



R3.01 - ProgWeb

TD 2 - Formulaires & Persistance

Introduction

Dans certains exercices vous verrez apparaître :

-  → Vous ne devez pas utiliser ces mots-clés ou ces fonctions !
-  → Fonctions ou mots-clés suggérés. Consultez le site **php.net**. Pas d'obligation de les utiliser si vous (et pas Google) trouvez une autre façon de faire.

Comme pour le TD 1, vous lancerez un **php -S localhost:8888** pour servir de mini serveur local de test durant ce TD.

Formulaire côté client (HTML)

Les formulaires côté client ne sont pas vus en R1.02 car ils vont de pair avec le traitement côté serveur, que nous ne voyons que cette année.

Leur mise en œuvre dans le code HTML ne présente pas de grande difficulté, c'est pourquoi nous n'allons pas nous étendre sur le sujet ici pour nous attarder plutôt sur le traitement côté serveur, en PHP.

Vous trouverez sur Moodle, aux côtés de ce sujet de TD, un PDF détaillant tous les champs utilisables, ainsi que leur syntaxe. Ci-dessous, quelques extraits :

tags impliqués dans la création d'un formulaire :

- Le conteneur : **<form>...</form>**
- Les champs de saisie, radio-boutons, cases à cocher, mots de passe : **<input... type=...>** où **type** indique la nature du champ (**text**, **checkbox**, **radio**, **password**). C'est un tag auto-fermant (pas de **</input>**).
Il y a aussi **file** qui est un type particulier, qui sera décrit en détail plus loin.
- Le texte multiligne : **<textarea>...</textarea>**
- La liste déroulante : **<select>...</select>** avec les valeurs de la liste sous forme de **<option>...</option>**
- Le bouton : **<button>...</button>**
- Le label : **<label>...</label>** qui est associé à un champ

Le tag **<form>** :

- Est un conteneur des champs et boutons.
- Contient un attribut **action** qui indique l'URL appelée à la soumission du formulaire
- Contient un attribut **method** qui indique la façon dont sont transmises les valeurs des champs du formulaire. Deux valeurs possibles pour **method** : **GET** et **POST**.

Dans le cas des formulaires Web, le choix entre **GET** et **POST** est dicté principalement par la nature des champs soumis et le volume de données qu'ils peuvent représenter (attention au **<textarea>**).

Pour quelques champs texte dont la longueur totale des valeurs cumulées est assez courte (maximum quelques centaines de caractères, < 2000), vous pouvez opter pour la méthode **GET**. Au-delà, choisissez la méthode **POST**.

Si vous soumettez des fichiers, ce sera obligatoirement en méthode **POST** et ce cas sera étudié spécifiquement plus tard.

Attention, dans le cas de la méthode **POST**, le formulaire soumis vous fait arriver sur un script PHP (qui est donc une page Web) qui, si vous tentez ensuite de la rafraîchir manuellement (**CTRL+R** ou clic sur l'icône de rafraîchissement du navigateur) va provoquer l'affichage d'un message assez ennuyeux et toujours sujet à interrogation (y a-t-il un risque à rafraîchir ou pas ?). Exemple pour Chrome :

La page que vous recherchez a utilisé des informations que vous avez envoyées. Si vous revenez sur cette page, chaque action précédemment effectuée sera répétée. Souhaitez-vous continuer ?

C'est pourquoi on peut hésiter à utiliser la méthode **POST**, mais parfois il n'y a pas le choix. Il y a une solution pour éviter ce genre d'inconvénient, on va la voir plus loin.

Notez aussi que la méthode **GET** n'est pas toujours une solution idéale non plus car elle dévoile en clair, dans l'URL appelée, tous les champs soumis et notamment les éventuels champs **<input type="password">**. Pas toujours idéal pour la sécurité !

Ce n'est pas que la méthode **POST** soit plus secrète mais trouver l'information requiert un peu d'investigation à l'aide des outils de développement des navigateurs. Disons qu'un **POST** est plus discret dans ce cas.

