# Formulaires, *cookies* et session

Cyril Rabat
cyril.rabat@univ-reims.fr

Licence 2 Informatique - Info0303 - Programmation Web 2

2023-2024



## Cours n°2

*Récupération des données d'un formulaire en PHP*
*Suivi de navigation : cookies et session*

*Version 3 septembre 2023*

# Table des matières

# Les formulaires HTML

- Les formulaires permettent d'envoyer des données au serveur Web
- Plusieurs méthodes possibles (proposées par le protocole HTTP) :
  - `GET`
  - `POST`
- Un formulaire peut contenir un ensemble d'éléments :
  - Zones de saisie ou de texte
  - Coches, boutons radio, boutons
  - Listes de sélection, *etc.*

  ↪ Les données envoyées sont des chaînes de caractères

Quelle que soit la méthode utilisée,
les données sont envoyées en clair dans le message HTTP.

# Un formulaire HTML

- Balise `form`
- Différents attributs :
  - `method` : `get` ou `post`
  - `action` : spécifie l'URL où envoyer les données
    ↪ Si `action="#"` appelle l'URL courante
  - `target` : comment est affiché la réponse
    ↪ `_blank` : dans une nouvelle fenêtre

```
<form action="traitement.php" method="post">
  <label for="inputLogin">Login</label>
  <input id="inputLogin" name="login" type="text"/>
  <label for="inputMotDePasse">Mot de passe</label>
  <input id"=inputMotDePasse" name="motDePasse" type="password"/>
  <button type="submit">Connexion</button>
</form>
```

# Élément `input`

- Champs de saisie
- L'attribut `type` définit le type de saisie :
  - `text` : texte (sur une ligne)
  - `password` : mot de passe (saisie masquée)
  - `submit` ou `reset` : bouton pour soumettre/réinitialiser le formulaire
    ↪ Possible aussi avec un bouton
  - `radio` : boutons radio
  - `checkbox` : coches
  - types ajoutés en HTML5 :
    ↪ `color`, `date`, `datetime-local`, `email`, `month`, `number`, `range`, `search`, `tel`, `time`, `url`, `week`
    ↪ Si non supportés par le navigateur, interprétés comme `type="text"` !

# Élément `select`

- Liste de sélection : élément `select`
  - Éléments HTML `option` pour définir les éléments de la liste
  - Élément sélectionné par défaut : attribut `selected="selected"`
- Possible de définir une taille pour l'affichage :
  ↪ Attribut `size`
  ↪ Par défaut 1 seul élément affiché
- Possible de définir des sélections multiples :
  ↪ Attribut `multiple`

# Autres éléments

- `textarea` : zone de saisie
  ↪ Sur plusieurs lignes
- `button` : un bouton
  ↪ Plusieurs types possibles suivant l'attribut `type` (`submit`, `button`, `reset`)
- Autres éléments HTML 5 (`datalist`, *etc.*)

Il est conseillé d'utiliser la balise `button` en lieu et place de la balise `input` avec comme valeurs pour l'attribut `type` : `submit`, `button`, etc.

# Gestion d'un formulaire en PHP

- Données récupérées automatiquement par PHP
- Création de variables superglobales :
  - `$_GET` et/ou `$_POST` et `$_REQUEST`
  - Accessibles dans tous les contextes
  - Tableaux associatifs dont les clefs sont les noms des éléments du formulaire (attribut `name`)

- Les éléments des formulaires ont des comportements spécifiques :
  ↪ Pour un `checkbox`, `$_POST['nom']` n'existe pas si non coché !
- Il est possible de récupérer des tableaux

- Attention, toujours vérifier les données récupérées depuis les formulaires
  ↪ Même pour une liste de choix (`select`) !
- Erreur fréquente : utilisation de `id` au lieu de `name`
  ↪ Les données ne sont pas présentes dans les variables superglobales

# Exemple (1/2)

## Le formulaire HTML (fichier `index.html`)

```
<form action="log.php" method="post">
  <label for="login">Login :</label>
  <input id="login" name="login" type="text"/>
  <label for="motDePasse">Mot de passe :</label>
  <input id="motDePasse" name="motDePasse" type="password"/>
  <button type="submit">Connexion</button>
</form>
```

