Initiation au PHP

Actuellement, on sait comment réaliser un site web statique avec des animations JavaScript ou CSS. Il nous manque quelque chose : on ne sait toujours pas créer des pages dynamiquement. On va maintenant voir le dynamisme côté serveur avec le langage PHP.

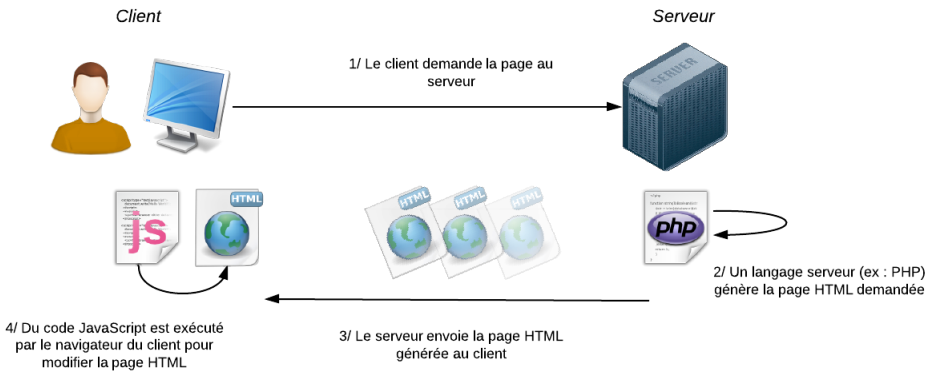

FIGURE 1 – Processus d'obtention d'une page web

On fonctionne sur le principe client/serveur et requête/réponse. Un client envoie une requête à un serveur web, celui-ci interprète la requête et peut alors effectuer deux actions différentes :

- le serveur envoie le fichier demandé (html statique, js, css, images, ...),
- le serveur génère le fichier html en fonction d'un script (par exemple le PHP), en prenant en compte des facteurs changeant.

1 Installation et configuration

Si jusqu'à présent tous les logiciels nécessaires étaient déjà présents sur votre machine, ce n'est normalement plus le cas. Si vous êtes sous Linux, vous devrez installer et configurer LAMP, alors que si vous êtes sous OSX, vous devrez installer et configurer MAMP. La suite de cette section ne s'adresse qu'aux utilisateurs de Windows 10. Vous devrez en effet télécharger la dernière version de WAMP, disponible à l'adresse suivante :

Téléchargement de WAMP

L'installation se fait via cet exécutable (dont le nom est "wamp3.2.0_x64.exe" à l'heure de l'écriture de ce sujet), cependant des dépendances sont généralement manquantes (packages Microsoft Visual Studio), et devront être installées manuellement après l'installation de WAMP. Déroulez l'installation sans changer les paramètres par défaut, mis à part le navigateur et l'éditeur que vous souhaitez utiliser. Dans la mesure du possible, laissez

le répertoire d'installation de WAMP par défaut. Lors du premier lancement de l'application, soit celle-ci vous spécifiera le ou les pacquages manquants à installer (et dans ce cas là, il vous faudra aller le télécharger depuis le site de Microsoft), soit le programme vous signifiera que certaines dll (généralement MSVCR110.dll ou VCRUNTIME140.dll) sont introuvables. Dans ce dernier cas, installez le pacquage suivant (en 86 ET 64 afin de pallier au problème).

Récupération du pacquage

Désormais, lors du lancement de l'application, trois fenêtres de commandes devraient apparaître puis disparaître successivement. Vous remarquez également une icône prendre place dans la barre des tâches, dont la couleur signifie l'état des services :

- rouge : le programme est lancé, mais aucun service ne fonctionne,
- orange : un service sur les deux est actif,
- vert : tout est ok !

Normalement, votre serveur web devrait être maintenant correctement installé sur votre machine et fonctionnel. Nous allons désormais travailler dans le répertoire racine du serveur, qui devraient être :

Ce répertoire est accessible par un client via l'adresse http ://localhost, ou encore http ://127.0.0.1. Notez que la connexion depuis l'extérieur est par défaut désactivée. Créez maintenant un répertoire "TP7", à l'intérieur duquel vous placerez tous les fichiers à servir (fichiers php, css, images, ...). A l'intérieur de celui-ci, créez un fichier "index.php", et intégrez le contenu suivant dans votre nouveau fichier :

Code de test PHP

```php
<?php
  echo "Hello, world !";
?>
```

Ouvrez ensuite votre navigateur et entrez l'adresse du serveur local : `http://127.0.0.1/TP7` ou encore `http://localhost/TP7`. Cela ouvrira la page correspondant à l'interprétation de `index.php` par le serveur, et devrait donc vous afficher le message "Hello, world !". Pour la suite, vous pourrez créer différents répertoires dans le dossier "www". Par exemple, si vous créez le dossier "Forum", contenant le fichier "accueil.php", vous pourrez charger celui-ci via l'adresse : `http://localhost/Forum/accueil.php`.