Vous trouverez une explication plus circonstanciée de ces deux méthodes (**GET** et **POST**) en R4.01 quand on abordera les Web Services.

La gestion, côté serveur, des données soumises est expliquée dans la partie suivante.

Formulaire côté serveur (PHP)

La méthode **GET** :

Côté client, les champs (noms et valeurs) sont placés dans l'URL, à la suite de la valeur fixée par l'attribut **action**, formant ainsi une chaîne d'URL du style de celles que vous avez déjà utilisées dans les **<a href...>** des TD 1 et TP 1 pour gérer la pagination d'une liste. Exemple :

http://localhost:8888/script.php?champ1=valeur1&champ2=valeur2&...

Ainsi, puisqu'un formulaire utilisant la méthode **GET** effectue un appel d'URL où figurent les champs, vous savez donc, depuis le TD 1, que ces champs sont accessibles via la superglobale **\$_GET**.

La méthode **POST** :

Côté client, l'URL donnée par l'attribut **action** est appelée sans aucune modification.

Les champs (noms et valeurs) sont transmis dans le corps de la requête HTTP (rien à voir avec le **<body>** d'une page HTML). Le détail de la façon dont fonctionne un protocole, de ce qu'est une requête HTTP et de la façon dont sont envoyées les données (ici les champs du formulaire) au serveur, sera vu en R3.05, ressource de SAÉ 3.

Pour le moment, reprenez juste que les champs passés par un formulaire utilisant la méthode **POST**, sont accessibles via la superglobale **\$_POST**.

Note importante : si vous placez, dans l'attribut **action**, une URL contenant aussi une partie **?champX=valeurX&champY=valeurY&...**, les champs de cette partie, puisque figurant sur l'URL, seront accessibles dans la superglobale **\$_GET**, même si la méthode de votre formulaire est un **POST** ! Dans ce cas, vous aurez deux sources de données, à vous de savoir où se trouvent les informations.

Nommage et type des données dans **\$_GET** et **\$_POST**

Peu importe la nature et le contenu des champs du formulaire côté client, les données reçues par le script PHP (attribut **action**) sont toutes des chaînes de caractères.

Ainsi **\$_GET** et **\$_POST** sont des tableaux associatifs dont les clés sont les **noms** des champs (attribut **name** d'un champ du formulaire) et les **valeurs** sont des chaînes de caractères qui représentent les valeurs associées à chaque champ.

Ceci signifie que si vous nommez plusieurs champs du même nom, ce qui est en principe interdit même si le navigateur n'applique en réalité aucune vérification ni restriction, seule la dernière valeur associée à un nom donné sera présente dans le tableau **\$_GET** ou **\$_POST**. Comme indiqué un peu avant, n'oubliez pas que les valeurs passées sur l'URL dans le champ **action** sont aussi concernées, mais vous pouvez avoir les mêmes noms

si l'une va dans **\$_GET** et l'autre dans **\$_POST**. Mais cela est une très mauvaise idée, évidemment !

Exception à la règle, sauce "PHP"

Il vient d'être expliqué qu'un nom de champ doit être unique.

Cependant, comme le navigateur ne l'interdit pas, rien ne vous empêche de le faire et il y a un cas pratique et acceptable où l'on peut exceptionnellement nommer plusieurs champs du même nom, à la condition que vous codiez en PHP côté serveur (peut-être que d'autres langages l'autorisent aussi, à vérifier) et que vous respectiez la convention suivante.

En nommant plusieurs champs d'un même nom terminé par **[]**, vous permettrez à PHP de comprendre que plusieurs champs peuvent avoir le même nom et qu'il doit créer **dans \$_GET** ou dans **\$_POST**, une entrée associant le nom en question, non pas à une valeur mais à un tableau de valeurs (celles des champs portant ce nom).

