

PHP

CAS CONCRETS

RAPPEL : ARCHITECTURE


ACTEURS



Client web

- peluche : un navigateur web, Chrome par exemple
- boîte : la machine sur laquelle tourne le navigateur, un ordinateur portable par exemple

ACTEURS

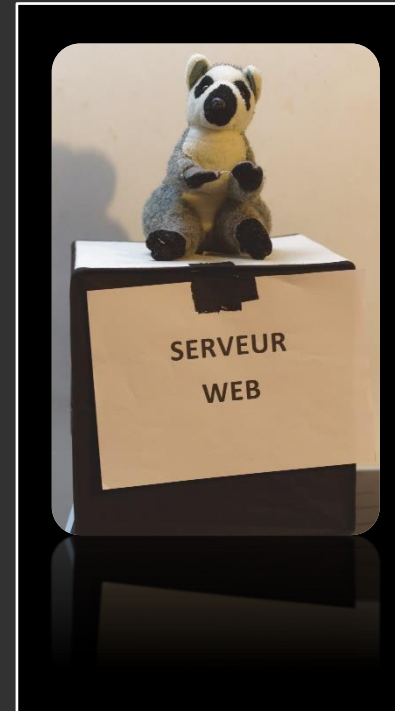


A small stuffed animal, resembling a koala or a similar marsupial, is sitting on top of a white box. The box has the words "SERVEUR" and "WEB" printed on it in black capital letters. The box is placed on a dark, reflective surface, creating a reflection. The background is a plain, light-colored wall.

Serveur web

- peluche : le **service-serveur web**, logiciel qui assure le fonctionnement, Apache par exemple.
- boîte : la **machine** qui héberge le site web et le service-serveur web

ACTEURS



Attention : en TP souvent une seule machine mais toujours un logiciel client et un logiciel serveur.
1 seule boîte, 2 peluches.

TRAITEMENT D'UN FORMULAIRE



FONCTIONNEMENT GÉNÉRAL

- Deux solutions pour le traitement des données saisies :
 - en un seul fichier : le fichier qui assure l'affichage est aussi chargé du traitement des données
 - ou bien avec deux fichiers : un fichier dédié assure le traitement des données→ détaillé dans les pages suivantes
- Des contrôles de validité des données sont effectués côté client, d'autres côté serveur :
 - côté client : c'est de l'ergonomie, une aide pour le visiteur
 - côté serveur : ce sont les contrôles qui assurent la sécurité de l'application

FONCTIONNEMENT GÉNÉRAL

FICHIERS D’AFFICHAGE ET DE TRAITEMENT DISSOCIÉS

- **Version 1** : traiter les données envoyées par le formulaire dans un script dédié, « qui ne fait que cela », donc **2 fichiers** au total

//extrait du code du fichier afficherForm.php

```
<form action="traiterForm.php" method="get">
  <label for="inputNom">Login : </label>
  <input type="text" name="login" ><br>

  <label for="inputPrenom">Password : </label>
  <input type="password" name="pass" >
  <br>
  <button type="submit" >Valider</button>
</form>
```

codes à retrouver dans le dossier « **cnx1fichier** »

```
<?php
// extrait du fichier traiterForm.php

// ...
if (strtoupper($_GET["login"])=="JPC" && $_GET["pass"]=="sassepaye")
{
  $_SESSION["login"] = "JPC";
  $_SESSION["droits"] = "adherent";
  header("location:pageInfoMembre.php");
  exit();
}
else
{
  // redirection vers l'accueil, en passant un message en GET
  header("location:afficherForm.php?message=erreur d'identification");
  exit();
}
```


FONCTIONNEMENT GÉNÉRAL

FICHER UNIQUE POUR AFFICHAGE ET TRAITEMENT

- **Version 2** : traiter les données envoyées dans le script qui affiche le formulaire, **1 seul fichier**.

Principe : `action` référence le fichier unique, on vérifie au début du script si des données ont été envoyées ou non, on agit en fonction

```
<?php
    // .....

    // vérification : est-ce qu'il y a des données envoyées ?
    if (isset($_GET["login"]) && isset($_GET["pass"]))
    {
        // si oui : on traite ces données (par ex : si OK redirection)
    }
?>
<!doctype html>
<html>
    <!-- Affichage du formulaire -->
</html>
```

codes à retrouver dans le dossier « **cnx2fichiers** »



FONCTIONNEMENT GÉNÉRAL

CONTRÔLES DES DONNÉES SAISIÉS

- **Côté client** : contrôles pour aider à la saisie (ergonomie).

extrait du code du fichier afficherForm.php

```
<input type="password" name="pass" ... .. >
```

- **Côté serveur** : contrôles de sécurité.

```
if ($_GET["login"]=="JPC" && $_GET["pass"]=="sassepaye")
{
    // traitements si accès OK
}
else
{
    // traitements si accès pas OK
}
```



FONCTIONNEMENT GÉNÉRAL

CONTRÔLES DES DONNÉES SAISIES

- **Contrôles de sécurité** : côté serveur et non côté client. Raisons :
 - aucune garantie sur l'exécution des contrôles côté client (accès libre au code HTML et JavaScript)
 - aucune garantie sur l'origine des requêtes (données) qui arrivent au serveur
- Fonctionnement **à généraliser** : la logique de l'application doit se trouver côté serveur.



