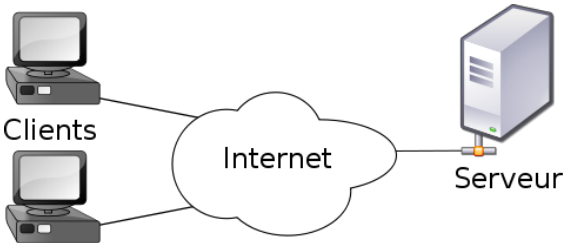
HTTP, serveurs web, et pages dynamiques

Alexandre Niveau

GREYC — Université de Caen

Application client-serveur

- Application client-serveur :
  - application en réseau
  - le programme « principal », appelé *serveur*, tourne en continu sur une machine (généralement distante)
  - pour accéder à l'application depuis sa machine, il faut lancer un programme *client*, qui va se connecter au serveur et échanger des messages avec lui suivant un protocole bien défini



[img/client-serveur.png]
Exemple d'architecture client–serveur : deux clients font leurs requêtes à un serveur via Internet.
Source : Wikimedia [https://commons.wikimedia.org/wiki/File:Modèle-client-serveur.svg] (LGPL 2.1).

Le web, une application client-serveur

- Dans le cadre du web:
  - Le protocole d'échange de messages s'appelle HTTP
  - Toute machine connectée à internet peut faire tourner un serveur web
    - elle écoute sur son port 80
    - elle interprète les requêtes HTTP reçues
    - elle renvoie les réponses HTTP (qui contiennent typiquement du HTML) au demandeur
  - Il y a de nombreux programmes clients, les plus courants étant les navigateurs web visuels (comme Firefox ou Chrome) :
    - ils envoient des requêtes HTTP aux serveurs
    - ils interprètent la réponse HTTP (typiquement, ils mettent en forme le HTML)

HTTP

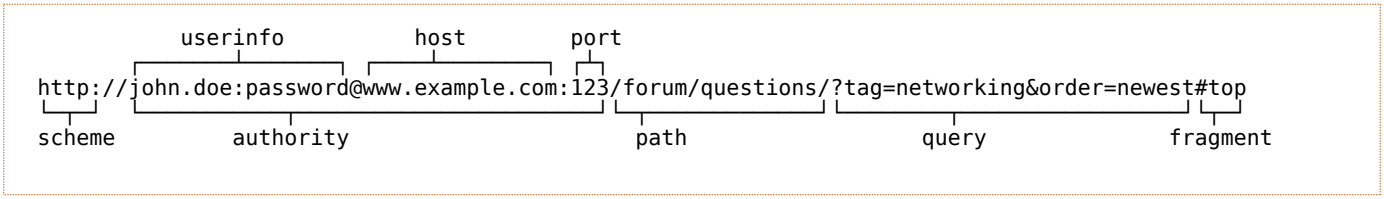
- *HyperText Transfer Protocol* ; c'est le langage que parlent le serveur web et le

navigateur web pour se communiquer les pages

- Élément le plus fondamental du web, et aussi le plus caché pour le grand public
- Pas complètement cependant :
  - son nom apparaît au début des URL (mais les navigateurs actuels ne l'affichent plus)
  - les célèbres *cookies* sont un élément du protocole
  - certains codes de statut sont bien connus… (erreur 404)
- HTTPS est la version sécurisée de HTTP : les messages sont chiffrés, et donc illisibles pour quiconque les intercepterait entre le client et le serveur

URL

- Le web est un ensemble de ressources (documents — ou autre !), liées entre elles
- Chaque ressource a une *adresse* (ce qui permet de la récupérer, et de la lier à d'autres ressources)
- Les adresses sont appelées URL (ou plus généralement URI)
- La syntaxe des URL a été créée pour le web, mais est utilisée dans de nombreux autres contextes



- La partie « scheme » indique le protocole à utiliser pour récupérer la ressource
- La partie « authority » indique à qui il faut s'adresser pour récupérer la ressource
- La partie « path » indique le chemin vers la ressource sur le serveur
- La partie « query » permet d'indiquer des paramètres (pour filtrer ou mettre en forme la ressource)
- La partie « fragment » identifie une sous-partie de la ressource (par exemple une section d'une page HTML)
- Autres exemples de la syntaxe URI [https://en.wikipedia.org/wiki/Uniform_Resource_Identifier#Examples]
- Remarque : il y a de nombreuses variations pour les noms des différentes parties d'une URL [https://tantek.com/2011/238/b1/many-ways-slice-url-name-pieces]

Requête HTTP

- Structure d'une requête HTTP :
  - Une ligne de requête avec une commande (*request method*), un chemin et la version du protocole
  - Plusieurs lignes de champs d'en-tête