Exemple :

```
<form action="traite.php" method="GET">
  Votre nom : <input type="text" name="nom"><br />
  Quelles langues parlez-vous ?<br />
  <input type="checkbox" name="lang[]" value="FR">Français
  <input type="checkbox" name="lang[]" value="EN">Anglais
  <input type="checkbox" name="lang[]" value="IT">Italien
  <input type="checkbox" name="lang[]" value="JP">Japonais
</form>
```

La soumission au script **traite.php** alimente la superglobale **\$_GET** ainsi :

Array

```
(
  [nom] => DUPONT
  [lang] => Array (
    [0] => FR
    [1] => JP
  )
)
```

Dans le tableau associé à **lang** ne figureront que les valeurs des cases cochées. Notez aussi que la clé **lang** ne contient plus les **[]** qui figuraient dans le nom des champs, PHP comprend qu'on voulait un tableau et adapte aussi le nom.

Cette technique de nommage est principalement utilisée dans le cadre de case à cocher, on en comprend bien l'idée et plus rarement dans le cadre de champs sans relation entre eux. On préférera alors nommer les champs différemment dans ce cas.

Exercice 1 - Sélecteur de région

Lisez l'exercice jusqu'au bout, il est découpé en **Questions**.

Reprenez les sources de données **regions.php** et **depts-fix.php** du TD 1.

Vous allez créer un script **form-reg.php** contenant un formulaire qui :

- Affiche un champ de libellé **Nom dép.** en saisie libre
- Affiche un champ de libellé **Région** affichant la liste des régions avec une valeur par défaut indiquant **Choisissez une région**
- Soumet à un script **liste-depts.php**

et le script **liste-depts.php** qui affiche (sans tri) la liste de départements () ciblés en fonction des critères suivants :

- Les champs **Nom dép.** et **Région** sont combinables
- Le champ **Région** seul : affiche la liste des départements de la région choisie
- Le champ **Nom dép.** seul : filtre et affiche les départements en fonction de la présence, quelque part dans le nom, de la chaîne de caractères saisie
- Les champs **Nom dép.** et **Région** ensemble : filtre et affiche les départements de la région sélectionnée en fonction de la présence, quelque part dans le nom, de la chaîne de caractères saisie
- Si rien n'a été sélectionné ni rempli (des espaces seuls ne représentent pas un remplissage), un message d'erreur, formaté pour être bien visible et explicite (i.e. : un message qui précise ce qui ne va pas dans les données soumises) sera présenté à l'utilisateur.
- Un lien de retour vers la page du formulaire devra être présent en haut de page.

Question 1

Quelle méthode de soumission allez-vous choisir ? Pourquoi ?

Question 2

Réalisez le code HTML de **form-reg.php** en utilisant un peu de CSS pour faire ressurgir vos qualités artistiques profondément enfouies.

Réalisez le code PHP du script **liste-depts.php**.



- Pensez à utiliser **print_r()** pour vous familiariser avec le contenu de la superglobale qui est remplie lors de la soumission.
- La page <https://www.php.net/ref.strings> sur **php.net** (votre bible pour PHP) fourmille de fonctions utiles sur les chaînes de caractères.
- Pensez aussi que min et MAJ sont différentes.

Question 3

Ajoutez à votre script la possibilité de filtrer aussi sur le numéro de département. Ce filtre ne devra s'appliquer que si la saisie fait exactement la bonne taille. Dans ce cas, le filtre sur la région sera inactivé et un message affichera au dessus de la liste :

Recherche sur le code département "<code>".

Le filtrage par région est donc inactif.

Exercice 2 - Paramétrage de PHP

PHP possède de nombreux paramètres qui régissent son comportement et ses capacités.

PHP est compilé initialement avec des valeurs par défaut pour tous ses paramètres, mais ils sont ensuite adaptables au démarrage de PHP qui lit un fichier de configuration qui s'appelle **php.ini**, dans lequel il est possible de spécifier d'autres valeurs pour tout ou partie des paramètres.

Si le fichier **php.ini** n'existe pas ou ne contient pas de valeur spécifique pour un paramètre donné, c'est la valeur par défaut, fixée lors de la compilation, qui est prise en compte.

Certains paramètres peuvent aussi être modifiés en cours de fonctionnement de PHP, soit temporairement et localement, par un script, le temps de son exécution, soit définitivement pour tous les scripts s'exécutant depuis un dossier donné, en plaçant un fichier de configuration dédié dans le dossier en question.

