Web - TP n^o 5
G. Subrenat, R. Zrour

Licence Informatique L3 - 2020-2021
UFR SFA - Université de Poitiers

Survol de PHP

Table des matières

1 Résumé

Dans ce TP nous verrons trois mécanismes fondamentaux des sites web dynamiques :
- variables de sessions
- cookies
- réception de formulaires

Pour chaque exercice ou question d'exercice, faites une page web à part pour être sûr que vos divers codes n'interfèrent pas.

Le site http://php.net contient, entre autre, la description des fonctions de PHP ; vous aurez besoin de vous y référer tout au long du TP.

2 Protocole HTTP

Point de cours : http

Le protocole HTTP est assez complexe et a évolué depuis ses débuts. Il ne s'agit pas dans ce cours de l'expliquer en détails. Si vous êtes intéressé, voici deux documents parmi tant d'autres :
- https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/definition-protocole-http/

C'est l'aspect "déconnecté" qui nous intéresse. Grossièrement, du point de vue du serveur, chaque requête ("chaque clic") est indépendante de la précédente. C'est problématique, notamment si le site nécessite une authentification : chaque changement de page devrait imposer une nouvelle authentification, ce qui est ingérable.

Il y a deux mécanismes principaux pour garder une mémoire des actions passées :
- les sessions (côté serveur)
- les cookies (côté client)

Point de cours : session

Le serveur doit garder en mémoire, entre deux requêtes, des informations sur l'internaute, par exemple :
- est-il authentifié ?
- qui est-il ?
- quels droits a-t-il ?
- ...

Ce type d'information doit être stocké sur le serveur de façon à ce que l'internaute ne puissent pas les modifier [a].

Une session est une zone mémoire, sur le serveur donc, associée à un internaute. Pour cela, lors de la première connexion, le serveur et le navigateur s'entende sur un identifiant (dit "identifiant de session" qui est souvent stocké dans un cookie à durée de vie limitée). Un identifiant ressemble à *uzpqnefof8tg5ntos8ugh49pq7*, et la session est affectée à cet identifiant.

Lorsque l'internaute change de page, le navigateur envoie systématiquement cet identifiant pour indiquer au serveur qui il est.
Donc le serveur, en se basant uniquement sur cet identifiant, cherche la session associée, la lit et peut savoir qui est connecté.

Pour être clair, vous pouvez vous connecter à un site en envoyant un identifiant au hasard. Si vous avez de la chance, vous avez généré un identifiant connu et vous usurpez l'identité d'une autre personne (c'est ce qu'on appelle le vol de session). La probabilité que ce type d'attaque (i.e. par génération aléatoire d'identifiant) fonctionne est nulle [b] :

- le nombre de combinaisons est “grand” [c]
- un identifiant a une durée de vie limitée (24 minutes généralement, et beaucoup moins pour les sites sensibles)

Et on peut rajouter d’autres points de vérification : le plus classique étant l’IP de l’internaute.

Comme indiqué ci-dessus, au bout d’un temps d’inactivité, la session est détruite et l’internaute doit se ré-authentifier.

a. si un internaute pouvait choisir les droits qu’il a sur un site, ...
b. i.e. tellement faible qu’elle est considérée comme nulle
c. $36^{26} = 2909812598873150618315302561643530656153 6$ dans le cas de l’exemple fourni

Point de cours : cookies

Les cookies permettent également, entre deux requêtes, de garder des informations.
Ils sont stockés côté client, i.e. par le navigateur, sous forme de petits fichiers.

Ils sont donc modifiables par l’internaute. Le serveur ne doit pas y mettre des informations qui mettent en péril sa sécurité (comme les droits de l’internaute). De plus le serveur doit gérer le fait que les cookies peuvent disparaître ou être altérés.

Outre le lieu de stockage et la finalité, il y a deux différences avec les sessions :

- leurs durée de vie peuvent être très longues (plusieurs mois)
- les informations demeurent après la fermeture du navigateur.

