dans le même fichier mais le serveur est configuré dans un mode safe (la variable doit être exploitée au travers d'une *supervariable globale* (\$_GET["nomVariable"])). Par défaut, c'est la méthode GET (données présentes dans le corps de l'URL) qui prédomine (versus POST (données dans le corps de la requête)) et nous n'avons pas besoin d'indiquer le nom du fichier.

```

<html><body>
  <form>
    <textarea rows=3 cols=40 name="saisie"> texte par défaut </textarea>
    <input type="submit" value="Traiter">
  </form>
  <hr><br>
  Les mots saisis sont :
  <ul>
    <?php
// explode retourne un tableau de mots, le caractere de delimitation est l'espace
$capture=$_GET["saisie"];
$mots = explode(" ", $capture);
for ($i=0; $i<count($mots); $i++) {
  echo "<li>".$mots[$i];
}

```

```

    ?>
</body></html>

```

Même chose mais le script et le formulaire sont séparés et la méthode POST est exploitée (fichier ScriptForm2.php ensuite exploité dans le formulaire).

```

<html>
<body>
<h1> Les mots saisis sont </h1>
    <?php
echo "<ul>";
$capture=$_POST["saisie"];
    $mots = explode(" ",$capture);
    for ($i=0; $i<count($mots); $i++) {
        echo "<li>".$mots[$i];
    }
echo "</ul>";
    ?>
</html></body>

<html><body>
    <form action="ScriptForm2.php" method="POST">
    <textarea rows=3 cols=40 name="saisie"> texte par default </textarea>
    <input type="submit" value="Traiter">
    </form>
    <hr><br>
</body></html>

```

3.0.8 PHP pour interfacier une base de données PostgreSQL

Il est de plus en plus fréquent d'utiliser, pour se faciliter la tâche, une couche d'accès aux données (cf PearDB). Nous allons cependant voir ici les principales fonctions nécessaires à la connexion à une base de données et aux traitements de différents opérations de consultation ou de mise à jour de la base de données (SQL imbriqué dans PHP).

- pg_connect : passer la chaîne de connexion
- pg_exec : exécuter une requête
- pg_fetch_array : permet de retourner tuple après tuple à l'aide d'un indice
- pg_num_rows : retourne le nombre de tuples résultats
- pg_close : ferme la connexion

Par exemple, si l'on reprend un des exemples précédents :

```

<?php

$link = pg_connect("host=neptune port=5432 dbname=ips1 user=ips1 password=ips1");
if (! $link) {
    echo "Erreur de connexion a la base. \n";
    exit;
}

$result = pg_exec($link, "select * from employe");
if ($result) {

```

```

$numrows = pg_numrows($result);
echo "<table>";
for($ri = 0; $ri < $numrows; $ri++) {
    echo "<tr>\n";
    $row = pg_fetch_array($result, $ri);
    echo " <td>", $row["nom"], "</td>
    <td>", $row["prenom"], "</td> <br></tr> ";
}
}
else {echo 'aucun tuple dans la table';}
echo "</table>";
pg_close($link);
?>

```

Les opérations d'insertion, de modification ou de mise à jour vont être également gérées au travers de `pg_exec`

```
<?php
```

```

$link = pg_connect("host=neptune port=5432 dbname=ips1 user=ips1 password=ips1");
if (! $link) {
    echo "Erreur de connexion a la base. \n";
    exit;
}

```

```

$requete = "insert into employe (num,nom,prenom) values "
."(3,'Martin','Pierre')";

```

```

$result = pg_exec($link, $requete);
pg_close($link);
?>

```

4. Projet à rendre

Vous travaillerez de manière générale sur de la manipulation d'informations concernant des livres dans le cadre d'une application Web. L'information sera gérée au sein d'une base de données PostgreSQL (comptes généraux ips1 à ips40). Vous construirez dans un premier temps un script de création de tables (et d'insertion de tuples dans ces tables). Vous construirez ensuite les scripts PHP nécessaires pour la consultation de la base de données comme pour sa mise à jour.

Plusieurs étapes peuvent être considérées pour démarrer avec PHP :

- Vous écrirez une fonction de conversion permettant de convertir une somme initialement en euros, en dollars ou en livres et inversement.
- Vous construirez un formulaire contenant des champs permettant de renseigner sur un livre informatique en particulier. Ce formulaire sera associé dans un premier temps à un script qui affichera simplement les choix faits (ou informations saisies) par l'utilisateur. Ce formulaire par suite pourra servir de préalable pour l'insertion d'un tuple dans la table **Livre**
- Pensez à rendre votre programmation modulaire. Dans un second TP, vous travaillerez sur les aspects `session` et couche métier objet.