Afficher les paramètres actifs

Il existe une fonction PHP qui affiche à l'écran tous les paramètres et les variables internes de PHP :

```
phpinfo();
```

Question 1

Créez un script **info.php** qui exécute cette fonction et testez-le dans un navigateur. La fonction effectue elle-même l'affichage, pas besoin de faire un **echo**.

Question 2

Parmi ces informations, localisez la version de PHP.

Question 3

Dans la section **PHP Variables**, ce ne sont pas des paramètres mais des variables (oui c'est marqué dessus !) et vous devez y voir quelques superglobales dont on n'a pas encore parlé, comme **\$_SERVER** et **\$_ENV**.

Vous y trouverez des informations utiles que vos scripts peuvent exploiter si besoin. Notez par exemple :

- **\$_SERVER['SCRIPT_FILENAME']** qui vous donne le chemin complet où se trouve votre script dans l'arborescence sur le serveur.
- **\$_SERVER['REQUEST_URI']** qui est l'URL qui a été utilisée pour appeler le script.
- **\$_SERVER['HTTP_USER_AGENT']** qui vous permet d'identifier le type d'ordinateur, d'OS et de navigateur, que le client utilise pour afficher votre page. Ça permet éventuellement d'adapter la construction de la page en conséquence, même si les media queries CSS permettent aujourd'hui de gérer cela de façon plus universelle du côté client.

Question 4

Au début du **phpinfo()**, localisez **Configuration File (php.ini) Path** qui vous donne le chemin du **php.ini** sur votre machine. Ce chemin est codé en dur lors de la compilation. Ainsi, si un fichier **php.ini** est trouvé sur ce chemin et que rien n'est indiqué à PHP pour qu'il en prenne un autre au démarrage, ce sera celui-ci qui sera lu et pris en compte.

Question 5

Localisez maintenant **Loaded Configuration File** qui vous donne le chemin du **php.ini** réellement pris en compte au démarrage. Si vous y voyez **(none)**, ça signifie qu'aucun **php.ini** alternatif n'a été spécifié au démarrage de PHP. C'est peut-être le cas pour vous si vous avez lancé un **php -S**.

Localisez aussi le paramètre **max_execution_time** dans la section **Core**, exprimé en secondes.

Quelle est sa valeur ?

Il s'agit du temps maximum alloué à un script pour s'exécuter. S'il dépasse ce temps, PHP l'arrêtera. Ce temps est suffisamment long pour la très grande majorité des scripts mais il peut s'avérer insuffisant dans certaines situations, notamment pour des scripts autonomes qui traitent de gros volumes de données. Nous allons le modifier.

Faites maintenant les actions suivantes dans le Terminal qui fait actuellement tourner votre **php -S** :

- Arrêtez votre serveur par **CTRL+C**
- Créer un fichier **php.ini** contenant une seule ligne : **max_execution_time=1200**
- Lancez un **php -S localhost:8888 -c ./php.ini**
- Rafraîchissez votre page **info.php**

Localisez une nouvelle fois **Loaded Configuration File** et **max_execution_time**.

Quelles sont les nouvelles valeurs ?

Nous n'allons pas entrer en détail dans tous ces paramètres. Vous trouverez la liste exhaustive des paramètres du fichier **php.ini** ici :

<https://www.php.net/manual/fr/ini.list.php>

Observez la colonne **Modifiable** dans cette liste. Elle peut prendre 4 valeurs dont les explications sont données ici :

<https://www.php.net/manual/fr/configuration.changes.modes.php>



Important pour la sécurité 🔥 ☠️ !!

ATTENTION, il est fréquent de se créer un script qui exécute **phpinfo()** pour vérifier les paramètres actifs du PHP installé sur un serveur.

Il est aussi fréquent d'oublier de le supprimer !!!