# Exemple (2/2)

## Le traitement en PHP (fichier `log.php`)

```
<?php
if(!isset($_POST['login']) || ($_POST['login'] == "")) {
  echo "Vous␣devez␣spécifier␣un␣login␣!";
}
if(!isset($_POST['motDePasse']) || ($_POST['motDePasse'] == "")) {
  echo "Vous␣devez␣spécifier␣un␣mot␣de␣passe␣!";
}
```

- Possible d'ajouter un lien vers la première page
  ↪ `<a href="index.html">Retour</a>`
- Possible d'utiliser une redirection (si rien n'a été écrit)
  ↪ `header("Location: index.php")`

# L'*upload* de fichiers

- Nécessite un attribut spécifique dans la balise `form` :
  ↪ Attribut `enctype="multipart/form-data"`
- Élément `input` de type `file`
- Possible de spécifier la taille maximale du fichier (côté client)
  ↪ Champ caché (type `hidden`) nommé `MAX_FILE_SIZE`
- Le fichier uploadé est placé dans un répertoire temporaire (sur le serveur) :
  ↪ Dépend de la configuration du serveur Web

# Gestion des erreurs

- Si un upload est démarré (choix par l'utilisateur) :
  ↪ Entrée dans la variable superglobale `$_FILES`
- Différentes informations associées (tableau associatif) :
  - `tmp_name` : le nom du fichier temporaire sur le serveur
  - `name` : le nom du fichier uploadé
  - `error` : les erreurs éventuelles
    - `UPLOAD_ERR_OK` : OK
    - `UPLOAD_ERR_NO_FILE` : aucun fichier téléchargé
    - `UPLOAD_ERR_CANT_WRITE` : problème d'écriture
    - `UPLOAD_ERR_INI_SIZE` : la taille du fichier dépasse la taille autorisée sur le serveur
    - ...

# Exemple (1/3)

## Le formulaire HTML

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <title>Upload de fichiers</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Sélectionnez un fichier à uploader</h1>

    <form action="traitement.php" method="post" enctype="multipart/form-data">
      <label for="fichier">Sélectionnez un fichier</label>
      <input type="file" id="fichier" name="fichier" />
      <button type="submit"> Valider </button>
    </form>

  </body>
</html>
```

# Exemple (2/3)

## Le traitement en PHP

```
<?php
$resultat = false;
if(isset($_FILES) && isset($_FILES['fichier']) &&
   ($_FILES['fichier']['error'] == UPLOAD_ERR_OK))
  if(move_uploaded_file($_FILES['fichier']['tmp_name'],
                        "fichiers/".$_FILES['fichier']['name']))
    $resultat = true;
?>
<!DOCTYPE html>
<html lang="fr">
  <head> ...
    <title>Upload de fichiers</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Réception du fichier</h1>
...
```

# Exemple (3/3)

## Le traitement en PHP (suite)