    - Une ligne vide
    - Le corps du message (optionnel)
- Commandes : GET, POST, HEAD, PUT, DELETE…
- Champs d'en-tête : Host, User-Agent, Accept… seul Host est obligatoire
- Exemple, le client veut accéder à la ressource `http://www.toto.fr/blog/posts/243.html?order=newest#top`

```
GET /blog/posts/243.html?order=newest HTTP/1.1
Host: www.toto.fr
Accept: text/html
Accept-Charset: utf-8
Connection: keep-alive
```

- NB: le fragment n'est pas transmis, il est géré uniquement par le client

Réponse HTTP

- Structure d'une réponse HTTP :
    - Une ligne de statut avec le *status code* et un petit message explicatif
    - Plusieurs lignes de champs de réponse
    - Une ligne vide
    - Le corps du message (optionnel), typiquement le code HTML de la page
- Codes : 200 (OK), 404 (not found), 500 (internal server error)…
- Champs de réponse : Content-type, Last-Modified, Location…

```
HTTP/1.1 200 OK
Date: Mon, 05 Jan 2015 12:12:12 GMT
Last-Modified: Wed, 02 Jan 2013 18:18:18 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 323
Connection: close

<html>
  <head> <title>Le blog de Toto</title> </head>
  <body> <p>Bienvenue sur mon blog !</p> </body>
</html>
```

Serveurs web

- Il existe de nombreux serveurs web (au sens logiciel), les plus utilisés étant Apache et nginx
- Rôle du programme serveur : reçoit des requêtes HTTP et renvoie des réponses HTTP (qui peuvent contenir n'importe quoi)
- En pratique, on a souvent besoin que les URL correspondent à des *fichiers*

  ➤ un des rôles principaux du programme serveur est d'interpréter le *chemin* demandé dans la requête comme un chemin dans le système de fichiers
- Interprétation pas directe ! On ne veut pas que tout le système de fichiers de la

machine soit accessible sur le web…

➤ le chemin demandé est « traduit »

- Typiquement, le serveur considère que la racine de l'URL (chemin `/`) correspond à un répertoire fixé — typiquement `/var/www/html` — appelé « **web root** » ou « document root »

  ➤ la « traduction » est plus ou moins une simple concaténation des chemins : l'URL
  `http://www.toto.fr/bidule/truc/machin.html`
  sera interprétée comme correspondant au fichier
  `/var/www/html/bidule/truc/machin.html`
  sur le serveur

- On dit que le serveur web « *sert* » le contenu du répertoire `/var/www/html` à la racine du site

Servir un fichier / un répertoire

- Si le chemin traduit de l'URL correspond à un fichier (HTML, ou image, PDF, audio…), c'est lui que le serveur envoie en réponse.
- Si le chemin traduit ne correspond à rien, le serveur enverra une erreur 404 (on peut configurer ce comportement)
- Si le chemin correspond à un répertoire le serveur regarde s'il contient un fichier `index.html`

  ➤ si oui, c'est ce fichier qu'il sert

  ➤ sinon, ça dépend de la configuration : il peut renvoyer une 404, une 403, ou générer une page pour montrer le contenu du répertoire

Pages dynamiques

- Si besoin que le contenu d'un site soit *dynamique*, par exemple dépende d'une base de données, il faut que le serveur génère du HTML
- Il va déléguer cette tâche à d'autres programmes, typiquement à l'interpréteur PHP
- Utilisation de Common Gateway Interface, ou bien exécution en fonction de l'extension du fichier

PHP sur le web

- PHP a été conçu au départ pour mettre des bouts de programme *au milieu d'une page HTML*

```
<!DOCTYPE html>
<html>
<head><title>Ma page</title><meta charset="UTF-8" /></head>
<body>
    <h2>Les nombres</h2>
    <p>Voici les nombres de 1 à 10000 :

        <?php
            for ($i = 1; $i <= 10000; $i++) {
                echo "$i, ";
```

```
            }
        ?>

    </p>
</body>
</html>
```

Résultat [demo/nombres.php]

- C'est une page HTML avec des blocs d'instructions encadrées par `<?php` et `?>`
- Ce qui est en dehors de ces blocs est affiché tel quel
- Les blocs sont remplacés par le résultat de l'exécution des instructions
- Regarder le code source de la page : noter bien qu'il ne subsiste aucune trace du code PHP !

Que s'est-il passé ?

