



Lyon 1



département
Informatique

Université Claude Bernard Lyon 1

Licence Math-Informatique 1^{ère} année

Partie 4

Olivier Glück

Université LYON 1 / Département Informatique

Olivier.Gluck@univ-lyon1.fr

<http://perso.univ-lyon1.fr/olivier.gluck>

Copyright

- Copyright © 2024 Olivier Glück; all rights reserved
- Ce support de cours est soumis aux droits d'auteur et n'est donc pas dans le domaine public. Sa reproduction est cependant autorisée à condition de respecter les conditions suivantes :
 - Si ce document est reproduit pour les besoins personnels du reproducteur, toute forme de reproduction (totale ou partielle) est autorisée à la condition de citer l'auteur.
 - Si ce document est reproduit dans le but d'être distribué à des tierces personnes, il devra être reproduit dans son intégralité sans aucune modification. Cette notice de copyright devra donc être présente. De plus, il ne devra pas être vendu.
 - Cependant, dans le seul cas d'un enseignement gratuit, une participation aux frais de reproduction pourra être demandée, mais elle ne pourra être supérieure au prix du papier et de l'encre composant le document.
 - Toute reproduction sortant du cadre précisé ci-dessus est interdite sans accord préalable écrit de l'auteur.

Plan du cours

- CM1 : Internet, les réseaux et le web
- CM2 : Pages HTML et feuilles de styles CSS
- CM3 : Web interactif, formulaires, pages dynamiques et PHP
- CM4 : Protocole HTTP, méthodes GET et POST
- CM5 : Les applications d'Internet
- CM6 : La couche transport : les protocoles TCP et UDP
- CM7 : Le protocole IP
- CM8 : Les protocoles Ethernet, ARP et ICMP. Synthèse des échanges entre un client et serveur Web



Lyon 1



département
Informatique

Université Claude Bernard Lyon 1

CM4 : Protocole HTTP, méthodes GET et POST

Quelques rappels

Le protocole HTTP

La programmation CGI

Plan du CM4

- Quelques rappels

Qu'est-ce que le web ? Format simple des URL, Le navigateur et le serveur web, Pages statiques et dynamiques, Interactions entre un navigateur et un serveur web

- Le protocole HTTP

Une transaction typique, Format des requêtes/réponses, Durée de vie des connexions, Cookies, Différentes versions de HTTP, Proxy et caches web, Les requêtes clientes : GET, POST et HEAD, Les réponses du serveur, Les en-têtes, les types MIME

- La programmation CGI

Qu'est-ce que la programmation CGI ? Un premier exemple, Récupérer les données du formulaire avec GET/POST, Format URL encodé, Format de la sortie standard, Les variables d'environnement



Lyon 1



département
Informatique

Université Claude Bernard Lyon 1

Quelques rappels

Qu'est-ce que le web ?