Voici des informations que l’on peut classiquement stocker dans des cookies :

- la langue de préférence
- le mot de passe de l’utilisateur pour lui éviter de se réauthentifier (mais dangereux, notamment si plusieurs personnes accèdent à la même machine avec le même compte [a]).

a. ce qui est également très risqué

Point de cours : et au-delà

Les sessions ont une durée de vie très limitée. Comment fait-on, lorsqu’on veut garder des informations côté serveur sur une longue durée ?

Réponse : on utilise une base de données.

3 Variables super-globales

Lorsque, dans une fonction, on veut utiliser une variable globale il faut la déclarer afin que l’interpréteur ne la considère pas locale.

Exemple de variables globales (on rappelle tout de même que les variables globales sont à éviter) :

global.php

```php
<?php
    // variables globales
    $miaou = 'chat';
    $ouah = 'chien';

    function f()
    {
        global $miaou;
        // $miaou est bien la variable globale
        echo $miaou . "\n";
        // en revanche $ouah est considéré comme locale et il y a un warning
        echo $ouah . "\n";
    }

    f();
```

Il existe des variables globales prédéfinies qui n'ont pas besoin d'être déclarées pour être utilisées ; elles sont qualifiées de "super-globales".

superglobal.php

```php
<?php
    function f()
    {
        // inutile de la déclarer en global
        print_r($_COOKIE);
    }

    f();
```

Affichez, avec *print_r*, les variables super-globales suivantes :

- `$_SERVER`, `$_ENV`, `$_REQUEST`
- `$_SESSION`, `$_COOKIE`
- `$_POST$`, `$_GET`, `$_FILES`
- `$GLOBALS$`

Finissez la page en appelant la fonction `phpinfo`.

Si vous n'avez pas fait de zèle, vous devez avoir un warning avec la variable `$_SESSION`. C'est objet de l'exercice suivant.

4 Session et cookies

Les trois exercices de cette section sont indépendants ; un squelette de code est fourni.

4.1 variables de session sans session

a) Faites une mini-application de 2 pages. La première injecte des cases dans le tableau `$_SESSION`, par exemple :

— *nom => "tartempion"*

— *age => 23*

Le code modifiant `$_SESSION`, doit se trouver en tout début de la page avant le "doctype".

Cette page contient un lien vers une deuxième page PHP qui elle testera l'existence des deux cases et affichera le cas échéant les valeurs.

Les deux pages afficheront également l'intégralité des tableaux `$_SESSION` et `$_COOKIE`.

Attention, dans cette première partie de la question on ne crée pas de session (i.e. pas de *session_start()*).

Tout se passe-t-il comme prévu ?
(spoil : le second fichier ne récupère pas les deux valeurs)

b) Puis faites un *session_start()* dans le premier fichier. Qu'affiche le second fichier ?
(spoil : le second fichier ne récupère toujours pas les deux valeurs)

c) Enfin faites un *session_start()* également dans le second fichier et vérifiez que le fonctionnement est celui voulu.

4.2 variables de session avec session

Point de cours : paramètre GET

Un paramètre GET est une valeur transmise d'une page à une autre via l'URL : la valeur est précisée explicitement dans l'URL.

Syntaxe :

`http://<serveur>/<chemin>/<fichier php>?<var1>=<val1>&<var2>=<val2>&...&<varn>=<valn>`

Exemple : on veut exécuter le fichier *cible.php* qui se trouve dans le répertoire *travail/ville* du serveur *monsite.com* ; et on veut transmettre trois informations :

- le code postal sous l'identifiant *code*
- le nom de la ville sous l'identifiant *nom*
- le nombre d'habitants sous l'identifiant *nb*

url : `http://monsite.com/travail/ville/cible.php?code=86000&nom=Poitiers&nb=90000`

Dans le fichier *cible.php*, les trois valeurs sont récupérées dans la variable `$_GET` : `$_GET['code']`, `$_GET['nom']` et `$_GET['nb']`

Faites une mini-application de 2 pages en utilisant les sessions pour transmettre les variables de session. A priori, seule la seconde page nécessite de gérer les sessions : la première page n'est qu'un ensemble de 5 liens vers la seconde page.

