1 Les sessions

Du fait que, comme on l'a vu au premier cours, le serveur HTML ne peut pas mémoriser les sessions, il ne peut pas indiquer à PHP quelle est l'avancée du dialogue avec l'utilisateur lors de l'appel d'un fichier PHP. Il faut donc faire gérer cela à PHP lui-même. Une réponse partielle est d'installer des cookies sur la machine du client mais nous allons plutôt minimiser cette approche, entre autres parce que l'utilisateur peut bloquer ou modifier à sa guise les cookies, ce qui nuit à la fiabilité du système. Nous allons utiliser la gestion de sessions de PHP qui minimise l'usage des cookies. Par ailleurs, la base de données SQL est bien plus fiable que des cookies restant sur la machine du client pour mémoriser les informations persistantes d'une session à une autre.

En PHP, on ouvre la session avec la fonction booléenne `session_start()` (sans arguments). Cela crée par défaut un cookie d'identification de session sur la machine du client et c'est le seul cookie qui sera utilisé (et de manière transparente) par PHP. Les autres informations seront stockées dans un tableau global : `$_SESSION[]`. En pratique, ce tableau est stocké dans un répertoire temporaire *du serveur*, ce qui évite entre autres que le client puisse modifier les données.

En général, après `session_start()`, on limite volontairement les pages offertes au client jusqu'à ce qu'une procédure de login soit passée avec succès. En pratique, on commence donc souvent par mémoriser quelque chose du genre `$_SESSION['loginOK']=FALSE` et on programme une procédure de login qui fera appel à la base de données SQL pour vérifier le mot de passe (encrypté !) avant de passer cette variable à `TRUE`. Il suffit alors de tester `$_SESSION['loginOK']` pour paramétrer le comportement des pages envoyées au client ou leur droit d'accès. Cependant bien d'autres variables `$_SESSION['...']` peuvent être utilisées, que le client soit logué ou non : la seule limite est l'imagination du programmeur PHP.

En pratique, ce sont bien sûr essentiellement les informations qu'on récupère des divers `$_POST['...']` au cours des pages de formulaires envoyées pendant la session (éventuellement retravaillées par diverses fonctions PHP) qui servent à remplir le tableau `$_SESSION[]`, lui-même à la base de mémorisations plus durables (fichiers ou base de données). Ne jamais oublier cependant que le client peut interrompre une session à tout moment sans préavis. Il faut donc que les données restent cohérentes dès que les `$_POST[]` reçus ont été traités, page après page, sans attendre un éventuel « logout » par exemple pour assurer la cohérence.

Il existe une commande `session_destroy` mais elle a fort peu d'intérêt puisque l'utilisateur peut de toute façon abandonner la session à tout moment sans prévenir. De plus cette commande se contente de vider le tableau `$_SESSION[]` sans supprimer le cookie de session, donc elle ne clos pas réellement la session.

2 La gestion des fichiers et le type `resource`

Commençons par remarquer que pour mémoriser valablement des informations dans le cadre qui nous préoccupe, mieux vaut utiliser une base de donnée (e.g. MySQL au prochain cours) plutôt que des fichiers de texte. Cependant il est parfois (exceptionnellement) utile de gérer des fichiers.

Pour ouvrir un fichier, on utilise `fopen()` qui prend en premier argument l'adresse du fichier (de type `string`) et pour second argument `"r"` pour ouvrir en lecture, `"w"` pour ouvrir en écriture, `"a"` pour ouvrir en écriture sans écraser le fichier. C'est donc très comparable à ce que l'on connaît déjà en Python. Pour `"w"` et `"a"`, si le fichier ne préexiste pas, il est créé.

En plus de ces modes d'ouverture de fichier identiques à Python, les modes `"w+"` et `"a+"` permettent de cumuler respectivement `"w"` ou `"a"` avec `"r"`. Même avec `"a+"` la lecture commence alors au début du fichier (mais l'écriture à la fin, bien sûr).

Le type retourné par `fopen()` est « `resource` », au lieu de « `file` » en Python, car on verra que d'autres structures de données que les fichiers sont gérés de manière similaire en PHP.

Compte-tenu du grand nombre de sessions qui peuvent être ouvertes en même temps sur un serveur, le risque que plusieurs scripts PHP manipulent un même fichier est grand. Il est donc recommandé de *vérouiller* le fichier dès qu'on l'ouvre. On peut vérouiller l'écriture (`1`), ou l'écriture et la lecture (`2`). Par exemple :

```
$f = fopen("/adresse/du/fichier.txt", "w");
flock($f,1);
```

Nota : par convention, `$f` est un entier (type `integer`) qui vaudra `0` si `fopen()` échoue, de sorte qi'il arrive souvent de tester : « `if($f) {...}` » après un `fopen()`.

De plus, une fois le travail sur le fichier terminé, il faut penser à le dévérouiller (`3`) *avant* de le clore :

```
flock($f,3);
fclose($f);
```

Note : dans certaines versions de PHP, flock() est défectueux et ne marche que si le fichier n'est pas vide (!).

Pour écrire dans le fichier, on utilise :

```
fwrite($f,"La chaîne à écrire");
```

Pour lire un nombre d'octets donné, le second argument indique combien :

```
fread($f,80);
```

Il se peut que la chaîne de caractères retournée soit plus courte si la fin de fichier est atteinte (ou si un time-out de lecture est atteint).

Pour lire jusqu'à la fin de ligne : `fgets($f);`

Enfin les fonctions suivantes sont également utiles :

- Pour vérifier si un fichier existe (par exemple avant de l'ouvrir) :
  `file_exists("/adresse/du/fichier.txt");`
- Pour connaître le taille d'un fichier en octets :
  `file_size("/adresse/du/fichier.txt");`

A propos des droits d'accès, il faut toujours avoir en tête que le processus PHP qui manipule les fichiers appartient à l'utilisateur `apache`. Il ne peut donc le faire que dans des répertoires et sur des fichiers où l'utilisateur `apache` a les droits requis pour le faire (cf. cours d'Administration Système et Réseau).

2.1 Exercices

Exercice 1 : Mettre dans un fichier la liste des numéros IP, avec le navigateur utilisé, qui ont accédé à une page web donnée. Bien structurer la chose en faisant une fonction PHP dans un fichier à part, ce qui permet de l'utiliser dans plusieurs pages différentes, mais en mettant à jour toujours le même fichier. Dans un second temps, ajouter les sécurités qui évitent que plusieurs pages ouvrent le fichier en même temps, au prix de manquer l'information si le fichier est non disponible.

Ensuite faire une autre page capable de présenter les résultats.

Exercice 2 : Utiliser un fichier unique pour mémoriser le nombre d'accès à une page web, ceci toujours avec une bonne structuration de sorte que plusieurs pages peuvent faire appel à cette fonctionnalité, mais cette fois, il faut naturellement un fichier différent par page.

Ecrire également la fonction produisant en résultat ce nombre d'accès, de sorte qu'il puisse (ou non) être rendu publique dans la page qui utilise la fonction.

Exercice 3 : Faire une page web qui centralise un inventaire de commandes à faire, par exemple pour tout un service administratif. Chaque client (i.e. employé du service administratif en question) pouvant choisir dans une liste de produits donnée, donner le nombre d'exemplaire qu'il veut, puis valider son choix. Après une page de confirmation, sa commande est ajoutée dans le fichier qui contient l'inventaire.

Faire également une page qui affiche proprement cet inventaire.