La fonction **phpinfo()** donne certaines infos très sensibles, notamment sur les chemins où sont placés les scripts, ou encore sur les modules installés, tels que les BDD. On peut aussi y trouver les variables d'environnement qui contiennent parfois des identifiants et des mots de passe de connexion à des services (BDD ou autres), même si cette pratique n'est pas recommandée.

Il est même assez commun de nommer ce script **phpinfo.php**, et les hackers n'ont pas long pour obtenir ces informations à votre insu si vous oubliez de le supprimer ou *a minima* de mettre en commentaire l'appel à **phpinfo()**. Pensez à le faire dès que vous n'en n'avez plus besoin !

Notez aussi que placer le **php.ini** dans le même répertoire que les scripts pour le lancement de PHP avec un **-c**, en terme de sécurité, ce n'est pas bon du tout ! Il faut impérativement que ce fichier **php.ini** soit inaccessible, de l'extérieur, par une URL, comme on l'a déjà évoqué en TD1. OK pour des tests mais pas plus...

Exercice 3 - Envoi de fichiers

Contexte

Lisez l'exercice jusqu'au bout, il est découpé en **Questions**.

Pour cet exercice, vous allez mettre en place des pages PHP permettant de :

- Soumettre des fiches signalétiques d'individus dont on récupérera les informations suivantes :
 - Nom
 - Prénom
 - Adresse mail

- Photo
- Afficher une liste (<table>) de ces individus avec les photos

Pour générer des photos, vous pourrez vous rendre sur <https://avatarmaker.com> mais n'y passez pas trop de temps !

Formulaire spécial

L'envoi de fichier met en œuvre un type particulier de formulaire et une méthode de traitement spécifique du côté serveur.

Côté client, le <form> doit soumettre en méthode **POST** et doit avoir un attribut supplémentaire :

enctype="multipart/form-data"

Côté serveur, les champs "normaux" du formulaire sont à traiter comme pour un formulaire traditionnel (avec les **\$_GET** et **\$_POST**).

Pour les fichiers transmis, il faut savoir ceci :

- La superglobale **\$_FILES** contient une entrée par fichier reçu. Chaque entrée possède les attributs suivants :
 - **name** : nom du fichier d'origine sur l'ordinateur du client
 - **type** : type MIME du fichier. Permet de vérifier que le client a soumis un fichier d'un format attendu.
 - **tmp_name** : le chemin du fichier temporaire créé sur le serveur
 - **error** : le code d'erreur éventuel si quelque chose s'est mal passé (0 = OK)
 - **size** : la taille du fichier soumis.
- Les fichiers reçus sont placés par PHP dans un dossier temporaire, souvent **/tmp**, et cette information figure dans chaque entrée de **\$_FILES** sous la clé **tmp_name** (voir ci-dessus)
- Le fichier doit être immédiatement déplacé, par le script, de son emplacement temporaire vers un emplacement définitif. Quand le script termine son exécution, si le fichier temporaire n'a pas été déplacé, il est supprimé par PHP, pour des raisons de sécurité et de nettoyage de **/tmp**.
Le déplacement se fait grâce à la fonction **move_uploaded_file()**.

Vérification des paramètres PHP

Dans le cas de l'envoi de fichiers, les paramètres de PHP à vérifier avant tout sont :

- **file_uploads = On** (Section **Core**)
- **max_file_uploads** (Section **Core**), qui est le nombre maximum de fichiers qu'un script peut accepter sur une soumission de formulaire.
- **upload_max_filesize** (Section **Core**), qui est la taille maximum autorisée par fichier.

- **post_max_size** (Section **Core**), qui est la taille maximum globale autorisée pour un **POST**. La somme des tailles des champs et des fichiers ne peut pas dépasser cette valeur.

Question 1

Vérifiez la présence et les valeurs de ces paramètres.

Sont-ils compatibles avec vos besoins ?

Par qui sont-ils modifiables si besoin ?

Question 2

Créez une page Web **fiche.html** contenant un formulaire permettant la saisie des informations d'une fiche signalétique et l'envoi d'un avatar.

Votre page doit soumettre le formulaire à un script **enreg.php**.