Les deux pages afficheront également l'intégralité des tableaux `$_SESSION` et `$_COOKIE`. Observez à quel moment un cookie apparaît (on parle ici du cookie de session).

La première page est un menu qui comporte 5 liens, tous vers la seconde page mais avec un paramètre GET différent :

— lien 1 : vers la seconde page avec le paramètre *action=1* passé en paramètre *GET*.
— lien 2 : vers la seconde page avec le paramètre *action=2* passé en paramètre *GET*.
— ...
— lien 5 : vers la seconde page avec le paramètre *action=5* passé en paramètre *GET*.

La seconde page propose un lien vers la première et a une action dépendant du paramètre *action* :

— *action* = 1 : on modifie les cases *nom* et *age* de `$_SESSION`,
— *action* = 2 : on modifie les cases *nom* et *age* de `$_SESSION` avec deux autres valeurs,
— *action* = 3 : on affiche, avec tests d'existence, les cases *nom* et *age* de `$_SESSION`, ainsi que l'intégralité du tableau `$_SESSION`,
— *action* = 4 : on supprime la case *nom* de `$_SESSION`,
— *action* = 5 : on détruit la session (fonction *session_destroy*).

Faites un semblant de sécurité et assurez-vous que le paramètre *action* passé en *GET* est correct (i.e. est bien un nombre et avec une valeur correcte).

Tout se passe-t-il comme prévu ?

Quittez et relancez le navigateur. Que sont devenues vos variables de session ? Et qu'est devenu le cookie ? [1]

4.3 cookies

Point de cours : cookies

Les cookies sont des petits fichiers présents sur le poste du client et gérés par le navigateur.

Il faut juste être conscient des problèmes de sécurité :

- ils peuvent être modifiés par l'internaute
- ils peuvent être supprimés par l'internaute

Cependant c'est bien sur directives du serveur que les cookies sont manipulés (CRUD : création, lecture, modification et destruction) :

- la fonction *setcookie* gère la création, la modification et la destruction,
- la variable `$_COOKIE` permet la lecture

Un premier point à bien comprendre : c'est le serveur qui donne les ordres d'altération, mais c'est le navigateur qui les exécute.

1. Sur mon navigateur, la session persiste ce qui me semble anormal : le cookie de session doit être détruit lorsque le navigateur se ferme.

Autrement dit lorsque PHP fait un appel à *setcookie*, l'exécution ne sera faite qu'à la fin du script lorsque le serveur enverra toutes les informations au navigateur.
Et donc la modification du cookie ne sera effective que sur la consultation de la page suivante. Il y a un un temps de décalage.

Lors de l'appel à *setcookie*, on précise la date d'expiration (et non pas la durée de vie). Cette date est un *timestamp*, c'est à dire le nombre de secondes depuis le 1er janvier 1970.
Pour détruire un cookie, on précise une date d'expiration dans le passé.

Attention : la date d'expiration est précisée par le serveur en fonction de son horloge ; mais c'est le navigateur (sur le poste du client donc) qui analyse cette date pour prendre la décision d'effacer le cookie ou non. Selon le décalage des horloges entre les deux machines, on peut avoir des comportements erronés.

Il existe une date d'expiration particulière : le cookie se détruira avec la fermeture du navigateur. C'est ce que l'on appelle un cookie de session (qu'il ne faut pas confondre avec les sessions du serveur ; mais pour perturber les esprits, c'est un cookie de session qui est utilisé par les sessions du serveur).

Cf. le manuel de la fonction *setcookie*.

Faites une mini-application de 2 pages :

- lien 1 : vers la seconde page avec le paramètre *action=1* passé en paramètre *GET*.
- lien 2 : vers la seconde page avec le paramètre *action=2* passé en paramètre *GET*.
- ...
- lien 5 : vers la seconde page avec le paramètre *action=5* passé en paramètre *GET*.

