

# **Sécurité des Technologies Internet**

## **Chapitre 1. Technologies des applications Web**

Abraham Rubinstein

[abraham.rubinstein@heig-vd.ch](mailto:abraham.rubinstein@heig-vd.ch)

Septembre 2018 - Février 2019

# Sécurité des applications Web

- Avant d'attaquer une application, il est nécessaire d'en comprendre (ou d'en imaginer) son fonctionnement !
- Il est donc nécessaire de connaître les technologies
- Exemples :
  - Protocole HTTP : requête, réponse, GET, POST, REST, URL, ...
  - Cookies, sessions, headers, ...
  - Encodage URL/HTML, Unicode, Base64, Hex, ...

# Le protocole HTTP

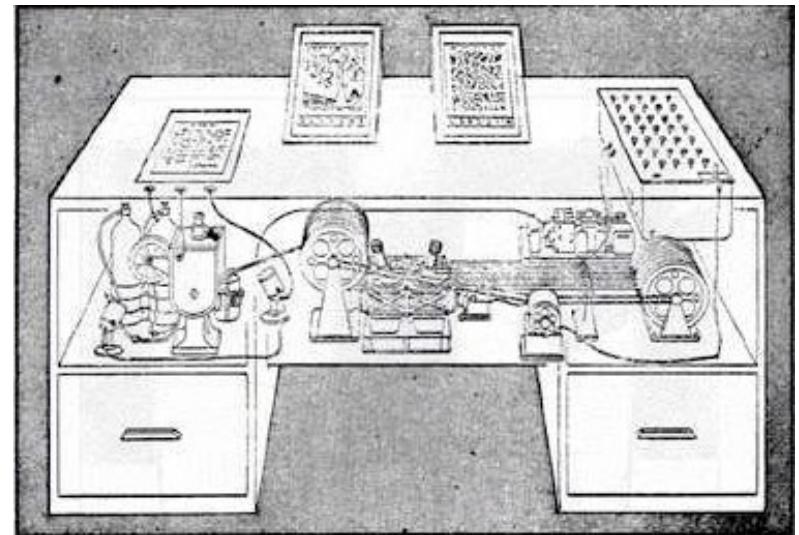
- HTTP est le cœur du World Wide Web !
- HTTP - HyperText Transfer Protocol
  - Protocole de transfert hypertexte

# L'histoire du Web : Memex (1945)

- Memex pour Memory Extender
- Auteur : Vannevar Bush  
Article : As We May Think  
Revue : The Atlantic Monthly. 1945



- *Bush décrit un appareil (bureau) relié à une bibliothèque capable d'afficher des livres et de projeter des films. Cet outil est aussi capable de créer automatiquement des références entre les différents médias.*



# L'histoire du Web : HyperText (1967)

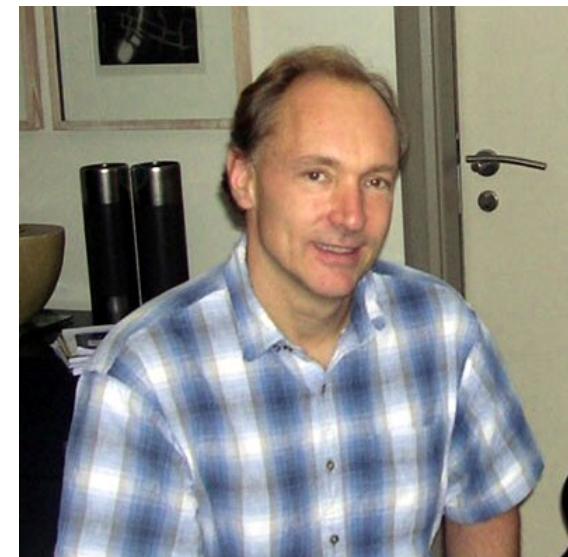
- Ted Nelson décrit le projet Xanadu :

*Xanadu would consist of a world-wide network that would allow information to be stored not as separate files but as connected literature. Documents would remain accessible indefinitely. Users could create virtual copies of any document. Instead of having copyrighted materials, the owners of the documents would be automatically paid via electronic means a micropayment for the virtual copying of their documents.*