2 Syntaxe de base

Le code PHP se fait au sein du code HTML. On utilise les balises dédiées `<?php` et `?>`. On peut afficher du texte, des nombres, ... avec l'instruction `echo`.

Votre fichier index.php

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8"/>
    <link rel="stylesheet" href="style.css"/>
    <title>Ma page PHP</title>
  </head>
  <body>
    Voici un texte écrit en HTML</br>
    <?php
    echo "Voici un texte écrit en PHP\n";
    ?>
  </body>
</html>
```

A la suite d'une requête client, ce code est interprété par le serveur. Celui-ci va effectuer l'algorithmique définie par le php afin de générer le code HTML manquant de la page. Par exemple, si on effectue une requête vers le fichier index.php actuel, on reçoit le code HTML suivant :

Code HTML reçu par le client

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8"/>
    <link rel="stylesheet" href="style.css"/>
    <title>Ma page PHP</title>
  </head>
  <body>
    Voici un texte écrit en HTML</br>
    Voici un texte écrit en PHP
  </body>
</html>
```

Le PHP est un langage largement inspiré du C. Il est donc possible d'effectuer un large panel d'instructions. Comme dans tous les langages, il est possible d'utiliser des variables. Une variable se déclare et s'utilise de la manière suivante `$nomVariable`. Il n'est pas nécessaire de déclarer le type, celui-ci est automatiquement reconnu parmi : int, float, boolean et string. Tous les opérateurs classiques sur les nombres sont utilisables.

Déclaration et manipulation de variables

```
<?php
$x = 5;
$y = 7;
echo $x." + ".$y." = ".($x + $y);
?>
```

Pour effectuer des séparations de code sur des tests, on utilise `if`, `elseif` et `else`. On

peut alors créer des conditions complexes grâce aux opérateurs logiques `and`, `or`, `xor` et `!` (non). On dispose également de la structure `switch`, `case` et `break`.

Code de test PHP

```php
<?php
$alea = rand(0,1);
if (!$alea)
{
  ?>
  <p>Bienvenue sur le paragraphe 0 !</p>
  <?php
}
else
{
  ?>
  <p>Voici le paragraphe 1</p>
  <?php
}
?>
```

On dispose des boucles `while` et des boucles `for`. Leur utilisation est similaire à celle dans les autres langages, notamment le C.

Code de test PHP

```php
<p> Voici des nombres aléatoires :
<?php
for ($i=0; $i<100; $i++)
{
  $alea = rand(0,100);
  echo "{$alea} ";
}
?>
</p>
```

2.1 Les chaînes

Contrairement au JavaScript, les chaînes de caractères sont des types primitifs en PHP, il ne s'agit pas d'objets. La déclaration d'une chaîne de caractères peut se faire entre guillemets simples ou doubles.

Code de test PHP

```php
<?php
$maChaine1 = 'Bonjour !';
$maChaine2 = "Hello, world !";
?>
```

Il est également possible d'utiliser Heredoc pour déclarer une chaîne sur plusieurs lignes :

Code de test PHP

```php
<?php
$chaine = <<<EOS
<h1>Une autre façon de déclarer des chaînes</h1>
<p>Ma chaîne de caractère s'étend sur plusieurs lignes ! Merci Heredoc.
↪ PS : Attention de bien mettre l'identifiant de fermeture sur la
↪ <strong>première colonne</strong> de la dernière ligne</p>
EOS;
echo $chaine;
?>
```

De nombreuses opérations sont possibles sur les chaînes de caractères, via des fonctions reconnues nativement par l'interpréteur :

- Concaténation :

Code de test PHP

```php
<?php
$chaine = "Bonjour ".$nom;
?>
```

- Composition :

Code de test PHP

```php
<?php
$chaine = "Bonjour {$nom}";
?>
```

- Calcul de la taille d'une chaîne de caractères :

Code de test PHP

```php
<?php
$chaine = "Hello, world !";
$taille = strlen($chaine);
?>
```

- Recherche de sous-chaînes :

Code de test PHP

```php
<?php
$str = "Mon OS préféré est Linux, Linux c'est cool !";
$n = substr_count($str, "Linux"); //Devrait être égal à 2
$sbStr = substr($str, 19, 5); //Devrait être égal à "Linux"
?>
```

Attention, si vous ne lancez pas cet exemple dans un document HTML correctement formé (notamment sans indication du charset utilisé), il est possible que la sous chaîne `$sbStr` ne soit pas celle attendue, à cause des trois accents présents dans la chaîne originale.