```
...
if($resultat)
    echo "<p>␣Fichier␣téléchargé␣:␣<a␣href=\"fichiers/".
        $_FILES['fichier']['name']."\">␣lien␣</a>␣</p>";
else
    echo "<p>␣Le␣fichier␣n'a␣pas␣été␣téléchargé.␣</p>";
?>
    <p> <a href="index.html"> Retour </a> </p>
  </body>
</html>
```

# Quelques problèmes de sécurité

- **La taille du fichier** :
  - Champ caché `MAX_FILE_SIZE` dans le formulaire :
    ↪ Spécifie la taille maximale acceptée sur le serveur
    ↪ Permet d'avertir directement l'utilisateur (sans avoir à *uploader*)
  - Information indicative (possible de modifier cette valeur du côté client) :
    ↪ À la réception, vérification de la taille (`filesize`)
- **Gestion des extensions** :
  - Possible d'interdire certaines extensions
  - Par exemple : méthode `pathinfo`
    ↪ Retourne (notamment) l'extension du fichier
  - Pas de vérification possible : c'est informatif !

# Le suivi de navigation

- Le protocole HTTP est sans état :
  ↪ Pas de suivi entre les scripts
- Le suivi de navigation :
  ↪ Conservation des données du client tout au long des échanges
- Deux solutions :
  - Transférer les données à chaque échange :
    ↪ Formulaire avec champs cachés
    ↪ *cookie*
  - Stocker les données sur le serveur et l'associer à une clé :
    ↪ Session

# Le champ caché

- Dans un formulaire, possibilité de créer un champ caché
- Champs lisible par l'utilisateur (afficher la source) :
  ↪ Possibilité de modifier cette valeur sans contrôle (injection)
- Obligation de passer d'une page à une autre via un formulaire :
  ↪ Que faire des liens ?
  ↪ Difficile à utiliser pour des données complexes

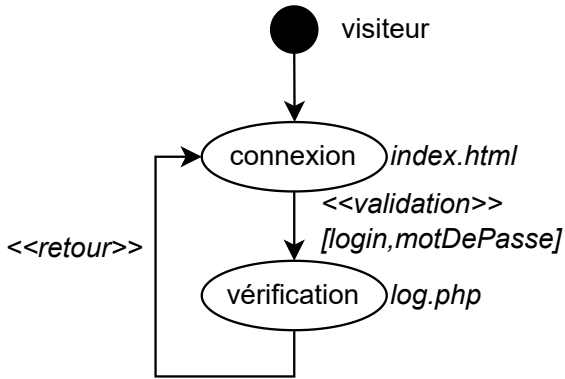
## Exemple de champ caché

```
<input type="hidden" name="login" value="toto"/>
```

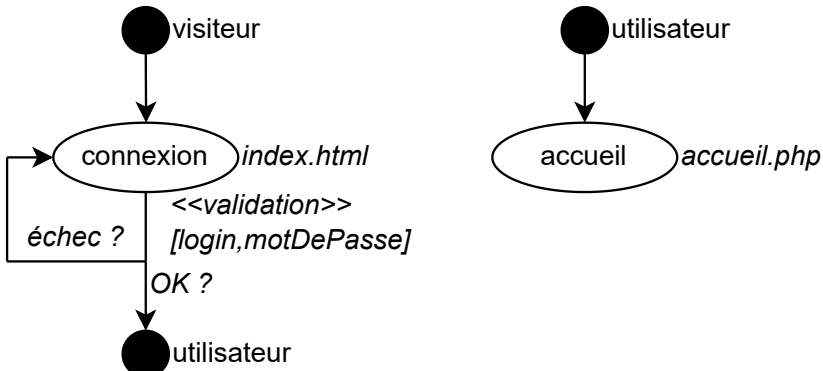
# Diagramme de navigation

- Représente les sections de l'application
  ↪ Dans un premier temps, cela correspond aux pages
- Les flèches représentent les liens entre ces sections
  - Actions de l'utilisateur entre guillemets « et »
  - États sans guillemets (par exemple : OK ?)
- Possible de spécifier les données échangées
  ↪ Données de formulaire, par exemple
- Un diagramme de navigation par type d'utilisateur
  ↪ Les sections accessibles sont spécifiques

# Exemple avec le formulaire de connexion

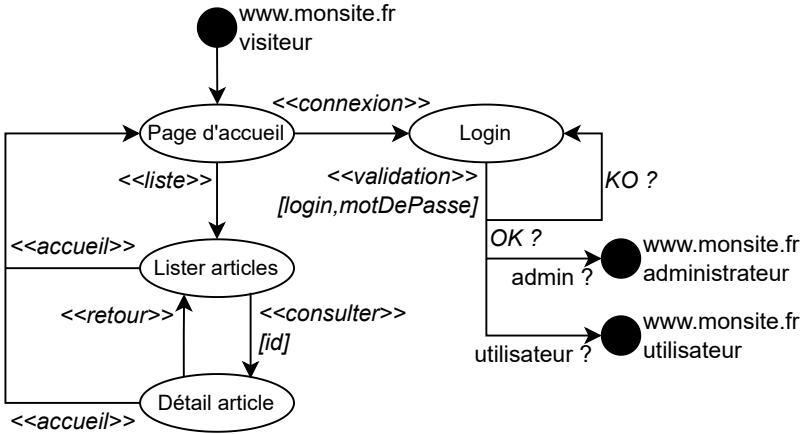