Source : <http://www.ibiblio.org/pioneers/nelson.html>

# L'histoire du Web : HTTP (1990)

- HTTP a finalement été inventé en 1990
  - l'inventeur est Tim Berners-Lee
  - développement effectué au CERN
  - HTTP a été développé spécialement pour le World Wide Web
  - Il a été développé en même temps que les adresses Web (URI) et le langage HTML



source : [http://fr.wikipedia.org/wiki/Fichier:Tim\\_Berners-Lee.jpg](http://fr.wikipedia.org/wiki/Fichier:Tim_Berners-Lee.jpg)

# Historique HTTP

- 1990 : première version
- 1991 : première version documentée
  - HTTP/0.9
  - uniquement GET, retourne de l'HTML uniquement
- 1996, HTTP est standardisé par l'IETF
  - HTTP/1.0 RFC 1945
  - Par rapport à la version précédente, HTTP supporte
    - les serveurs virtuels,
    - la gestion du cache,
    - l'identification

IETF : Internet Engineering Task Force

# Historique HTTP

- 1997/1999 : version la plus utilisée actuellement
  - HTTP/1.1 RFC 2616 de 1999 (RFC 2068 de 1997)
  - Nouveautés : 5 nouvelles méthodes, connexions persistantes, authentification par digest
- 2014 version actuelle :
  - HTTP/2 RFC 7540 de 2015
  - Nouveautés : binaire au lieu de text, multiplexé (une seule connexion permet l'envoi en parallèle), header compression, “push” proactif
- IETF : Internet Engineering Task Force

# Qu'est-ce que HTTP ?

- Protocole de communication
- Architecture client-serveur
- Protocole de la couche application
- Peut fonctionner sur n'importe quelle couche fiable
  - en pratique TCP/IP

# Les clients/serveurs HTTP

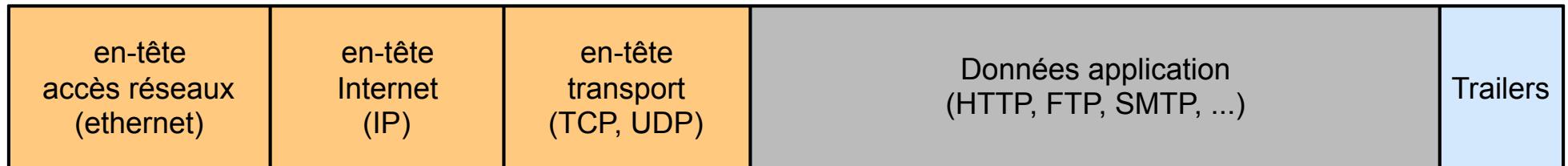
- Clients HTTP :
  - Les clients les plus connus sont
    - les **navigateurs Web** (Firefox, Chrome, Safari, Opera, IE...)
  - il ne faut pas oublier
    - les **robots** (indexation de sites Web)
    - les **aspirateurs** de sites
- Serveurs HTTP :
  - les clients se connectent aux serveurs HTTP
  - les plus connus sont :
    - **Apache** (Linux, Unix, Windows, MacOS)
    - **Nginx** (Linux, Unix, Windows, MacOS)
    - Microsoft Internet Information Services (**IIS**)
- Entre eux, il peut y avoir des proxys, passerelles, ou tunnels.

# Modèle en couches

Modèle OSI	Modèle TCP/IP	Exemples de protocoles
Application		
Présentation	Application	<b>HTTP, FTP, SMTP, POP, ...</b>
Session		
Transport	Transport	<b>UDP, TCP</b>
Réseau	Internet	<b>IP, ICMP</b>
Liaison		
Physique	Accès réseau	<b>Ethernet</b> ou autre

# En-têtes typiques d'un paquet IP

- Paquets standards



# X/HTML

- HTML (Hyper Text Markup Language)
  - langage de publication (pas de programmation)
- XHTML (EXtensible Hyper Text Markup Language)
  - langage de représentation de documents, dans le web
- Buts initiaux (HTML) :
  - **Publier des documents** avec texte, en-tête, tables, etc...
  - **Récupérer** de l'information en ligne via de simples **cliques** sur des **liens hypertextes**.
  - Fournir des **formulaires** pour mener des transactions avec des services à distance.
  - Inclure des **applications** diverses / multimédia telles que feuilles de calcul, clips vidéo / son, etc.