- Modificateur de casse :

Code de test PHP

```php
<?php
$str = "Linux";
$strUp = strtoupper($str);
$strLow = strtolower($str);
?>
```

- Remplacement des caractères spéciaux HTML :

Code de test PHP

```php
<?php
$str = "<strong>texte en gras</strong>";
echo htmlentities($str, ENT_QUOTES);
?>
```

- Découpage de chaîne :

Code de test PHP

```php
<?php
$str = "Linux OSX Windows";
$os = strtok($str, " ");
while ($os)
{
  echo $os;
  $os = strtok(" ");
}
?>
```

2.2 Les tableaux

On dispose de deux types de tableaux. Les tableaux classiques s'utilisent comme en C. Cependant, le nombre de cases n'est pas fixe, on peut en ajouter (ou en retirer) à la volée, et il n'est pas nécessaire de gérer l'allocation (youpi !) :

Code de test PHP

```php
<?php
$tab = array('Zelda', 'Halo', 'Resident Evil');
$tab[3] = 'The Elder Scrolls';
$tab[] = 'The witcher';

foreach ($tab as $jeu)
{
  echo "<li>{$jeu}</li>";
}
?>
```

Tout comme en JavaScript, il existe de nombreuses fonctions de gestion des tableaux, reconnues nativement et permettant de nombreuses opérations pratiques :

- La fonction `count($tab)` renvoie la taille du tableau `$tab`.

- La fonction `array_sum($tab)` renvoie la somme des éléments du tableau `$tab`.
- La fonction `in_array($elt, $tab)` vérifie si $*elt* se trouve dans le tableau `$tab`.
- La fonction `array_search($elt, $tab)` renvoie la première position de `$elt` dans le tableau `$tab`, `false` sinon.

Toutes les fonctions sont disponibles de base, et leur documentation est consultable via le lien ci-dessous.

Documentation

Les tableaux associatifs gèrent des couples de clefs et valeurs : à chaque clef sa valeur. La déclaration se fait via `$tab = array('clef' => $val, ...)`, les clefs sont des chaînes de caractères. La fonction `array_key_exists('clef', $tab)` vérifie l'existence d'une valeur associée à la clef dans le tableau. L'accès à un élément se fait via `$tab['clef']`, ou via l'énumération de la boucle `foreach`.

Code de test PHP

```php
<?php
$tab = array(
"Nintendo" => "Zelda",
"Microsoft" => "Halo",
"Sony" => "Resident Evil");

foreach ($tab as $plateforme=>$jeu)
{
  echo "<li>{$plateforme} : {$jeu}</li>";
}
?>
```

2.3 Le temps

La fonction `date` permet de générer une chaîne de caractères représentant le temps actuel.

Code de test PHP

```
string date ( string $format [, int $timestamp = time() ] )
```

De nombreuses lettres permettent d'afficher des composantes de la date de différentes manières, on citera par exemple :

- "d" : pour afficher le numéro du jour dans le mois, avec un zéro supplémentaire si besoin.
- "m" : pour afficher le numéro du mois dans l'année, avec un zéro supplémentaire si besoin.
- "Y" : pour afficher l'année sur 4 chiffres.
- "H" : heures au format 24h.
- "i" : minutes avec les zéros initiaux.
- "s" : secondes avec les zéros initiaux.