Dans un premier temps, ce script affichera uniquement un **print_r()** de la superglobale **\$_FILES**.

Exécutez, soumettez une fiche et observez le contenu de **\$_FILES**.

Question 3

Les fichiers soumis par formulaire arrivent dans **\$_FILES** avec l'information de leur nom originel sur la machine de l'utilisateur. Il peut être tentant de conserver ce nom pour le fichier conservé sur le serveur, mais ce n'est pas forcément une bonne idée car le nom peut contenir des caractères, des espaces et des symboles qui ne sont pas adaptés à l'OS de votre serveur ou que ne seront pas aisés à manipuler ensuite.

Il est donc conseillé de renommer ces fichiers plus simplement, en utilisant, par exemple, la clé d'enregistrement dans une BDD qui contiendrait les informations sur le fichier réceptionné. Parmi ces informations en BDD, vous pourriez aussi stocker le nom originel du fichier, pour en conserver trace.

Comme vous n'avez pas de BDD pour cet exercice, vous allez procéder différemment pour obtenir un identifiant numérique simple et unique. Vous allez utiliser le temps système exprimé en secondes écoulées depuis l'**Epoch**. Rappel de R1.04, l'Epoch est le temps de référence Unix, le **1^{er} janvier 1970 00:00:00**. Pour obtenir le temps système, consultez la fonction PHP **time()**.

Créez un dossier **avatars/** dans votre répertoire de test.

Modifiez votre script **enreg.php** pour déplacer l'image transférée à la soumission du formulaire, dans le dossier **avatars/**, en nommant cette image avec la valeur de **time()** + une extension en fonction du type MIME récupéré. Vous limiterez les types d'image acceptables à **PNG, GIF, JPEG** et **WEBP**.

 **time()**

Question 4

Pour terminer votre script **enreg.php**, vous allez alimenter, sur le même principe qu'au TD 1 et au TP 1, un fichier **data** qui contiendra un tableau sérialisé dont les clés seront les numéro des images (voir **Question 3**) associées aux données des individus (nom, prénom et email).

A chaque soumission d'un formulaire, vous ajouterez une ligne à votre tableau.

Si le fichier **data** n'existe pas, vous le créerez à la 1^{ère} soumission de formulaire.

Rappel : pour les tests, placer le fichier **data** parmi les scripts c'est OK, mais pas en production.

Question 5

Créez une page **liste.php**, qui affiche (`<table>`) la liste des individus.

Ajoutez un lien vers cette page dans la page **fiche.html**.

Question 6 - Astuce

Nous avons évoqué le problème qu'avec un formulaire en méthode **POST**, faire ensuite un rafraîchissement de la page provoque un message du navigateur indiquant que le formulaire va être soumis une seconde fois.

Pour éviter cela, il est possible, une fois le traitement du formulaire effectué, de renvoyer automatiquement vers une autre page, qui peut être le formulaire lui-même ou une page de confirmation de bonne réception.

Il existe des techniques impliquant du Javascript mais ce n'est pas idéal ni garanti sur 100% des navigateurs. La meilleure des solutions est celle utilisant la fonction `header()` qui écrit une réponse dans l'en-tête HTTP (rien à voir avec le `<head>` du HTML). C'est du protocole HTTP, c'est donc standard et garanti fonctionnel sur 100% des navigateurs.

La seule contrainte est que le script PHP ne doit pas avoir écrit quoi que ce soit avant (**echo**, **print**, **printf**, **print_r**, **var_dump**, etc), car les en-têtes HTTP se terminent dès le 1^{er} affichage.

La syntaxe est la suivante : **header("location: url_a_afficher");** puis généralement le script arrête son exécution après avec un **exit;**

Modifiez votre script **enreg.php** pour envoyer vers une page **succes.html** (à créer aussi), une fois le traitement du formulaire terminé ou vers **echec.html** (à créer aussi) en cas d'erreur.

Exercice 4 - Persistance des données

Nous avons vu comment persister des données de façon globale, sur le serveur, à l'aide de la sérialisation.