# X/HTML

- HyperText Markup Language (HTML)
  - Représentation des documents en ASCII
  - Les navigateurs interprètent l'HTML lorsqu'ils affichent la page
  - Fonctions de base :
    - Formatage de texte, références des images, hyperliens (HREF)
- Une page Web peut avoir plusieurs composants
  - Le fichier HTML de base
  - Les objets référencés, p.ex : images, CSS, RSS, ...
- Langage de représentation simple
  - Syntaxe facile à comprendre/apprendre

# X/HTML

- Références :
  - HTML 5.2 : <https://www.w3.org/TR/html52>
  - HTML 4.01: <http://www.w3.org/TR/html401>
  - XHTML 2.0: <http://www.w3.org/TR/xhtml2>
- Exemple de document HTML :

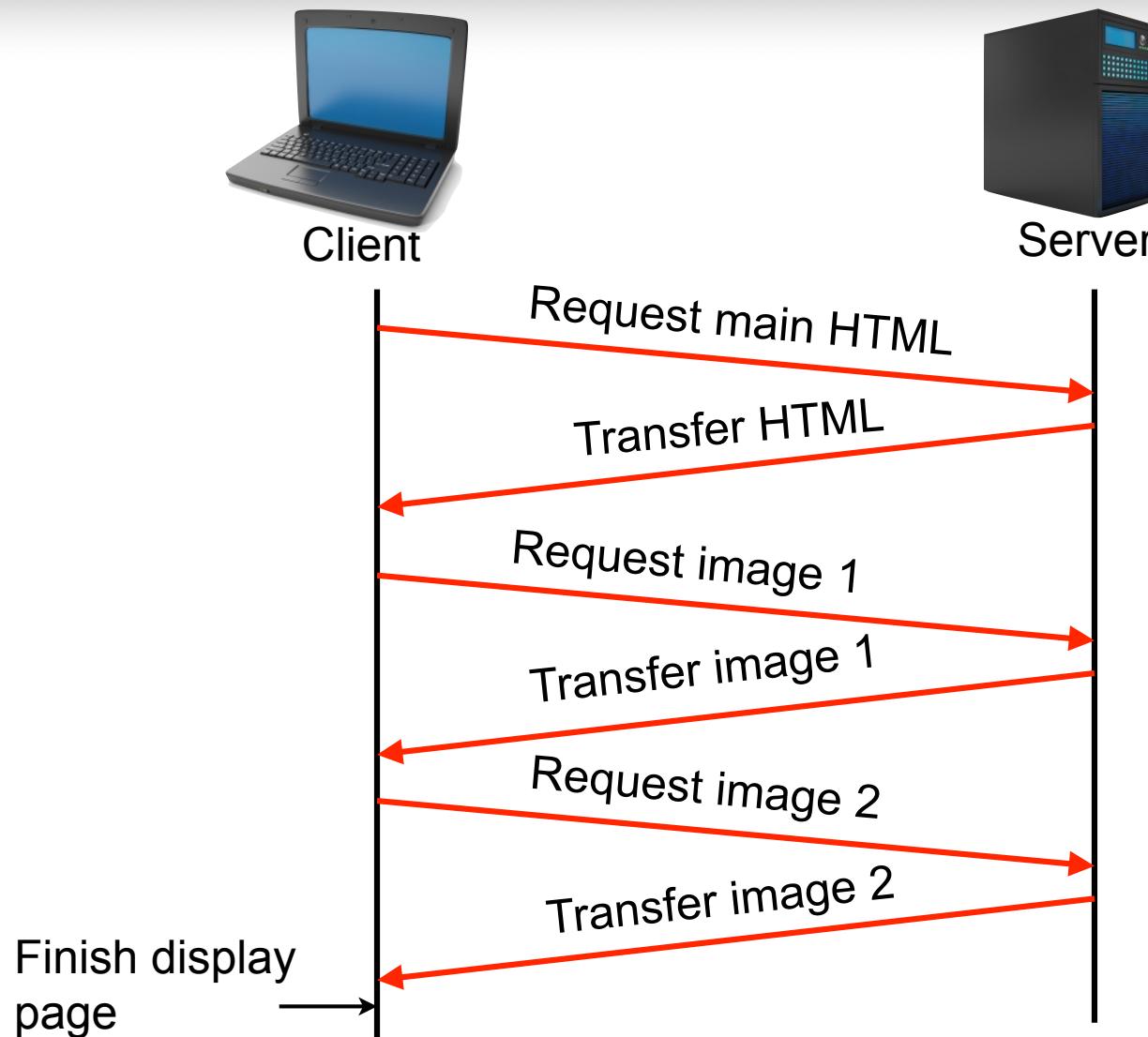
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
    "http://www.w3.org/TR/html4/strict.dtd">  
<HTML>  
    <HEAD>  
        <TITLE>My first HTML document</TITLE>  
    </HEAD>  
    <BODY>  
        Hello <A HREF="http://www.heig-vd.ch">world</A>!  
    </BODY>  
</HTML>
```

} DOCTYPE

# Un peu de vocabulaire

- Une **page web** est constituée d'objets.
- Un **objet** peut être un fichier XHTML, CSS, une image JPEG, une applet Java, un fichier audio, etc.
- Une **page web** est généralement constituée d'un fichier XHTML de base et d'un certain nombre d'objets référencés (via URIs).
- Exemple : une page web contenant de l'XHTML référençant un fichier CSS et cinq images JPEG est composée de combien d'objets?
- HTTP est dit **sans état** : chaque requête est **indépendante** de la précédente. Le serveur ne conserve pas d'historique (états)...