Voici un exemple d'utilisation de la fonction `date`, en profitant de l'occasion pour déclarer nos premières propres fonctions :

Code de test PHP

```php
<?php
function formatterDate(){
  return date("d/m/Y");
}
function formatterHeure(){
  return date("H:i:s");
}
?>
<p>
<?php
  echo "Le ".formatterDate().", à ".formatterHeure();
?>
</p>
```

2.4 L'inclusion de fichiers

Code de test PHP

```php
<!doctype html>
<html>
<head>
<meta charset="utf-8"/>
<link rel="stylesheet" href="style.css"/>
<title>Site de Flash !</title>
</head>
<body>
<?php require "includes/baniere.php";?>
<?php require "includes/menu.php";?>
<div id="main">
...
</div>
<?php require "includes/piedDePage.php";?>
</body>
</html>
```

3 Les variables superglobales

Les variables superglobales sont des variables déclarées par le serveur. Elles sont toujours de la forme `$_NOMVAR`. Certaines permettent de récupérer le contenu des formulaires saisis par les utilisateurs, alors que d'autres permettent d'assurer une continuité entre les pages via les sessions ou les cookies. On peut également récupérer des fichiers envoyés par l'utilisateur !

Pour rappel, un formulaire est une balise encadrante `<form>` et `</form>`. Chaque élément de formulaire possède un nom, qui permettra de récupérer sa valeur dans le PHP.

Une balise `<form>` possède un attribut `method` (get ou post) et un attribut `action` (la page cible).

Code de test PHP

```
<form method="get" action="validation.php">
  <input type="text" name="nom"/>
  <input type="password" name="mdp"/>
  <input type="submit" value="Envoyer"/>
</form>
```

Une requête GET passe les paramètres dans l'url, qui sont donc visibles par tout le monde.

Code de test PHP

```
http://domaine/validation.php?nom=val1&password=val2
```

Ces paramètres seront accessibles sur la page cible via le tableau associatif `$_GET`, où les clefs sont les noms des éléments de formulaire, associés aux valeurs saisies par l'utilisateur.

Code de test PHP

```
<?php
if ((isset($_GET["nom"]))&&(isset($_GET["mdp"])))
{
  echo "Salut {$_GET["nom"]} !";
}
else
{
  echo "Nous n'avons pas été présenté...";
}
?>
```

Une requête POST passe les paramètres dans le corps de la requête, ce qui est bien plus sécurisant, notamment grâce au https.

Code de test PHP

```
http://monUrl/maPage.php
```

Ces paramètres seront accessibles sur la page cible via le tableau associatif `$_POST`, où les clefs sont les noms des éléments de formulaire, associés aux valeurs saisies par l'utilisateur. On reprend le formulaire précédent, en modifiant la méthode du formulaire (post), et la cible.

Code de test PHP

```php
<?php
if ((isset($_POST["nom"]))&&(isset($_POST["mdp"])))
{
  echo "Salut {$_POST["nom"]} !";
}
else
{
  echo "Nous n'avons pas été présenté...";
}
?>
```

On peut envoyer des fichiers via les formulaires : ceux-ci seront uploadé sur le serveur. Il est nécessaire d'ajouter l'attribut `enctype` à la balise `<form>`. On doit également ajouter un champ caché contenant la taille maximum autorisée.

Code de test PHP

```php
<form enctype="multipart/form-data" action="recupFichier.php"
 ↪  method="POST">
  <input type="hidden" name="MAX_FILE_SIZE" value="3000000" />
  <input name="fichier" type="file" />
  <input type="submit" value="Envoyer" />
</form>
```

Côté PHP, on récupère un tableau associatif, chaque case représentant un fichier (plusieurs fichiers possibles par formulaire). Le fichier sera enregistré avec un nom aléatoire dans le dossier temporaire défini dans php.ini Le fichier temporaire est supprimé après l'exécution du script PHP. Le code suivant est un exemple de récupération du fichier, et de déplacement de celui-ci dans le répertoire "Uploads" (à créer au préalable) :

Code de test PHP

```php
<?php
if (isset($_FILES["fichier"])){
  if ($_FILES["fichier"]["error"]==UPLOAD_ERR_OK){
    $path = __DIR__."/Uploads/".$_FILES["fichier"]["name"];
    if (move_uploaded_file($_FILES["fichier"]["tmp_name"], $path))
    {
      echo "Enregistrement effectué";
    }
    else
    {
      echo "Echec de l'enregistrement";
    }
  }
  else{
    switch ($_FILES["fichier"]["error"]){
      case UPLOAD_ERR_INI_SIZE:
        echo "Taille dépassant la taille maximum définie dans php.ini";
      break;
    ...
    }
  }
}
?>
```

Les variables de sessions permettent de maintenir une continuité entre les pages. Pour chaque client, un tableau associatif temporaire est créé. Ce tableau a une durée de vie limitée. Pour utiliser les sessions dans une page, il est nécessaire d'inclure le code ci-dessus **en début de fichier**.