- Déroulement :
  - Le navigateur demande au serveur la page `nombres.php`
  - Le serveur voit qu'il s'agit d'un script PHP : il appelle l'*interpréteur PHP* sur le script `nombres.php`
  - L'interpréteur exécute le script, ce qui génère une page HTML
    - Ce qui est en dehors des blocs PHP est reproduit tel quel
    - Les blocs sont remplacés par le résultat de l'exécution des instructions
  - La page HTML générée est renvoyée au navigateur
- Le navigateur **ne voit jamais le code PHP** : il ne récupère que de l'HTML !
- Les navigateurs ne savent d'ailleurs pas interpréter un fichier PHP. Si vous essayez d'en ouvrir un en local avec Firefox, il devrait vous proposer de le télécharger, comme pour tout type de fichier qu'il ne connaît pas.

Intégration avec HTML

```
<?php
// variables et calculs
$titre = "Page démo";
$contenu = "Le contenu sera généré par mon programme";
?>
<!DOCTYPE html>
<html lang="fr">
<head>
<title><?php echo $titre; ?></title>
</head>
<body>
<h1><?php echo $titre; ?></h1>
<p><?php echo $contenu; ?></p>
</body>
</html>
```

Résultat [demo/integ.php]

- Regarder le code source de la page : noter que le premier bloc d'instructions a simplement disparu (il ne génère aucune sortie), mais il a bien été exécuté puisque les variables ont la bonne valeur.

Affichage des erreurs

- PHP peut afficher les erreurs dans un fichier de log, et sur la sortie standard (c'est-à-dire dans le HTML généré)
- L'affichage dans le HTML est pratique pour développer… mais interdit sur un serveur en prod !
- Configuration dans le fichier de config (`php.ini`) :
  - Log géré par le flag `log_errors` (on/off) et la variable de config `error_log` (chemin absolu vers un fichier de log)
  - Affichage HTML géré par le flag `display_errors` (on/off)

Parenthèse sur la configuration de PHP

- Si pas accès au fichier de config, on peut utiliser des fichiers de config Apache locaux, les `.htaccess` (un guide [http://perishablepress.com/stupid-htaccess-tricks/] ; solution pour activer le log [http://perishablepress.com/how-to-enable-php-error-logging-via-htaccess/])
- On peut aussi modifier la valeur des paramètres de config dans un script avec la fonction `ini_set`, par exemple `ini_set('display_errors', 'on')`. Cependant en cas de *parse error*, le script n'est pas exécuté du tout, et donc le `ini_set` n'a aucun effet…
- **Attention, sur votre www-prod, le flag `display_errors` est `off` et n'est modifiable que par un admin.**

En-têtes HTTP : fonction `header`

- Surtout utilisé pour les redirections : `header("Location: http://google.fr");` (PHP se charge alors de passer un code 302)
- `header("Content-type: text/plain;");` pour renvoyer du texte brut par ex. (exemples texte [demo/header.php], html par défaut [demo/header_defaut.php])
- On peut modifier/ajouter n'importe quel en-tête, par exemple `header('Content-Disposition: attachment; filename="exemple.html"');` va proposer la page à l'enregistrement (démo [demo/header_attachment.php])
- Attention : `header` modifie l'en-tête HTTP, qui est avant le HTML
  ⇒ il ne faut pas commencer à écrire du texte avant l'instruction `header` : même un saut de ligne suffit à provoquer une erreur « Headers already sent »

  démo et explications [demo/header_error.php]

- Certaines installations bufferisent la sortie pour éviter ça, mais toutes ne le font

pas : c'est donc une fausse sécurité (un peu comme désactiver les erreurs d'un compilateur…)

[#output-buffering]
Output buffering

- L'inclusion d'un script PHP provoque son exécution : tout ce qu'il écrit est directement affiché par le script appelant.
- Il est parfois utile de ne pas afficher tout de suite le résultat de l'exécution d'une série d'instructions PHP, notamment pour ce cas-là
- Il faut utiliser l'*output buffering (OB)* :

```php
<pre>
<?php
    echo "One\n";

    ob_start();  /
* on active l'output buffering :
                 * tout ce qui suit sera enregist
                 * dans un buffer au lieu d'être
                 */

    echo "Two\n";
    include "demo/fichier.txt";

    $var = ob_get_clean();  /
* on désactive l'OB, on
                           * récupère le contenu
                           * buffer, et on le vi
                           */

    echo "Three\n";
    echo "Contenu du buffer :
\n « " . $var . " »\n";
?>
</pre>
```

```
One
Three
Contenu du buffer :
 « Two
Coucou,
je suis un fichier !

Au revoir

 »
```

- Il y a d'autres fonctions pour manipuler l'OB, mais dans 95% des cas ces deux-là suffisent.