La seconde page propose un lien vers la première et a une action dépendant du paramètre *action* :

- *action* = 1 : crée (en mettant la valeur 0) ou modifie (en incrémentant la valeur courante) le cookie nommé *compteur* ; le cookie doit expirer au bout de 24 heures,
- *action* = 2 : crée ou écrase un cookie nommé *nombres* qui contient un tableau de trois cases *(un => one, deux => two, trois => three)* ; ce(s) cookie(s) doi(ven)t expirer avec la fermeture du navigateur,
- *action* = 3 : affiche avec tests d'existence les cookies sus-cités,
- *action* = 4 : détruit le cookie *compteur*,
- *action* = 5 : détruit le(s) cookie(s) *nombres*.

Dans tous les cas, après vos actions, affichez le tableau `$_COOKIE` pour vérifier le bon comportement de vos actions.
De même avec votre navigateur regardez la liste des cookies présents et supprimez-en manuellement pour voir comment réagissent vos pages.

Tout se passe-t-il comme prévu ? Notamment n'avez-vous pas l'impression qu'il y a un temps de retard pour la prise en compte de l'affectation des cookies (cf. note de cours) ? Un tableau de cookies est-il réellement un tableau (en mémoire et sur disque) ?

5 Formulaires

On commence ici à simuler une application un peu plus complète.
Les trois exercices de cette section sont indépendants ; un squelette de code est fourni.

5.1 création du questionnaire et test

Le formulaire suivant vous est fourni :

+ un champ texte
- un champ texte en lecture seule, initialisé par une valeur par défaut

+ un champ mot de passe
— une zone de texte multi-lignes
— une zone cachée
+ un groupe de trois boutons radio dont aucun n'est pré-selectionné
— un autre groupe de trois boutons radio dont un est pré-sélectionné
+ une série de cinq cases à cocher, dont deux sont pré-sélectionnées
+ une liste déroulante avec 3 groupes de deux (ou trois) entrées chacun
— une liste semblable mais dans un cadre avec ascenseur
+ une liste semblable, toujours avec un ascenseur, et avec une possibilité de choix multiples
+ un champ caché limitant la taille du fichier à environ 50 Ko
+ un bouton de sélection de fichier
— le bouton d'envoi
— le bouton de réinitialisation

Vous devez compléter les attributs des différentes balises pour que la soumission du formule ressemble à :

Affichage des valeurs par POST

```
Array
(
    [nom] => Ada Lovelace
    [nom_interne] => 0xaff23df03
    [passwd] => azerty
    [cv] => la reine des mots de passe robustes
    [cache] => test caché
    [radio1] => Mme
    [radio2] => 18-65
    [cb1] => Array
        (
            [0] => info
            [1] => jeux
        )

    [liste2] => france
    [liste3] => france
    [liste4] => Array
        (
            [0] => info
            [1] => jeux
            [2] => theatre
        )

    [MAX_FILE_SIZE] => 50000
)
```

Affichage des valeurs par FILES

```
Array
(
    [fichier] => Array
        (
            [name] => traitement.php
            [type] => application/x-php
            [tmp_name] => /tmp/phpTgMksB
            [error] => 0
            [size] => 545
        )

)
```

5.2 analyse du questionnaire

Récupérez, dans un autre fichier PHP, les données du questionnaire préfixées par un + et affichez la ou les valeurs de chacun de ces champs dans un tableau HTML. Gérez le fait qu'une variable peut ou non être présente : c'est à dire indiquez explicitement si un champ n'est pas présent ou est vide.
On demande de gérer les champs les uns après les autres dans le code (pas de parcours automatique du tableau `$_POST`).
En outre affichez toutes les données relatives au fichier téléchargé, si le téléchargement a bien eu lieu sans encombre.

Ensuite affichez violemment les variables `$_GET`, `_POST` et `_FILES` pour vérifier que tout est correct.