Code de test PHP

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

On peut détruire manuellement une variable de session via :

Code de test PHP

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

On va réaliser une page gérant l'authentification. On doit gérer 3 cas d'arrivée de l'utilisateur sur la page :

- L'utilisateur est authentifié : il a accès à sa page.
- L'utilisateur a envoyé une requête d'authentification : on teste si les paramètres sont corrects et on actualise le statut si les identifiants sont valides.
- L'utilisateur n'est pas authentifié et n'a pas envoyé de requête d'authentification : on affiche un formulaire.

On va utiliser pour cela deux variables superglobales : `$_SESSION` et `$_POST`. Commencez par créer un fichier `mapage.php`, contenant le code d'une page HTML minimale (balises html, head, body, ...). Ensuite, entre les balises `<body>` et `</body>`, insérez le code suivant :

Code de test PHP

```
<?php
session_start();
if (isset($_SESSION["login"]))
{
  ?>
  <p>Bonjour utilisateur ! bienvenue sur votre espace personnel !</p>
  <?php
}
```

Si la variable `$_SESSION["login"]` est définie, alors l'utilisateur est déjà authentifié. Il a donc accès à ses informations.

Code de test PHP

```
elseif ((isset($_POST["login"]))&&(isset($_POST["mdp"])))
{
  if (($_POST["login"]=="flash2016")&&($_POST["mdp"]=="souris"))
  {
    $_SESSION["login"] = $_POST["login"];
    ?><p>Authentification réussie !</p><?php
  }
  else
  {
    ?><p>Vous ne passerez PAAAS !!</p><?php
  }
}
```

Sinon, on vérifie l'existence des variables et . Si elles existent, alors l'utilisateur vient de faire une demande d'authentification via un formulaire. On compare donc leurs valeurs à celles qui sont attendues (ici on le fait de manière simpliste, mais il est évident que l'on ne stocke jamais en clair les mots de passe).

Code de test PHP

```
else
{
  ?>
  <form action="mapage.php" method="post">
  <input type="text" name="login"/>
  <input type="password" name="mdp"/>
  <input type="submit" value="Envoyer"/>
  </form>
  <?php
}
?>
```

Dernier cas : l'utilisateur n'est pas authentifié, ni en train de s'authentifier. On affiche alors un formulaire permettant la saisie du nom d'utilisateur et du mot de passe.

Il est important de noter que l'on ne réalise jamais une authentification via la méthode GET : les noms d'utilisateurs et mots de passe seraient visibles par tout le monde, malgré le chiffrement dû au protocole HTTPS.

4 Interaction avec une base de données

Les bases de données permettent le stockage optimisé des informations. Il existe plusieurs systèmes de gestion de bases de données (SGBD), on se servira de mysql. Pour dialoguer avec une base de données, on utilise le langage SQL. Vous le verrez (ou bien l'avez vu, selon l'année) de manière intensive dans votre cours de bases de données. Une base de données contient des tables, regroupant les informations. On peut imaginer une table comme un tableau sous Excel. Les "colonnes" d'une table sont les champs, chaque champ possédant un type : entier, réel, chaîne de caractère, ... Les "lignes" d'une table sont les entrées, chaque entrée possédant une valeur pour chaque champ. Pour se connecter à une base de données, on peut utiliser PDO, pour peu que l'on possède l'adresse du serveur SGBD, le nom de la base, ainsi que le login et mot de passe d'un utilisateur autorisé du SGBD :

Code de test PHP

```
1  try
2  {
3    $bdd = new PDO(
4    'mysql:host=localhost;dbname=maBase;charset=utf8',
5    'login',
6    'mdp',
7    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)
8    );
9  }
10 catch (Exception $e)
11 {
12   die($e->getMessage());
13 }
```

On utilise `$bdd->query` pour exécuter une requête SQL. On peut par exemple procéder à une lecture (SELECT), une insertion (INSERT) ou encore une suppression (DELETE). On utilise ensuite `$reponse->fetch()` pour traiter les informations renvoyées par la base de données ligne par ligne.