  - Plus performant car les protocoles avec état sont complexes...

# Chargement d'une page Web complète

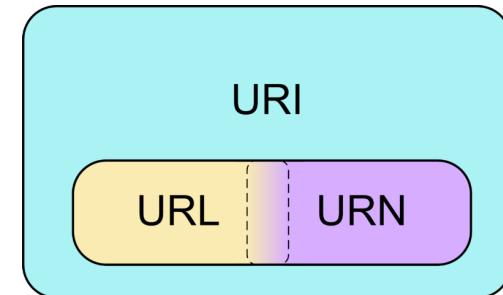


# Connexions persistantes

- Connexions HTTP non persistantes :
  - Un seul objet à la fois est transféré dans une connexion TCP
  - HTTP 1.0 utilise des connexions non persistantes
- Connexions HTTP persistantes :
  - Plusieurs objets peuvent être transférés dans une seule connexion TCP
  - HTTP 1.1 utilise des connexions persistantes par défaut

# URI

- Uniform Ressource Identifier
  - identifie une ressource (physique ou abstraite) sur le réseau
  - respecte une syntaxe (RFC 3986)
- Les URI sont la technologie de base du WWW
  - tous les hyperliens sont exprimés sous forme d'URI
- URL - Uniform Ressource Locator
  - fournit les moyens d'agir ou d'obtenir la ressource
  - décrit le mode d'accès et l'emplacement
  - exemple : <http://www.ietf.org/rfc/rfc3986.txt>
- URN - Uniform Ressource Name
  - identifie une ressource par son nom
  - respecte l'espace de noms
  - pas directement lié à son emplacement ou à la méthode d'accès
  - exemple : urn:ietf:rfc:3986



Source : [http://fr.wikipedia.org/wiki/Fichier:URI\\_Venn\\_Diagram.png](http://fr.wikipedia.org/wiki/Fichier:URI_Venn_Diagram.png)

# Syntaxe d'une URL

**protocol : //hostname[ : port]/path/resource?query#fragment**

protocol	http, ftp, https, smtp, rtsp, etc.
hostname	soit le nom de domaine complet soit l'adresse IP
port	permet de spécifier un port (optionnel) si non spécifié, port par défaut du protocol
directory path	le chemin d'accès
