

TP n° 7

Persistance des données chez le client

Comme pour les TP précédents, vous êtes encouragés à chercher les informations dont vous avez besoin auprès des sources fiables. Nous vous conseillons les deux adresses suivantes :

- <https://www.w3schools.com/html/default.asp> ;
- <https://www.php.net/>.

Préambule

HTTP : Un protocole sans état. À chaque requête HTTP, le **client** envoie une requête au **serveur**, qui consiste en une commande (GET ou POST) pour accéder à la ressource associée à une URL, avec des paramètres (cookies...) et éventuellement un contenu (pour POST). Le **serveur** envoie ensuite une réponse qui contient des paramètres (nouveaux cookies, taille de la réponse...) et le contenu de la réponse (une page HTML, une image...). À chaque nouvelle requête, la communication « repart à zéro » : le **serveur** ne se souvient pas (à priori) du **client**.

Donc, on dit du protocole HTTP qu'il est **sans état** : à priori, on n'a besoin de conserver aucune trace des communications précédentes entre un **serveur** et un **client**.

Ajouter de l'état à HTTP. Dans de nombreux cas, on voudrait que le **serveur** se souvienne du **client**. Par exemple, si vous allez sur MonReseauSocial.fr, et que vous vous êtes déjà connecté au site, la réponse devrait inclure vos amis, vos derniers messages reçus, etc. Comment faire pour que les utilisateurs de MonReseauSocial aient une réponse différente alors qu'ils vont tous à la même adresse (MonReseauSocial.fr) ?

Comme le protocole HTTP est sans état, la seule façon d'obtenir une réponse différente est d'envoyer une requête différente.

En ajoutant une mémoire au **client** (et/ou au **serveur** pour les sessions qu'on verra la semaine prochaine), il deviendra alors possible de se souvenir des communications précédentes. Cela va créer une persistance de données. Dans ce TP, nous allons pratiquer plusieurs techniques de persistance des données. Chaque technique a ses défauts et ses avantages ; à force de pratique, vous apprendrez à utiliser la technique qui convient à chaque situation.

Toutes les techniques se basent sur l'idée de modifier la requête HTTP afin que le **serveur** puisse, en examinant cette requête, décider de renvoyer une réponse différente.

1 Persistance par paramètres GET et POST

On commence avec POST et GET que vous avez déjà vus.

Exercice 1 — *Pseudo-persistance par GET*

Nous allons créer une page qui compte combien de fois elle a été visitée. Elle offrira un prix au visiteurs les plus fidèles !



1. Créez une page `countGET.php` qui contient un lien vers elle-même, comme ci-dessus.
2. La page utilisera un paramètre GET nommé `visites` pour compter le nombre de visites précédentes. Le lien pointera vers la même page, mais avec le paramètre `visites` incrémenté d'une unité. De cette façon, à chaque fois que vous cliquerez sur le lien, le compteur de visites va être incrémenté.
3. Faites en sorte qu'au delà de la millionième visite, le message « Félicitations ! Pour vous remercier de votre fidélité nous vous offrons 1000 euros ! » s'affiche.
4. Sans modifier votre code (et sans cliquer un million de fois !), gagnez les 1000 euros.

Exercice 2 — Pseudo-persistence par POST

Nous répétons le même exercice, mais cette fois grâce aux paramètres POST.

Nous allons utiliser un champ de formulaire caché pour envoyer le nombre de visites automatiquement.

La syntaxe ci-dessous

```
<input type="hidden" name="clé" value="valeur" />
```

envoie par la méthode POST le couple clé-valeur lors du submit du formulaire, mais cette information est invisible pour l'utilisateur.

1. Créez une page `countPOST.php` qui compte le nombre de visites comme dans l'exercice précédent. Vous remplacerez le lien par un formulaire contenant :
 - (a) un bouton submit
 - (b) un champ caché (`hidden`) contenant le nombre de visites.
2. Utilisez les outils d'inspection de votre navigateur pour gagner les 1000 euros sans modifier votre code PHP : Pour cela, dans Firefox, allez dans le menu “≡” en haut à droite, puis, “Web developer/Network”. Ensuite, appuyez sur “reload” : vous cliquez sur la ligne où il est écrit “POST”, maintenant, il vous reste à choisir “Resend/Edit and resend” (en haut à droite de la demie-page) et vous modifiez la valeur de visites. “Send”, fait apparaître une deuxième ligne “POST”, en cliquant dessus puis sur “Response”, vous verrez la page web (en bas de l'écran) avec le cadeau.

2 Persistance par les Cookies

Les méthodes précédentes ont un défaut majeur : l'information de persistance n'est stockée nulle part. C'est à vous de vous charger de la renvoyer à chaque fois, d'une manière ou d'une autre (édition de l'url ou formulaire).