Code de test PHP

```
1  $reponse = $bdd->query('SELECT * FROM Utilisateurs');
2  while ($donnees = $reponse->fetch())
3  {
4    echo "<p> {$donnees["pseudo"]} {$donnees["inscription"]}</p>";
5  }
6  $reponse->closeCursor();
```

On va vouloir procéder à l'exécution de requêtes en fonction des informations envoyés

par l'utilisateur. Cela pose un problème de sécurité : on est tenté de concaténer directement les variables à la requête, comme dans le code suivant :

Code de test PHP

```
1 $reponse = $bdd->query('SELECT * FROM Messages WHERE
  ↪ ID_Sujet='.$_POST["sujet"].' AND
  ↪ creation<='.$_POST["dateLimite"].'');
```

Pour éviter les injections SQL, il est préférable de passer via `$bdd->prepare` : c'est bien plus sûr, mais également plus rapide en moyenne :

Code de test PHP

```
1  $req = $bdd->prepare('SELECT * FROM Message WHERE ID_Sujet = :sujet AND
   ↪ creation <= :dateLimite);
2
3  $req->execute(array(
4    "sujet" => $_POST['sujet'],
5    "dateLimite" => $_POST['dateLimite']
6  ));
7
8  while ($donnees = $req->fetch())
9  {
10   //TRAITEMENT
11 }
12 $req->closeCursor();
```

5 Création de classes

Depuis PHP5, on peut faire du PHP objet ! On peut donc déclarer nos classes et organiser notre code plus proprement. On ne rentrera pas dans les détails de la programmation orientée objet, mais on profitera de ces avantages afin de regrouper les données de manière plus logique. Une classe et ses attributs se déclarent comme ci-dessous :

Code de test PHP

```
1 class Etudiant
2 {
3   private $_nom;
4   private $_prenom;
5   private $_ddn;
6   private $_groupe;
7 }
```

On peut également ajouter un constructeur à notre classe. Le constructeur va initialiser la valeur des attributs d'un objet (contenu dans une variable) qui instanciera notre classe :

Code de test PHP

```
1  class Etudiant
2  {
3    private $_nom;
4    private $_prenom;
5    private $_ddn;
6    private $_groupe;
7
8    public function __construct($nom, $prenom, $ddn)
9    {
10     $this->_nom = $nom;
11     $this->_prenom = $prenom;
12     $this->_ddn = $ddn;
13   }
14 }
```

Les attributs étant par défaut privés, on ne peut pas accéder directement aux valeurs stockées dans notre objet (ce qui est dommage pour le coup). On déclare pour chacun d'eux des mutateurs (setters), qui modifieront leur valeur, ainsi que des accesseurs (getters) pour consulter leur valeur. Dans l'exemple suivant, on définit le modificateur `setNom($nom)` et l'accesseur `nom()` :

Code de test PHP

```
1  class Etudiant
2  {
3    ...
4    public function setNom($nom)
5    {
6      $this->_nom = $nom;
7    }
8    public function nom()
9    {
10     return $this->_nom;
11   }
12 }
```

L'intérêt est non seulement de regrouper les données au sein d'un même objet, mais également d'ajouter d'autres méthodes, par exemple `toString()` qui va formater l'affichage de l'objet :

Code de test PHP

```
class Etudiant
{
  ...
  public function toString()
  {
    return $this->_prenom." ".$this->_nom.", né le
  ↪  ".date("d/m/Y",$this->_ddn);
  }
}
```

Imaginons que cette classe soit définie dans un fichier "Etudiant.php", vous pouvez alors l'utiliser de la manière suivante :

Code de test PHP

```
require "Etudiant.php";

$etudiant = new Etudiant("Le Berre","Matthieu",557043625);
echo "<p> {$etudiant->toString()} </p>";
```

6 Création d'un livre d'or

Vous avez normalement vu suffisamment de notions en cours de bases de données pour vous débrouiller. Cependant, je vous recommande l'usage de l'outil **phpmyadmin** pour la gestion rapide des bases de données et des tables que nous utiliserons dans ce TP et le suivant. Pour information, phpmyadmin est un utilitaire fourni sous la forme d'un ensemble de fichiers php, permettant la création et la modification de bases de données de manière graphique. Cet outil est automatiquement installé via WAMP (et accessible via le menu déroulant), mais doit être déployé manuellement sous Linux (on télécharge l'ensemble des fichiers que l'on place ensuite dans /var/www/html).

Lancez phpmyadmin via le menu, ou en entrant l'adresse `http://localhost/phpmyadmin`. Vous devriez voir apparaître une fenêtre de connexion. Sous WAMP, l'utilisateur **root** n'a pas de mot de passe par défaut. Sous Linux, vous devrez créer un utilisateur avec son mot de passe via la ligne de commandes, puis lui accorder les privilèges suffisant pour qu'il puisse créer une nouvelle base de données.