resource	la ressource cible
query	Permet de passer des paramètres à un script PHP, Perl etc....
fragement	Permet d'indiquer une position (ancre, fragment) dans une page

# Encodage d'URL

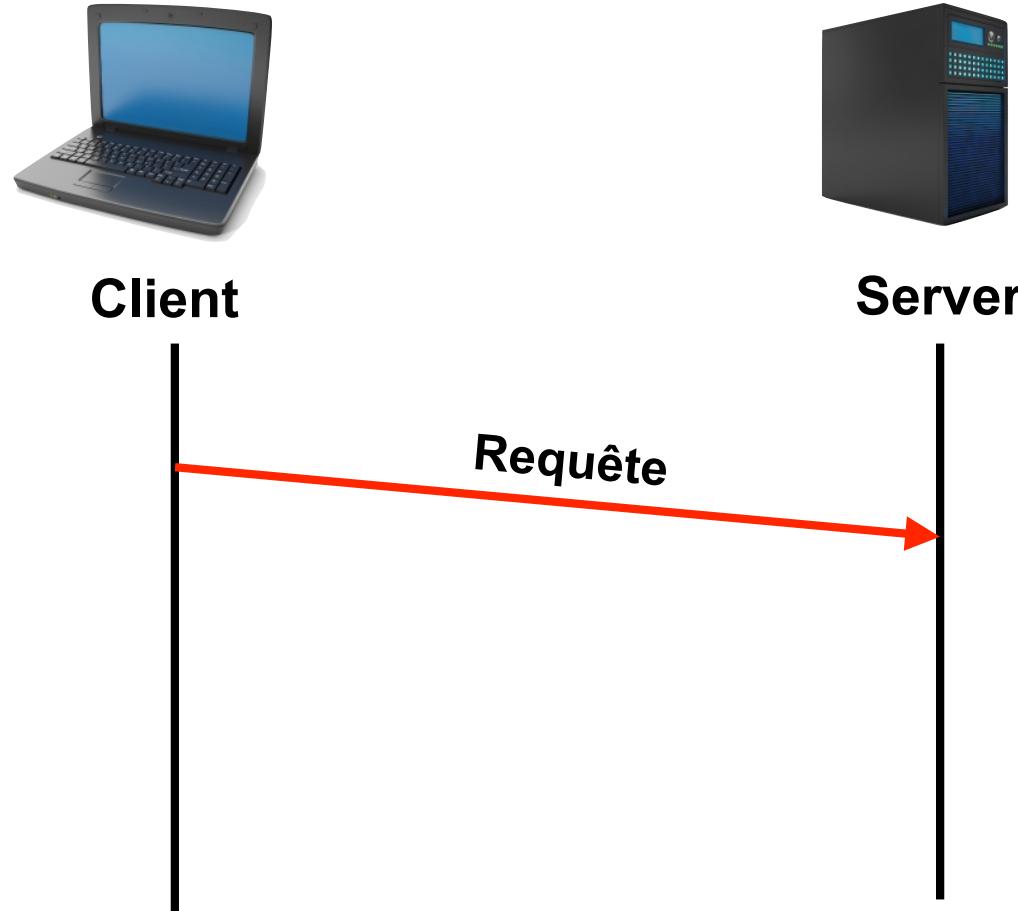
- Certains caractères ne peuvent pas être représentés
  - typiquement ; / ? &
- Ils doivent être "escaped"
- Il faut remplacer le caractère par son code ASCII
  - par exemple :

"&"	->	%26
" "	->	%20

# Protocole HTTP

- Protocole client-server
- Utilisé pour transférer des ressources
- Propriétés importantes :
  - Protocole de type requête - réponse
  - Ressource metadata
  - Sans état (stateless)
  - Format ASCII

# Requête HTTP



# Requête HTTP

<METHODE> <RESSOURCE> <PROT\_VERSION>

Ligne de requête

<EN-TETES>

permet de fournir des informations au serveur

Lignes d'en-têtes

<ligne vide>

<Eventuellement un contenu>

Contenu

# Méthodes HTTP

Méthode	HTTP/1.0	HTTP/1.1	
GET	X	X	permet d'obtenir une ressource paramètres dans l'URL de la requête uniquement
POST	X	X	permet d'effectuer une action paramètres dans URL et dans contenu de la requête
HEAD	X	X	permet de recevoir uniquement les lignes d'en-têtes de la réponse (pas le corps)
OPTIONS		X	permet au client de connaître les options du serveur
PUT		X	permet au client de transmettre un document
DELETE		X	permet d'effacer la ressource
TRACE		X	permet d'indiquer au serveur qu'il doit renvoyer la requête telle qu'il l'a reçue
CONNECT		X	permet de se connecter à un proxy ayant la capacité de faire du tunneling

# Requête HTTP

- Exemple