# Idem avec la redirection




Il est aussi possible d'utiliser le même diagramme.

# Autre exemple : un site de vente en ligne



# Le *cookie*

- Élément du protocole HTTP
- Enregistrement des données **du côté client** :
  ↪ Associations clé/valeur (chaînes de caractères)
- Stockage sous forme de fichiers (dépend du navigateur)
- Données envoyées dans chaque requête HTTP :
  ↪ Nom et valeur du *cookie* spécifiés dans l'en-tête
- Information limitée au site courant :
  ↪ Impossible de récupérer un *cookie* d'un autre site

# Limitation des *cookies*

- Suivant le navigateur :
  ↪ La taille et le nombre de *cookies* sont limités
- Toutes les données doivent être transférées à chaque requête
- Stockage sous forme de chaîne de caractères :
  - Sérialisation des données
  - Séparation éventuelle par un délimiteur
  - Réalisation d'un *parsing* en PHP
- L'utilisateur peut les bloquer

Des langages tels que PHP facilitent heureusement
la récupération des données

# Créer un *cookie* en PHP

- Utilisation de la fonction `setcookie`
- Paramètres principaux :
  - `name` : le nom (unique) du *cookie*
  - `value` : la valeur du *cookie* (une chaîne)
  - `expire` : la date d'expiration du *cookie* (*timestamp* UNIX)
- Autres paramètres :
  - `path` : chemin sur le serveur, sur lequel le cookie sera disponible
  - `domain` : domaine où le *cookie* est disponible
  - `secure` : *cookie* envoyé que si la connexion est sécurisée
  - `httponly` : *cookie* accessible uniquement via HTTP classique (pas via *Javascript* ; dépend du navigateur)
- Rappel : le *cookie* est un élément du protocole HTTP, envoyé dans l'entête :
  ↪ Appel à `setcookie` **avant** l'envoi de données

# Récupérer ou modifier la valeur d'un *cookie*

- Variable superglobale `$_COOKIE` (tableau associatif)
  ↪ Correspond aux *cookies* reçus (pas ceux créés dans le script)
- Pour récupérer la valeur d'un *cookie* : depuis le tableau
- Pour modifier la valeur d'un *cookie* : fonction `setcookie`
  ↪ La modification de `$_COOKIE` n'a pas d'effet sur le tableau
- Pour détruire un *cookie* :
  ↪ Spécifier une date de validité antérieure à la date actuelle

Lorsque la valeur d'un *cookie* est modifiée via `setcookie`,
l'entrée dans le tableau `$_COOKIE` n'est pas mise à jour !

# Exemple d'utilisation d'un *cookie* (1/3)

*Création du cookie*