Variables « superglobales »

- Elles sont accessibles partout
- Ce sont des tableaux associatifs
- `$_POST` : contient les données passées dans le corps d'une requête `POST`
- `$_GET` : contient les paramètres de l'URL (même si la requête était `POST` !)

  Démo GET [demo/get.php?nom=toto&x=12]

- `$_SERVER` : données du serveur, on y trouve notamment
  - l'URL locale de la page courante (`$_SERVER['REQUEST_URI']`)

  - et la « partie virtuelle » de l'URL courante (`$_SERVER['PATH_INFO']`).
  - Attention, il y a beaucoup de données qui se ressemblent mais dont le fonctionnement est subtilement différent… et la doc n'est pas toujours très explicite
- `$_ENV` : données sur l'environnement
- `$_FILES $_COOKIE $_SESSION` : feront l'objet de cours spécifiques

Exemple [https://dev-ensweb.users.info.unicaen.fr/l3/php/superglobals.php/chemin/virtuel?variable=toto&test=titi&num=15]

- Rappel : `phpinfo()` affiche la configuration PHP de votre serveur et la valeur de toutes les superglobales (Voir [https://dev-ensweb.users.info.unicaen.fr/l3/php/info.php])

Existence de variables

- Si on utilise une variable qui n'existe pas, un avertissement (notice) est lancé

  ➤ il n'y a pas d'erreur, PHP considère que la variable a la valeur `null`
- Ne pas s'appuyer là-dessus : s'il y a un avertissement, c'est pour une bonne raison (code pas propre, plus difficile à maintenir)
- Il y a une pseudo-fonction pour tester si une variable existe ou non : `isset`

```
if (isset($toto))
        echo "La variable \$toto existe, voilà son contenu : $toto";
else
        echo "La variable \$toto n'existe pas";
```

  Le code précédent ne génère aucun avertissement.
- A priori, dans du code propre et bien organisé, **on n'a *jamais* besoin de `isset()`** !
- Pourtant, en pratique, `isset` est très souvent utilisée pour vérifier que les tableaux `$_GET` ou `$_POST` contiennent un paramètre donné
  ➤ utiliser plutôt `key_exists`, qui risque moins de cacher une erreur dans le code :

```
if (key_exists('toto', $_GET))
        $toto = $_GET['toto'];
else
        $toto = '';
```

`empty()`

- La pseudo-fonction `empty` permet de vérifier qu'une variable est « vide », c'est-à-dire que
  - soit elle n'existe pas
  - soit elle existe, mais est (faiblement) égale à `false`

- **Attention**, dans la liste des valeurs égales à `false` [http://php.net/manual/en/function.empty.php#refsect1-function.empty-returnvalues], il y a notamment la chaîne `'0'` !

  ➤ même si vous savez que votre variable contient une chaîne, `empty()` ne vous permet pas de vérifier qu'elle est *vraiment* vide !

  ➤ si vraiment besoin de manipuler des variables qui n'existent pas forcément, il est conseillé d'utiliser plutôt `isset` et un test explicite (hors cas très particuliers).

- Il est également très fortement déconseillé d'utiliser `empty($maVariable)` si la variable est censée exister : cela peut cacher des erreurs dans le code (erreur dans le nom de variable, par exemple).

  ➤ si besoin de tester qu'une variable est « vide », utiliser plutôt `!$maVariable` ou `$maVariable == false` (comparaison volontairement faible), ou mieux, comparer explicitement à la valeur vide qui vous intéresse : `$maChaine === ''` ou `$monTableau === []`.

- Pour résumer, il est encore plus déconseillé d'utiliser `empty` que `isset`.

- Détails sur `isset` et `empty`, leur relation avec `null`, et quand les utiliser : the definitive guide to PHP's `isset` and `empty` [http://kunststube.net/isset/]

Spécifications et normes

- RFC 7230 [https://tools.ietf.org/html/rfc7230] et suivantes : Hypertext Transfer Protocol (HTTP/1.1)
- RFC 3986 [https://tools.ietf.org/html/rfc3986] Uniform Resource Identifier (URI): Generic Syntax

Sites

- Site du W3C [http://w3c.org/] World Wide Web Consortium
- HTTP Cats [https://http.cat/]

Tutoriels

- How DNS works [https://howdns.works/] (en bande dessinée)

Lectures complémentaires

- Vidéo TED, Tim Berners-Lee sur le futur web [http://www.ted.com/talks/lang/fr/tim_berners_lee_on_the_next_web.html]

 [http://creativecommons.org/licenses/by/4.0/]

Ce cours est mis à disposition selon les termes de la licence Creative Commons Attribution 4.0 International [http://creativecommons.org/licenses/by/4.0/].