GET / HTTP/1.1

Ligne de requête

Host: www.heig-vd.ch

User-Agent: Mozilla/5.0 (X11; Linux i686; rv: 14.0) Gecko/20100101 Firefox/14.0.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip, deflate

DNT: 1

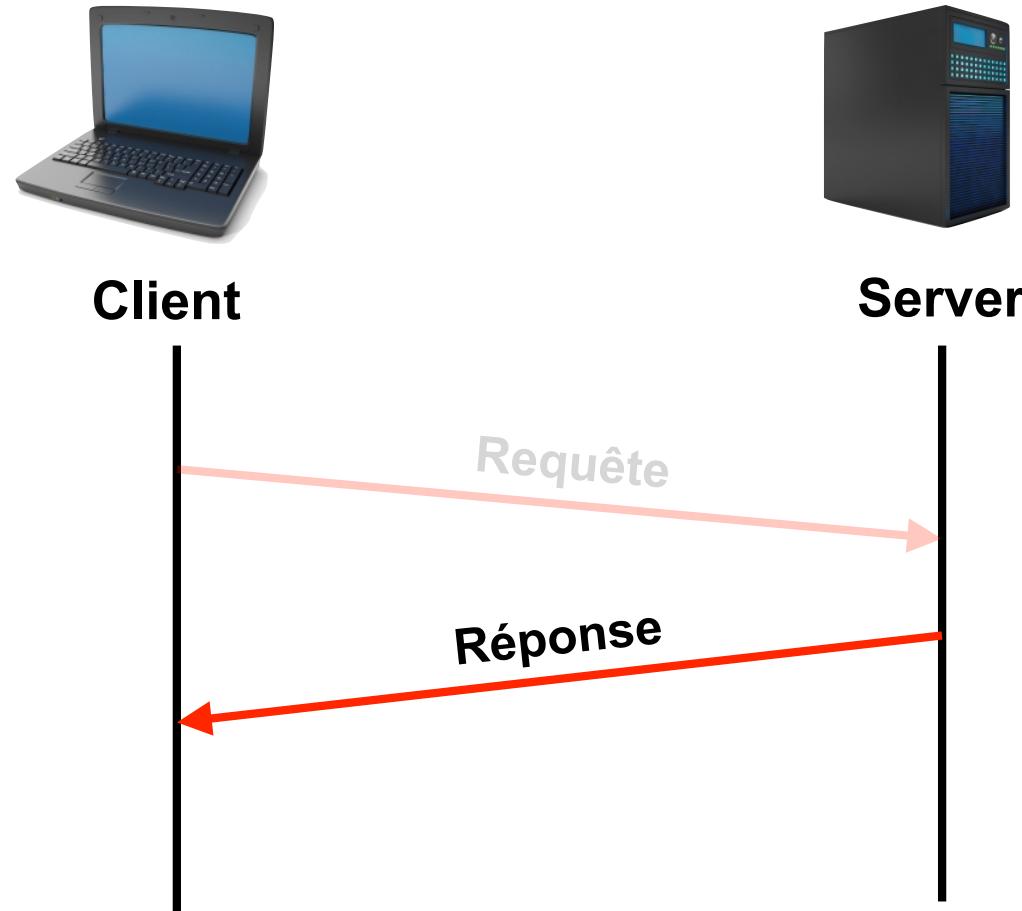
Connection: keep-alive

<ligne vide>

Lignes d'en-têtes

Contenu

# Réponse HTTP



# Réponse HTTP

<PROT\_VERSION> <STATUS-CODE> <STATUS-PHRASE>

Ligne de statuts

<EN-TETES>

permet de fournir des informations au client

Lignes d'en-têtes

<DATA>

Contenu

# Code de statuts HTTP

Code	Classe	Exemple
1xx	Informational	100 Continue
2xx	Success	200 OK
3xx	Redirection	301 Moved Permanently 304 Not Modified
4xx	Client error	400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found
5xx	Server error	500 Internal Server Error 503 Service Unavailable

# Réponse HTTP

- Exemple

HTTP/1.1 200 OK

Via: 1.1 ADTMG01  
Connection: Keep-Alive  
Proxy-Connection: Keep-Alive  
Content-Length: 32304  
Date: Wed, 05 Sep 2012 14:57:26 GMT  
Content-Type: text/html; charset=utf-8  
Server: Microsoft-IIS/7.0  
Cache-Control: private  
X-AspNet-Version: 2.0.50727  
Set-Cookie: .ASPXANONYMOUS=AajrmQjCzQEkAAAANGU2NzM5MDktNGEzNy00NDZjLWE3NGYtYzg3OTFhM2UzZTIw0; expires=Wed, 14-Nov-2012 01:37:26 GMT; path=/; HttpOnly  
Set-Cookie: language=en-US; path=/; HttpOnly  
X-Powered-By: ASP.NET

Ligne de statuts

Lignes d'en-têtes

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
<head id="Head"><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"><meta http-equiv="Content-Language" content="fr-FR"><meta
```