```
<?php
setcookie("moncookie", "valeur␣du␣cookie", time() + 3600);
?>
<html>
  <head>
    <title>Dépôt d'un cookie</title>
    <meta charset="UTF-8">
  </head>
  <body>
<?php
if(isset($_COOKIE["moncookie"]))
    echo "<p>␣Le␣cookie␣'moncookie'␣était␣déjà␣présent.␣</p>";
else
    echo "<p>␣Cookie␣mis␣en␣place.␣</p>";
?>
  </body>
</html>
```

# Exemple d'utilisation d'un *cookie* (2/3)

*Accès conditionné sur l'existence d'un cookie*

```
<?php
if(!isset($_COOKIE["moncookie"]))
    header("Location:␣script1.php");
$valeur = $_COOKIE["moncookie"];
?>
<html>
  <head>
    <title>Récupération d'un cookie</title>
  </head>
  <body>
    <p> Valeur du cookie : <?php echo $valeur; ?>. </p>
    <p> Le cookie est supprimé. </p>
  </body>
</html>
```

# Exemple d'utilisation d'un *cookie* (3/3)

*Suppression d'un cookie*

```
<?php setcookie("moncookie", "", time() - 1); ?>
<html>
  <head>
    <title>Suppression d'un cookie</title>
  </head>
  <body>
<?php
if(isset($_COOKIE["moncookie"])) {
    echo <<<HTML
  <p> Valeur du cookie : {$_COOKIE["moncookie"]} </p>
  <p> Le cookie est supprimé. </p>
HTML;
}
else
    echo "<p>␣Pas␣de␣cookie␣trouvé.␣</p>";
?>
  </body>
</html>
```

# Les *cookies* de session

- Conservé tant que l'utilisateur est sur le site
- Supprimé automatiquement (quand l'utilisateur ferme son navigateur)
- Utilisation de `setcookie` sans préciser la date de validité
- Certains navigateurs peuvent avoir des comportements spécifiques :
  ↪ *Chrome* continue de fonctionner même si tous les onglets sont fermés !

Pour tester, utilisez la navigation privée.

# Exemple de *cookie* de session

```php
<?php
$compteur = 1;
if(!isset($_COOKIE["compteur"]))
    setcookie("compteur", 1);
else {
    $compteur = intval($_COOKIE["compteur"]) + 1;
    setcookie("compteur", $compteur);
}
?>
<html>
  <head>
    <title>Cookies de session</title>
  </head>
  <body>
    <p> La valeur du compteur est <?php echo $compteur; ?>. </p>
    <p> <a href="#"> Recharger </a> </p>
  </body>
</html>
```

# La session

- Objectif : conserver des données en mémoire sur le serveur
- Données associées à une clé unique :
  ↪ À la création, envoyée au client
- À chaque échange (HTTP), envoi de la clé :
  ↪ La session peut être associée au client
- Possibilité d'associer un jeu de données quelconque
  ↪ Tout type de données possible (pour les objets, attention aux inclusions)
  ↪ Sérialisation/dé-sérialisation automatiques
- Données stockées dans la variable superglobale : `$_SESSION`
  ↪ Tableau associatif contenant toutes les données
- Contrairement aux cookies, les modifications sont répercutées immédiatement

# Transmission de la clé de session

- La création/récupération de la session est automatique :
  - Si l'identifiant est présent, la session correspondante est restaurée
  - Sinon, une nouvelle session est créée
- Deux (principales) solutions :
  - Utilisation d'un *cookie* de session
  - Spécifier la clé dans l'URL ou par la méthode POST
- La configuration par défaut de PHP supporte les deux :
  - Si les *cookies* sont acceptés par le client, création d'un *cookie* de session
  - Sinon, réécriture automatique des URL (sauf externes !)

# Méthodes associées aux sessions en PHP

- `session_start()` : démarre une nouvelle session
- `session_destroy()` : détruit la session en cours
- Pour récupérer des informations sur la session en cours :
  - `session_name()` : nom de la session
  - `session_id()` : identifiant de la session
    ↪ Ou constante `SID` (définie si la session a débuté)

La clé de session pouvant être transmise dans un `cookie`,
l'appel à ces fonctions doit être réalisé avant l'envoi de données.

# Utilisation d'une session

## Exemple d'utilisation d'une session

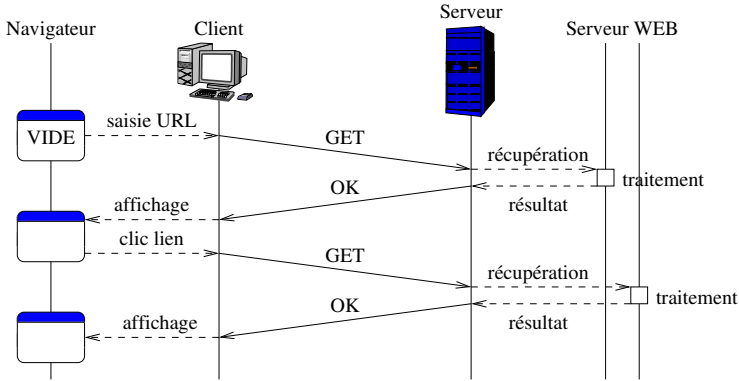
```php
<?php
session_start();

if(!isset($_SESSION['compteur']))
  $_SESSION['compteur'] = 1;
else
  $_SESSION['compteur']++;