Un cookie est une information stockée par le **client** (par exemple, votre navigateur) et qu'il ajoute automatiquement à chaque requête à un **serveur**. Cela permet au **serveur** de « reconnaître » le **client**. Plus précisément :

En imaginant que le **serveur** est sur le domaine `monBrowser.fr`, un cookie fonctionne de la façon suivante :

1. Le **serveur** envoie au **client** une réponse qui contient dans son en-tête le paramètre suivant (par exemple) :
Set-Cookie: visites=12.
2. Le **client** se souvient d'associer l'information visites=12 au nom de domaine monBrowser.fr.
3. Lors de ses prochaines requêtes (GET ou POST) vers google.fr, le **client** ajoutera le paramètre suivant dans l'en-tête de sa requête HTTP :
Cookie: visites=12

Le **serveur** peut donner des instructions au **client** sur la façon de traiter le cookie. En particulier, il peut lui dire de :

- renvoyer le cookie jusqu'à une certaine date et pas après. Si aucune date n'est précisée, le **client** continuera de renvoyer le cookie jusqu'à être fermé, par exemple en fermant la fenêtre du navigateur.
- renvoyer le cookie sur tous les chemins du domaine, voire d'autres domaines ou seulement sur un certain chemin. Si aucun chemin n'est précisé, le **client** enverra uniquement le cookie sur le répertoire courant.

Remarque Les cookies ont aussi des usages très discutables : ils peuvent en particulier servir à proposer de la publicité ciblée : vous allez sur un site mesChaussures.com, vous y regardez des modèles de sandales, et ensuite quand vous consultez un site d'information, vous avez des encarts publicitaires vous proposant des sandales de chez mesChaussures.com.

Cela sert aussi à une compagnie aérienne de se rendre compte que ça fait 3 fois que vous regardez le prix d'un vol Paris-Tahiti pour le 15 juillet, et d'augmenter le prix à chaque visite pour vous pousser à acheter plus vite.

En pratique dans PHP pour envoyer un cookie au **client**, on utilise la fonction

```
setcookie('clé', 'valeur');
```

et pour récupérer les cookies envoyés par le **client**, on consulte le tableau `$_COOKIE` comme on ferait pour les tableaux `$_POST` ou `$_GET`. Pour savoir si un **client** possède déjà un cookie, vous pouvez vous servir de

```
isset($_COOKIE['clé'])
```

comme vous aviez fait auparavant pour tester les contenus de GET et POST.

ATTENTION : la fonction `setcookie` doit être appelée avant tout affichage, c'est-à-dire avant tout `echo`, mais aussi avant tout code en dehors de `<?php ... ?>` (souvenez-vous que mettre du code en dehors de ces balises équivaut à faire un `echo`) : même un simple espace blanc avant `<?php` peut tout casser ! Ainsi, tous vos fichiers PHP qui se servent de cookies commenceront par du code ressemblant à ça :

```
<?php
...
/* Pas d'echo avant cette ligne !!!! */
if (!isset($_COOKIE['mon-cookie'])) {
    setcookie('mon-cookie', 'ma-valeur');
    ...
} else {
    ...
}
?>
<!-- Du code HTML ici, si vous voulez -->
```

On ne l'aura pas assez dit : pour envoyer un cookie au **client**, on se sert de `setcookie` ; faire

```
$_COOKIE['mon-cookie'] = 'ma-valeur'; //CODE INUTILE!!!!
```

n'aura aucun effet ! Ne confondez pas, d'une part : donner au **client** un cookie à conserver (par `setcookie`), et d'autre part : lire le cookie envoyé par le **client** (`$_COOKIE`).

Exercice 3 — Persistance par les Cookies

1. Créez un fichier `countCookie.php` qui se sert des cookies pour compter le nombre de visites. Cette fois-ci, vous n'êtes même pas obligés de mettre un lien ou un bouton vers la même page : rafraîchir la page suffira à renvoyer le cookie et en recevoir un nouveau incrémenté. Si ça ne marche pas, parce que vous refusez tous les cookies, allez dans “≡/Preferences/Privacy/&Security” et ajoutez une exception pour le site de la fac.
2. Par défaut en PHP, les cookies ont une durée de vie qui ne dépasse pas la session du **client** : si vous fermez votre navigateur, ils seront effacés. Essayez. La fonction `setcookie` vous permet de préciser une durée de vie plus longue grâce à un troisième paramètre : la syntaxe

```
setcookie('clé', 'valeur', time() + 3600 );
```




définit un cookie qui dure une heure (l'appel `time()` donne l'heure courante en secondes et le `+ 3600` ajoute une heure). Modifiez votre page pour que le cookie survive une heure malgré la fermeture éventuelle du navigateur.