FONCTIONNEMENT GÉNÉRAL

GET OU POST ?

- Requêtes HTTP, différentes méthodes : GET, POST, PUT, DELETE, ...
- Pour les formulaires HTML : GET et POST, sans respecter la sémantique HTTP

	GET	POST
Mode d'envoi	Dans l'URL : visible directement pour l'internaute	Dans le corps de la requête : moins facilement visible
Possibilité de sauvegarder (bookmark) la page « résultat » (en réalité : bookmark de la requête http)	Oui. (Permet également de copier-coller)	Non.
Taille maximale	Limitée (2000, éventuellement plus selon le client et le serveur)	Non limitée
Mise en cache	Mise en cache dans le navigateur, et noté dans les logs du serveur	Pas de données en cache, pas de données dans les logs

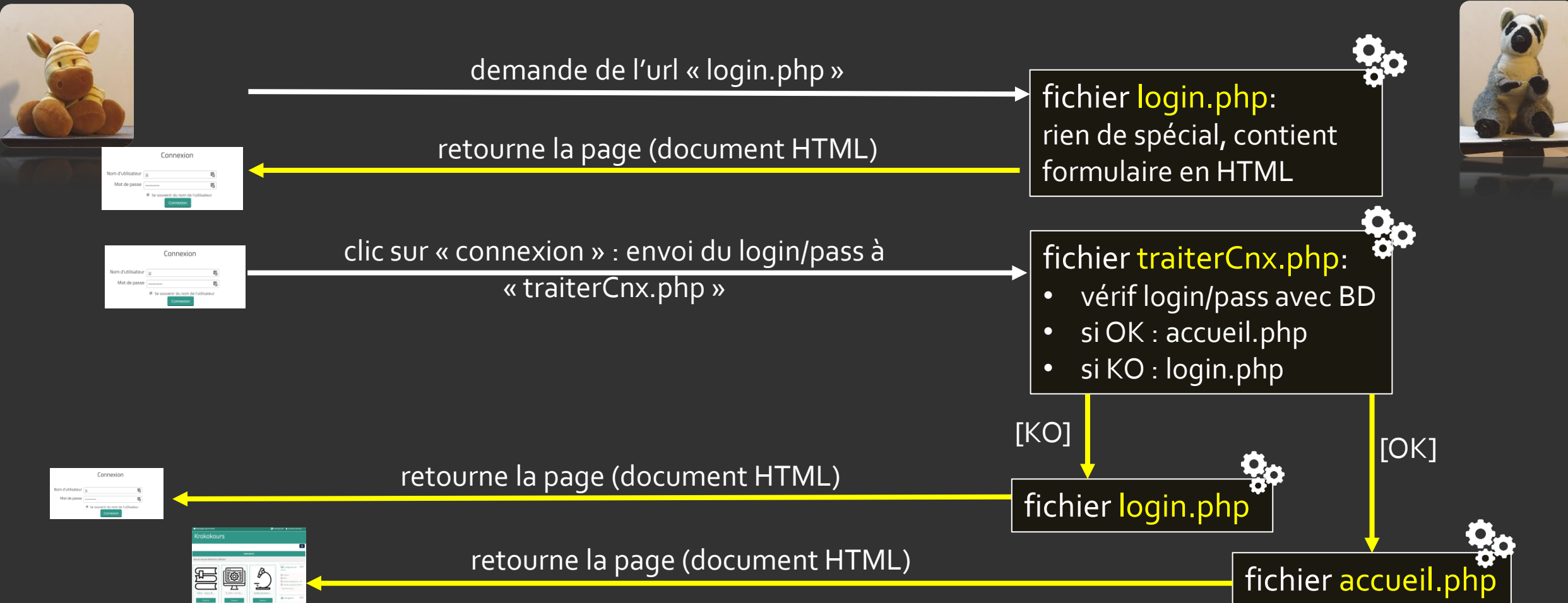


FONCTIONNEMENT GÉNÉRAL GET OU POST ?