Cette méthode et les exercices proposés n'avaient qu'un but expérimental pour découvrir et comprendre la sérialisation. En production on n'utiliserait pas cette technique qui pose plusieurs problèmes, le plus important étant qu'il y a un risque de collision si plusieurs utilisateurs font des requêtes en même temps : lire et écrire des données dans le même fichier en même temps ne se fait pas sans garde-fou et dans notre cas rien n'est prévu en ce sens. Ce n'est pas la sérialisation qui est à remettre en question, elle a son utilité, mais pas dans le cadre du remplacement d'une BDD. En production, vous aurez certainement une BDD.

Nous allons maintenant nous pencher sur la persistance des données individuelles, c'est-à-dire par client (au sens "utilisateur du service").

Les quelques dernières diapo du CM présentent les sessions et les cookies. Retournez y jeter un œil.

Mise en place d'une session

Quand PHP met en place une session, il crée un cookie et initialise une session tout seul.

La mise en place d'une session se fait si aucun cookie n'existe ou si aucune donnée de session n'est trouvée sur le serveur.

Dans le cas contraire (le cookie existe et des données sont présentes), la session est simplement rouverte par PHP et les données de session restaurées en mémoire du script.

Tout ceci est automatique quand vous appelez la fonction **session_start()**.

Accès aux données de session

Les données de session PHP sont conservées sur le serveur et ne sont **JAMAIS** envoyées au client (navigateur). Vous pouvez donc y stocker des informations sensibles (enfin, jusqu'à un certain point car un développeur ou un administrateur sur le serveur y aura accès), il n'y a aucun risque du côté client.

La seule chose qui transite entre le serveur et le client (dans les deux sens) est le **cookie** qui identifie la session et qui permet au script (via l'appel à **session_start()**) de restaurer les données de session (et seulement celles-là) en mémoire, le temps de l'exécution du script, sur le serveur.

Comme vous en avez maintenant l'habitude avec PHP, les données de sessions sont accessibles par une... superglobale oui ! Elle s'appelle **\$_SESSION**, on s'en serait douté et c'est évidemment un tableau associatif, comme toutes les superglobales.

Lire les données se fait donc simplement par l'accès à une cellule du tableau et y écrire est aussi simple. Les valeurs stockées sont de tout type supporté par les tableaux et PHP s'occupe de la persistance (écriture dans un fichier sur disque) pour vous. Vous n'avez juste qu'à ajouter ou supprimer (**unset()**) ce que vous voulez et c'est tout. On peut difficilement faire plus aisé.

Cookie de session

Le client a la charge de conserver le cookie, c'est fait nativement sur tous les navigateurs sans code spécial de votre part, en tout cas dans le cadre des sessions.

Il vous revient de mettre en place un mécanisme de péremption du cookie et de la session, passé un certain délai, si vous le souhaitez.

ATTENTION, un cookie fait partie des en-têtes HTTP (rien à voir avec le **<head>** d'une page Web). On a déjà évoqué ces en-têtes avec la fonction **header()** en PHP. Retournez voir l'**Exercice 2 - Question 6 - Astuce** si besoin. Les en-têtes doivent impérativement être déclarés avant tout affichage dans la page Web. C'est pourquoi il est fortement conseillé de placer le **session_start()**, qui est responsable de la gestion du cookie, dès le tout début du script, et même, pourquoi pas, comme 1^{ère} instruction du script !

Question 1

Dupliquez vos scripts de l'**Exercice 3** dans un autre dossier et transformez votre code pour utiliser des sessions et non plus une sérialisation.

Désormais votre liste n'affichera donc que les individus qui ont été créés par vous, dans votre session.

Pour tester plusieurs sessions et voir le cloisonnement qui découle du mécanisme de session (si vous avez bien fait les choses), vous devez soit ouvrir vos pages Web (**liste.php**, par exemple) dans des navigateurs différents, ou encore mieux, utiliser le mode navigation privée de votre navigateur, qui permet de s'assurer que les cookies ne sont pas partagés entre les différentes fenêtres ou onglets de votre navigateur. Attention, à la fermeture d'une page de navigation privée, le cookie est perdu, et vous n'aurez plus de moyen d'accéder à vos données de session sur le serveur (mais il existe des solutions pour cela).