Contenu

# Génération des réponses

- Retourne simplement un **fichier (statique)**
  - L'URL correspond à un fichier (HTML, image, ...)
  - Le serveur génère les en-têtes appropriés
  - Le serveur retourne le fichier
- Génération d'une **réponse dynamique**
  - L'URL pointe sur un programme du serveur
  - Le serveur exécute le programme et renvoie la sortie au client
- Retourne des **meta-données** sans corps de message

# HTTP reste stateless !!!

- Stateless
  - chaque réponse dépend de la requête
  - chaque requête est traitée de manière indépendante !
- Comment rendre HTTP statefull ?

# URL personnalisées

- Exemple :
  - <http://www.STI.com/member.php?id=2>

- Récupérer la variable (en PHP) :

```
<?php  
    if (isset($_GET['id'])) {  
        $id = $_GET['id'];  
        ...  
    }  
?>
```

- Garder la variable active dans les liens (en PHP) :

```
echo '<a href="members_manage.php?id=' . $id . '">Manage members</a><br />';
```

# Cookies

- Spécifications originales :
  - [http://wp.netscape.com/newsref/std/cookie\\_spec.html](http://wp.netscape.com/newsref/std/cookie_spec.html)
- Spécifications "officialisées" : RFC 2965
- Fonctionnement :
  - utilisation de 2 nouveaux en-têtes HTTP :
  - **Set-Cookie** (dans les réponses) : fixer/inclure un cookie.
  - **Cookie** (dans les requêtes) : voilà mon cookie.

# Cookies

- Assigner une variable dans un cookie (en PHP) :
- Récupérer une variable du cookie (en PHP) :

```
<?php
// set the cookies
setcookie("cookie[three]", "cookiethree");
setcookie("cookie[two]", "cookiethree");
setcookie("cookie[one]", "cookieone");

// after the page reloads, print them out
if (isset($_COOKIE['cookie'])) {
    foreach ($_COOKIE['cookie'] as $name => $value) {
        $name = htmlspecialchars($name);
        $value = htmlspecialchars($value);
        echo "$name : $value <br />\n";
    }
}
?>
```

Source <http://www.php.net/manual/en/function.setcookie.php>

# Syntaxe de Set-Cookie

- **Set-Cookie:** NAME1=VALUE1; expires=DATE; path=PATH;  
domain=DOMAIN\_NAME; secure, NAME2=VALUE2; ... (1 ligne)
- NAME=VALUE
  - seul attribut obligatoire...
- max-age=value
  - durée de vie du cookie (en secondes).
- expires=date
  - date de fin de vie du cookie (HTTP/1.0).
- path=value
  - sous-ensemble des chemins pour lesquels le cookie sera retourné (cf. transparent suivant).
- domain=value
  - nom de domaine pour lequel le cookie est valide. P. ex. : cookie valide pour requête sur a.b.com ou encore c.b.com si domain=.b.com. (cf. ch. 1 et 3.2.2 de la RFC2965). Si pas précisé, par défaut, le navigateur prend le nom de domaine du serveur qui a émis le Set-Cookie2.