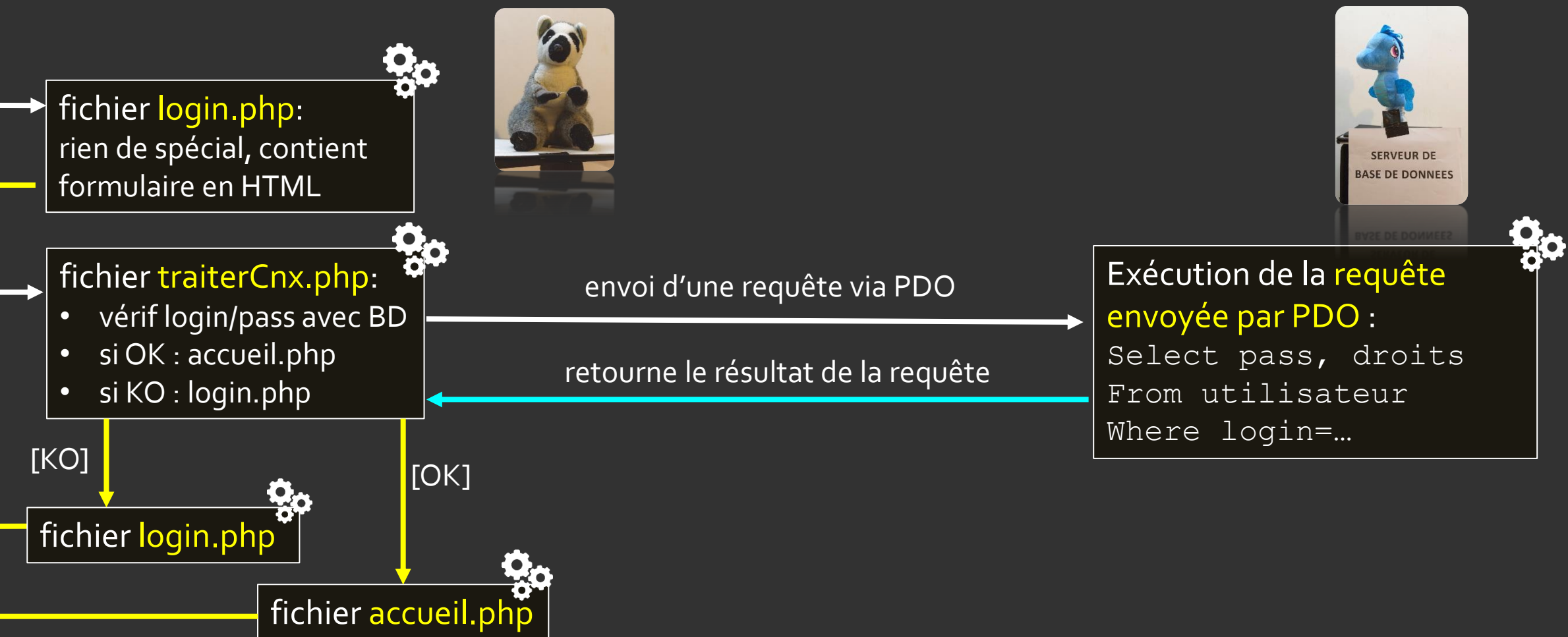
- En bref : **POST** n'est pas une garantie de sécurité... mais c'est « **plus sûr** » que **GET**.
- En développement : **GET** peut être plus **pratique**, on teste en changeant l'URL plutôt qu'en saisissant à nouveau. Plus rapide et possibilité de **copier-coller des URL de test**.
- En production :
 - Pas de données sensibles dans les GET (comme un mot de passe) :
 - « tout le monde » peut le lire par-dessus votre épaule
 - mise en cache dans le navigateur et présent dans l'historique : pb si ordinateur partagé
 - lisible dans les logs du serveur et des machines traversées (routeurs, proxys...)
 - **Ne pas systématiser l'usage de POST** : la possibilité de bookmarker ou copier-coller un lien (pour partage par exemple) peut être intéressante.

TRAITEMENT D'UN FORMULAIRE FORMULAIRE DE CONNEXION ET CONTRÔLE D'ACCÈS

FORMULAIRE DE CONNEXION, EXEMPLE



FORMULAIRE DE CONNEXION, EXEMPLE



PAGE DE RÉCEPTION DES DONNÉES (CAS D'UN MOT DE PASSE CHIFFRÉ)

- Fichier traiterForm.php, grandes lignes (voir le répertoire *cnx2fichiersPDO*) :
 1. On **vérifie** que les champs attendus, login et pass, sont **présents dans la requête HTTP**
 2. On **recherche** dans la **base de données** le mot de passe chiffré et les droits associé à ce login
 3. S'il n'y a pas de correspondance : retour à la page de login
 4. S'il y a une **correspondance** :
 1. On vérifie que le mot de passe saisi correspond avec celui qui est chiffré (password_verify), si ce n'est pas OK retour à la page de login
 2. On place le login et le niveau d'autorisation dans la **session**
 3. On **redirige** vers une page d'accueil

CONTRÔLE D'ACCÈS

- **Objectif** : garantir que certaines pages ne soient accessibles « qu'aux personnes autorisées »
- **Principe** :
 - Lorsque la connexion d'un utilisateur est acceptée, ses droits sont stockés en **session**
 - A chaque tentative d'accès à une page, on contrôle :
 - Que les **droits** sont suffisants, d'après la session
 - Eventuellement que la **page** demandée « **appartient bien** » à la personne. Par exemple, pour voir les informations de la commande numéro 156897 il ne suffit pas d'être un client connecté, il faut être le client qui a passé cette commande.