On souhaite réaliser un livre d'or, afin de permettre aux visiteurs de votre site de laisser des messages. Afin d'enregistrer ceux-ci, on va créer une nouvelle base de données 'LivreOr' sur votre serveur. Dans l'interface de phpmyadmin, il suffira, sur la partie gauche de l'écran, de cliquer sur "Nouvelle base de données" :

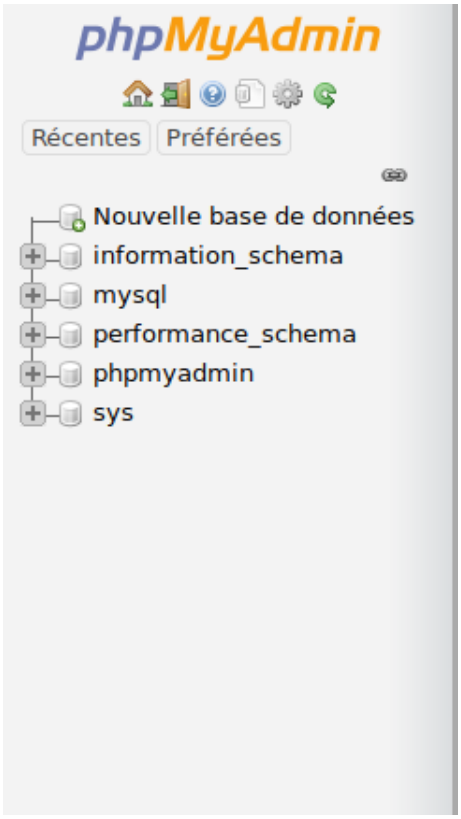

FIGURE 2 – Créer une nouvelle base de données

Cette base de données est créée vide. Il va donc être nécessaire de lui ajouter une table "Messages", qui contiendra quatre colonnes :

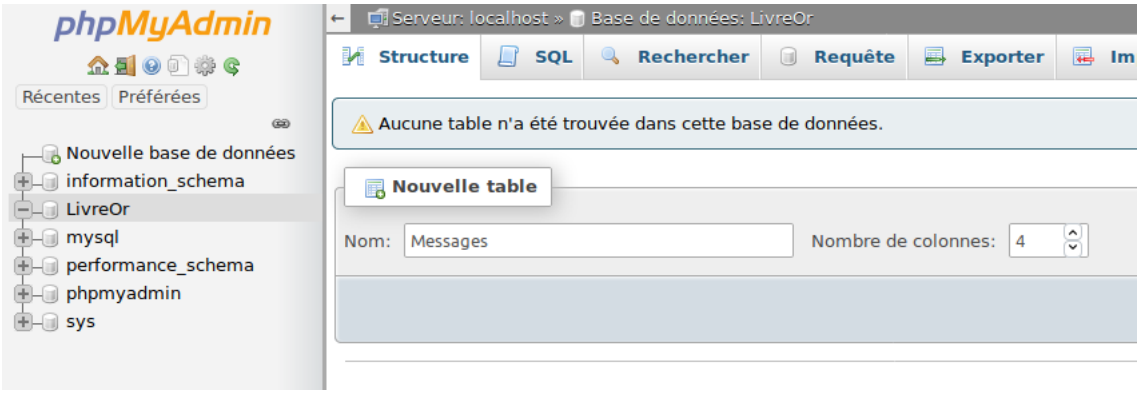

FIGURE 3 – Création de la table Messages

Nous utiliserons la structure de table suivante :

- "ID" représentera l'identifiant unique du message, et sera la clef primaire de la table,
- "pseudo" sera le nom d'utilisateur laissé par le visiteur,
- "message" sera le texte laissé par le visiteur, en prenant soin de supprimer les caractères spéciaux afin d'éviter les attaques,
- "creation" sera le timestamp représentant le moment où le message a été posté.