3. Les cookies sont stockés par le navigateur. Ouvrez la même page avec deux navigateurs différents (Firefox et Konqueror, par exemple) et vérifiez que les deux compteurs sont indépendants.
Pour voir votre cookie, allez dans “≡/Web developer/Inspector/storage”. Cliquez droit sur le cookie, puis “delete ...”, rafraîchissez votre page, il doit afficher 1 visite.
Par ailleurs, vous pouvez voir tous les cookies mémorisés par votre navigateur : allez dans “≡/Preferences/Privacy/&Security”.
4. **Pour ceux qui vont vite.** Gagnez le million d'un collègue (Firefox ne vous permet pas d'éditer les cookies, mais vous pouvez aller voir du côté des extensions pour Firefox...).

Exercice 4 — Profil utilisateur (à commencer cette semaine si vous avez le temps)

Les cookies vont nous permettre de construire un premier site multi-utilisateurs.

Dans `public_html` créez un répertoire `monsie` où vous travaillerez pour la suite de cette partie.

-  Créez une page `inscription.php` contenant un formulaire d'inscription dans lequel on peut renseigner un nom d'utilisateur, un prénom, un nom, une date de naissance et une adresse e-mail. La validation du formulaire doit envoyer vers la page `sauvegarde.php` via le tableau POST.
-  Créez une page `sauvegarde.php` qui reçoit les informations sur l'utilisateur envoyées par POST et les vérifie (toutes les informations sont bien reçues, aucun champ n'est vide). Si les données sont valides, elle les sauvegarde dans autant de cookies et affiche le message « Inscription réussie » et un lien « Revenez à l'accueil » qui pointe vers `index.php` ; sinon elle affiche un message d'erreur et un lien pour retourner au formulaire d'inscription. Utilisez des cookies avec un temps de vie assez long (au moins la durée du TP, voire une semaine).
-  Créez une page `index.php` qui, **si l'utilisateur n'a pas les bons cookies**, affiche le message « Bonjour, vous n'êtes pas inscrit » et un lien « Créez votre compte » qui pointe vers `inscription.php`. Si par contre le **client** a bien envoyé les cookies, la page doit afficher « Bonjour Prénom Nom. Vous êtes né le ..., votre e-mail est ... ».
4. Modifiez `index.php` pour que, si l'utilisateur est bien reconnu, elle contienne un lien « Modifier mes données » qui pointe vers `inscription.php`
5. Modifiez `inscription.php` pour que, si le **client** est bien reconnu, **les champs soient pré-remplis avec les données de l'utilisateur** (vous devrez vous servir de l'attribut `value` des balises `<input>`).
6. Votre site est-il robuste ? Que se passe-t-il si vous effacez un seul des cookies de la mémoire du navigateur (par exemple, la date de naissance) ? Ne vous cassez pas trop la tête à régler ce genre de problèmes : on verra une structure plus robuste pour ça la semaine prochaine.

3 Redirection (optionnel)

Exercice 5 — Redirections

Modifiez la page `sauvegarde.php` pour qu'elle renvoie directement sur la page d'accueil si les données sont valides. Vous utiliserez l'appel de fonction

```
header('Location: index.php');
```

qui ordonne au navigateur de charger la page `index.php`. Comme pour `setcookie`, l'appel à la fonction `header` doit apparaître avant tout affichage.

Attention à ne pas confondre l'appel précédent avec la fonction `include` ! La fonction `header` permet d'envoyer au navigateur une requête de redirection : lorsque le navigateur la reçoit il génère une nouvelle requête pour la page spécifiée. Au contraire, avec `include` tout se passe sur le serveur : c'est lui qui va chercher le fichier et qui en copie le contenu dans la sortie de `sauvegarde.php`.

Nombre de forums sur PHP vous conseillent d'utiliser la fonction `header` pour tout un tas de choses ; néanmoins avec cette fonction les données font un aller retour de plus que si on s'était servis de `include`, ce qui ralentit le chargement de la page. `header` devrait être utilisée seulement pour rediriger vers des pages externes à votre site et quelques autres situations exceptionnelles ; si votre site est bien structuré vous pourrez toujours vous en passer au profit de `include`. Ceci n'est pas encore totalement satisfaisant : si un utilisateur ne passe pas par la page `index.php` et demande directement la page `sauvegarde.php`, des erreurs se produiront (essayez). La solution consiste à faire en sorte qu'aucune page, à l'exception de `index.php`, ne contienne d'instructions d'affichage. Ce sera l'objet d'un exercice du prochain TP.