echo "Valeur␣du␣compteur␣:␣".$_SESSION['compteur']."<br/>";
```

# Différences entre sessions et *cookies*

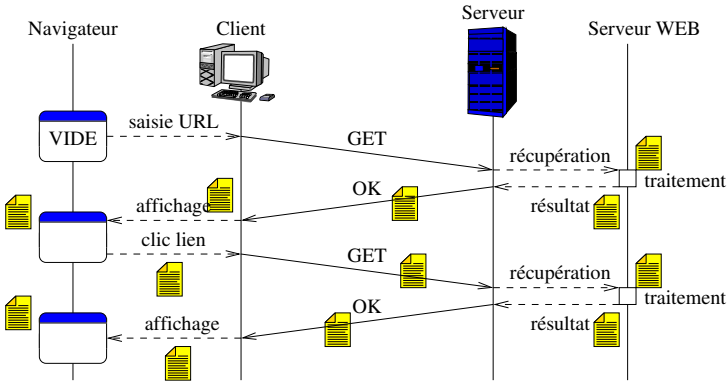
## Illustration




*Série de requêtes/réponses HTTP classiques*

# Différences entre sessions et *cookies*

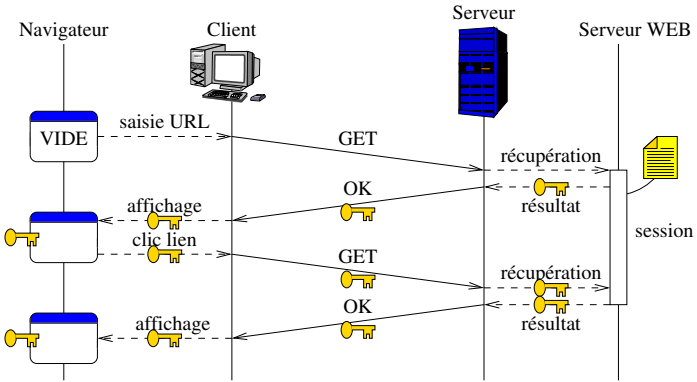
## Illustration




*Utilisation d'un* `cookie`

# Différences entre sessions et *cookies*

## Illustration




*Utilisation d'une session*

# Problématique

- Le routage permet à l'utilisateur d'accéder à toutes les pages de l'application
- Dépend de ses droits
  ↪ Connecté ou non, administrateur, etc.
- Certaines pages sont accessibles en fonction des choix précédents
  - Comment transmettre les données d'une page à une autre ?
  - Que faire si plusieurs pages ont besoin de ces données ?

# Usage d'un formulaire

- Le choix de l'utilisateur peut être envoyé à l'aide d'un formulaire
- Les données sont ensuite stockées en session
  ↪ Attention à la taille des données !
- Problème : que se passe-t-il si l'utilisateur fait un retour arrière avec le navigateur ?

```
<form method="POST" action="choix.php">
  <select name="couleur" onchange="this.form.submit()">
    <option value="-1">Sélectionnez une couleur</option>
    <option value="1">Vert</option>
    <option value="2">Rouge</option>
  </select>
</form>
```

# Passer les données via l'URL

- Le passage des données en GET revient à modifier l'URL
- `index.php?couleur=1` permet de passer la valeur 1 à la variable `couleur`
- Avantage de cette méthode : rien n'est stocké sur le serveur
- Attention à bien vérifier toutes les valeurs (et les accès associés)

```
<form method="POST" action="#">
  <select name="couleur" onchange="location=this.value">
    <option value="#">Sélectionnez une couleur</option>
    <option value="?couleur=1">Vert</option>
    <option value="?couleur=2">Rouge</option>
   </select>
</form>
```

# Un mot sur la réécriture d'URL

- *Apache* permet la réécriture d'URL
- Possible de mapper de fausses URL pour simplifier le routage
- Par exemple : `choix/couleur/1` devient `choix.php?couleur=1`
- PHP peut alors récupérer ces valeurs (dans `$_GET`)
- Les règles de réécriture sont spécifiées dans le fichier `.htaccess`
  ↪ Il faut que la configuration d'*Apache* le permette

# Routage complet de l'application

- Pour chaque page de l'application, il faut déterminer :
  - Les données nécessaires
  - Les redirections si besoin (en cas d'erreur)
  - La vérification des accès
- Chaque page peut alors être accessible de n'importe où
  ↪ Et l'utilisateur peut faire des retours arrière...
- Les `frameworks` proposent en général tout un système de routage
  - Création des URL
  - Récupération des données
  - Vérification des accès