5.3 analyse du questionnaire

Point de cours : gestion des erreurs d'un formulaire

Les erreurs de saisies dans un formulaire peuvent être gérées à deux niveaux :
- dans le navigateur avec HTML et JavaScript : l'intérêt est que les erreurs sont détectées avant l'envoi du formulaire ; on est ainsi plus réactif et on économise de la bande passante.
- au niveau du serveur avec PHP à la réception du formulaire.

L'idéal est de le faire aux deux endroits, même s'il est curieux de faire le travail deux fois.

Si possible il faut le laisser au niveau du navigateur pour les avantages cités ci-dessus.

Mais un internaute peut modifier le code de la page ou désactiver JavaScript ; donc lorsque le serveur reçoit le formulaire il ne peut pas être assuré que les vérifications ont été faites. Par conséquent il doit les (re)faire.

Si on ne doit garder qu'une seule vérification, pour des raisons de sécurité, c'est celle du serveur qui doit être choisie.

Nous partons maintenant d'un questionnaire plus court pour lequel il faut gérer les erreurs.

Le questionnaire a trois informations :
- nom (vide par défaut)
- mot de passe (vide par défaut)
- état civil (aucun radio-bouton coché par défaut)

Les contraintes sont les suivantes :
- le nom ne doit pas être vide
- le mot de passe doit comporter au moins 3 caractères
- il faut sélectionner un des radio-boutons.

Voici les directives :
- le fichier *php* qui affiche le formulaire et le fichier qui le reçoit et le traite sont le même.
- la première fois où le fichier est appelé, le formulaire s'affiche vide
- si le formulaire est correctement rempli, on écrit son contenu dans un fichier [2] (fonctions *fopen, fwrite, ...*), puis on fait une redirection vers une autre page (fonction *header*) qui affiche un message de réussite ainsi que le contenu du fichier précédement écrit.
- si le formulaire est erroné, il doit se réafficher avec les champs précédemment saisis toujours présents (pour que l'internaute n'ait pas à resaisir l'entièreté du formulaire). En outre un texte en rouge (penser au CSS) doit apparaître à droite des champs erronés.
  Pour le mot de passe, on ne le remplit pas avec la valeur saisie, mais on le laisse vide (pour des raisons de sécurité).

6 Envoi de fichier (facultatif)

Faites un formulaire qui demande :
- un pseudo : chaîne de 8 caractères au maximum, tous alphabétiques,
- un fichier image : 100 ko au maximum, obligatoirement au format *jpg* ou *png*.

Le script PHP doit vérifier que le pseudo et l'image vérifient les critères. Si c'est le cas, il doit **rediriger** (avec la fonction *header*) vers une nouvelle page qui affichera le pseudo et l'image. Si ce n'est pas le cas, le formulaire est réaffiché, pré-rempli avec les informations précédemment saisies (est-ce possible avec le champ récupérant le nom de fichier ? [3]) et un message d'erreur précisant les corrections à apporter pour chaque champ invalide.

2. assurez qu'il y a les droits nécessaires
3. Si l'on pose la question, c'est que la réponse est négative. La vraie question est donc : pourquoi ne peut-on pas préremplir ce champ ?

Note : le script PHP affichant le formulaire et le traitant doit être le même.

Pour aller un peu plus loin, l'URL de la page qui affiche ces données doit pouvoir être sauvegardée et réutilisée ultérieurement (plusieurs jours après) et afficher les mêmes informations (i.e. pseudo et image). Pour cela on utilise un, et un seul, paramètre GET qui est le pseudo.
Si plusieurs images sont envoyées avec le même pseudo, c'est la dernière qui est conservée.

Extensions possibles : imposer une définition maximale de l'image, retailler une image trop grande, ...

Voici quelques fonctions utiles : *exif_imagetype*, *is_uploaded_file*, *move_uploaded_file*, *rename*, *getimagesize*, *unlink*, *is_file*, *imagecrop*, ...