Questions 2

Ajoutez la possibilité de supprimer une fiche à partir de son numéro.

La suppression doit aussi supprimer l'avatar.



<https://www.php.net/manual/fr/book.filesystem.php>

Suppression d'une session

Pour supprimer une session, placez ce code dès que possible dans votre script :

```
session_unset();  
session_destroy();
```

Attention, ne faites pas de **unset(\$_SESSION)**, ces deux fonctions se chargent de nettoyer le tout (contenu de la session et cookie de session)

Exercice Bonus

Question 1

Sur la base de l'**Exercice 1**, si le champ **Nom dep.** contient plusieurs valeurs séparées par un espace, une virgule ou un point-virgule, affichez les départements qui contiennent au moins une de ces valeurs.

Question 2

Sur la base de l'**Exercice 2**, proposer un téléchargement de fichier en CSV, contenant la liste des individus, sans les avatars.

Créez les pages **csv1.php**, **csv2.php** et **csv3.php** qui répondent aux 3 solutions suivantes.

Solution 1 - URL

Votre script doit :

- Créez un dossier **csv/** s'il n'existe pas déjà
- Générer par code (sans librairie) un fichier **liste.csv** dans ce dossier
- Afficher sur la page un lien vers ce fichier, que l'utilisateur pourra télécharger en cliquant sur le lien.

En fonction de la nature des informations récupérables dans le fichier, cette solution peut présenter des risques de confidentialité des données. Ici ce sont des fiches signalétiques d'individus, et comme l'URL est toujours la même, ça pose évidemment un problème !

Fonctions ou librairies CSV

Solution 2 - URL (méthode plus sûre)

Reprenez le même mécanisme que pour la solution 1, mais en créant, cette fois-ci, un nom de fichier différent à chaque appel.

Créez et utilisez une fonction **randname(\$min, \$max)** qui génère un nom de fichier aléatoire composé de lettres min/MAJ, de chiffres et de quelques symboles aléatoires de votre choix. Le nom du fichier doit contenir entre **\$min** et **\$max** de ce jeu de caractères.

 <https://www.php.net/manual/fr/ref.random.php>

Solution 3 - Attachement

Au lieu d'afficher un lien sur la page, vous pouvez aussi déclencher le téléchargement immédiat du fichier en utilisant encore les propriétés du protocole HTTP.

Le téléchargement provoquera le stockage du fichier dans le dossier **Téléchargements** de l'ordinateur client.

Par défaut, quand un script PHP renvoie une page HTML (c'est le comportement par défaut), il inclut dans l'en-tête de la réponse HTTP (c'est du protocole, rien à voir avec le `<head>` de la page) la nature de ce qui est envoyé. Par défaut c'est un type **"text/html"**. Notez que pour une feuille de style ce sera **"text/css"** et pour une image ça peut être **"image/png"** ou encore **"image/jpeg"**. Il y en a bien d'autres et ça permet au navigateur de savoir ce qu'il doit faire avec le fichier qui lui arrive.

Pour spécifier le type de réponse et les informations nécessaires au téléchargement, on utilise encore la fonction **header()** déjà évoquée par deux fois, donc on ne revient pas en détail dessus. La syntaxe de ce qui doit être écrit dans le header HTTP dans le cadre d'un téléchargement de fichier est :

Content-Type: `<type_de_contenu>`

Content-Disposition: `attachment; filename="<nom_du_fichier_a_créer_chez_le_client">`

Content-Length: `<taille_du_fichier>`

Après avoir spécifié le contenu de l'en-tête HTTP, il ne reste qu'à inclure, dans le corps de la réponse, le contenu binaire du fichier à envoyer.

Pour ce faire, utilisez la fonction **readfile()** qui lit le contenu d'un fichier et l'envoie dans le corps de la réponse tel quel.

Pour CSV le type de contenu sera **"text/csv"**.

Adaptez vos en-têtes, testez.