Vous obtiendrez facilement ce résultat via phpmyadmin en remplissant les champs de la manière suivante :

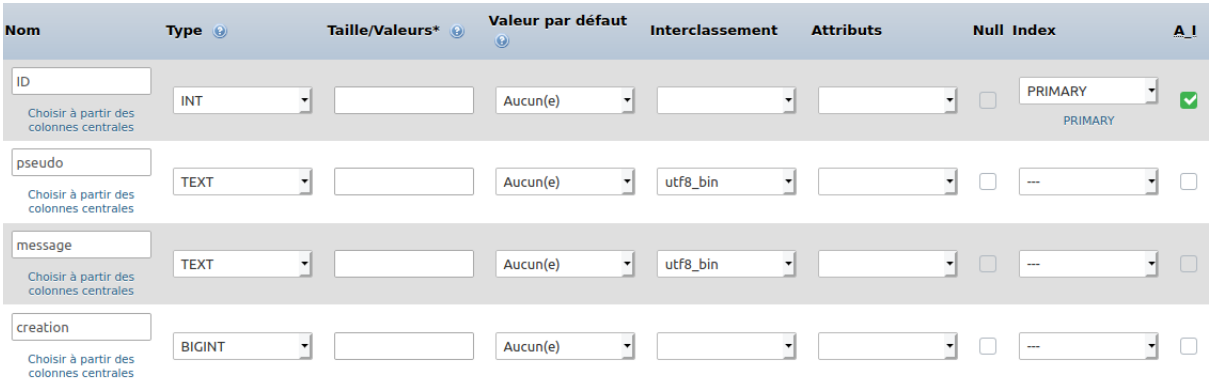

FIGURE 4 – Création de la table Messages

Passons maintenant au développement de notre livre d'or. Commencez par déployer le site web créé lors du deuxième sujet sur votre serveur web, en collant toute l'arborescence du site dans un répertoire, lui-même dans le dossier "www" de votre serveur. Vous devriez être capable de naviguer sur celui-ci depuis l'adresse http ://localhost/monsite/index.html Créons maintenant une page "livreor.php", qui sera chargée à la fois de l'affichage des messages, mais également de l'enregistrement de ceux-ci. Cette page reprend la charte graphique de votre site (rapidement, ce n'est pas l'objectif principal de cet exercice). Le première chose à faire est d'assurer la connexion à notre base de données :

Code de test PHP

```
try
{
  $bdd = new PDO(
    'mysql:host=localhost;dbname=LivreOr;charset=utf8',
    'root',
    '',
    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)
  );
}
catch (Exception $e)
{
  die($e->getMessage());
}
```

La page vérifie ensuite la présence d'un post en cours via la variable superglobale `$_POST` :

Code de test PHP

```
if ((isset($_POST["pseudo"]))&&(isset($_POST["message"])))
{
  $date = new DateTime();
  $req = $bdd->prepare("INSERT INTO Message(ID, pseudo, creation,
  ↪ message) VALUES (NULL, :pseudo, :creation, :message)");
  $req->execute(array(
    "pseudo" => $_POST["pseudo"],
    "message" => $_POST["message"],
    "creation" => $date->getTimestamp()
  ));
}
```

Si un pseudo et un message ont été postés, on créé une nouvelle entrée dans la table 'Message' de notre base de données. Ensuite, le corps de page affiche un formulaire permettant la saisie des messages :

Code de test PHP

```
<form method="POST" action="livreor.php">
  <h1>Poster un nouveau message</h1>
  <div id="pseudo">
    <label for="pseudo">Votre pseudo :</label>
    <input type="text" name="pseudo"/>
  </div>
  <div id="message">
    <textarea name="message" rows=10></textarea>
  </div>
  <div id="envoyer">
    <input type="submit" value="Envoyer"/>
  </div>
</form>
```

Enfin, on s'occupe de l'affichage des messages présents dans la base de données. Pour cela, vous créez un fichier "Message.php", qui contiendra la classe suivante :

Code de test PHP

```
class Message
{
  private $_pseudo;
  private $_creation;
  private $_message;

  ...
}
```

Complétez celle-ci afin de mettre en place le constructeur et les accesseurs, ainsi qu'une méthode permettant d'afficher la date et l'heure correspondant au timestamp du message. Enfin, il ne nous reste plus qu'à afficher la liste des messages sous le formulaire :

Code de test PHP

```
1  require "Message.php";
2  $req = $bdd->prepare("SELECT * FROM Message");
3  $req->execute();
4
5  while ($donnees = $req->fetch())
6  {
7    $message = new Message($donnees["pseudo"], $donnees["message"],
   ↪  $donnees["creation"]);
8
9    echo "<div class='message'><h1>{$message->pseudo()},
   ↪  {$message->formatDate()}</h1>{$message->message()}</div>";
10 }
11 $req->closeCursor();
```