Format simple des URL

Le navigateur et le serveur web

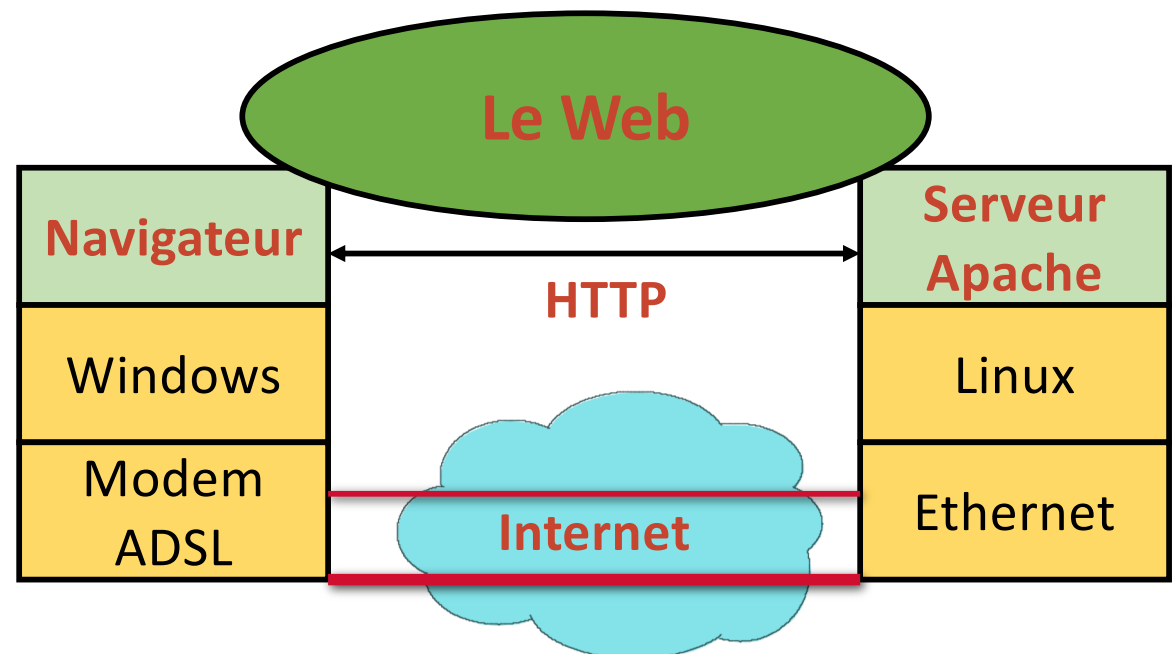
Pages statiques et dynamiques

Interactions entre un navigateur et un serveur web

Qu'est-ce que le web ? (1)

- Une application d'Internet qui permet le partage de documents liés entre eux et appelés "pages web"
- Une page web peut contenir du texte, des images, des programmes, des liens vers d'autres pages web...
- Fonctionne en mode Client/Serveur au dessus de l'architecture TCP/IP

L'application est répartie sur le client et le serveur qui dialoguent selon un protocole applicatif spécifique



Qu'est-ce que le web ? (2)

- Des clients : les navigateurs qui font l'interface avec l'humain (Firefox, Internet Exploreur, Chrome, Safari...)
- Des serveurs : ils hébergent les pages web et répondent aux demandes des clients (Apache, Microsoft IIS...)
- Le web est né en 1994 avec la création du W3C (**WWW Consortium**) par le CERN et le MIT (Tim Berners-Lee président) qui s'occupe de la normalisation et des développements du web
- Sa popularité est due à :
 - Des interfaces graphiques conviviales
 - Une très grande quantité d'informations très diverses

Qu'est-ce que le web ? (3)

- Le web repose sur 3 concepts :
 - **L'URL** : permet au client de désigner la page demandée
 - Uniform Ressource Locator : Comment ? Où ? Quoi ?

Comment ? **Où ?** **Quoi ?**

`http://etu.univ-lyon1.fr/licence/lifasr2.html`
 - **HTTP** : permet de définir le format et la signification des messages échangés entre le client et le serveur (protocole)
 - **Requête HTTP** : du client vers le serveur, pour demander une page web
 - **Réponse HTTP** : du serveur vers le client, pour répondre au navigateur
 - **HTML, CSS, PHP, Javascript...** : les langages du web
 - **HTML** : permet de décrire le contenu d'une page web, interprété par le navigateur web pour afficher la page et demander les objets incorporés
 - **CSS** : permet de définir les styles de la page (format, couleurs, positions...)
 - **PHP** : permet d'exécuter un programme par le serveur
 - **Javascript** : permet d'exécuter un programme par le navigateur

Format d'une URL

`proto://host_name:port/path?arguments`

- la racine "/" de **path** est définie par la configuration du serveur Web

(**Attention** : à ne pas confondre avec la racine du système de fichiers sur le serveur)

- **/path** peut contenir une étiquette (point d'ancrage)

`http://www.monsite.fr/projet/doc.html#label`

- **arguments** permettent de passer des informations à des programmes s'exécutant sur le serveur

Par exemple, **?action-joueur=gauche** dans le jeu 2048

Le navigateur web (1)

- Analyse l'URL demandée et récupère le nom du serveur
- Demande au DNS l'adresse IP de la machine serveur
- Etablit une connexion TCP vers le numéro de port de l'URL (80 par défaut)
- Fabrique la requête HTTP et l'envoie au serveur
- Réceptionne la réponse HTTP
- Interprète le code HTML reçu : commandes de formatage et de mise en forme (police, gras, couleurs...)
- Demande les objets incorporés au serveur et affiche la page correctement formatée
- Exécute les programmes Javascript s'il y en a

Le navigateur web (2)

- Pour faire l'affichage de la page, il se base sur
 - les valeurs par défaut du navigateur,
 - les préférences de l'utilisateur fixées dans le navigateur,
 - les valeurs fixées dans le document ou les feuilles de styles.
 - Exemples : couleur des liens (visités ou non), du texte, fond de la page, polices...

Le serveur web

- Il est en permanence à l'écoute des requêtes formulées par les clients (qui peuvent être très nombreux !)
- Il vérifie la validité de la requête...
 - Le document demandé peut ne pas exister
 - L'accès a un document peut être restreint (authentification possible)
- ... et y répond si la requête est valide : envoi du texte, des images, de la feuille de styles, du code à exécuter sur le client (Javascript).
- Il peut renvoyer un message d'erreur, une demande d'authentification...
- Il peut exécuter un programme localement (PHP) qui va générer une réponse HTML (page **dynamique**) en fonction des arguments transmis par le navigateur.

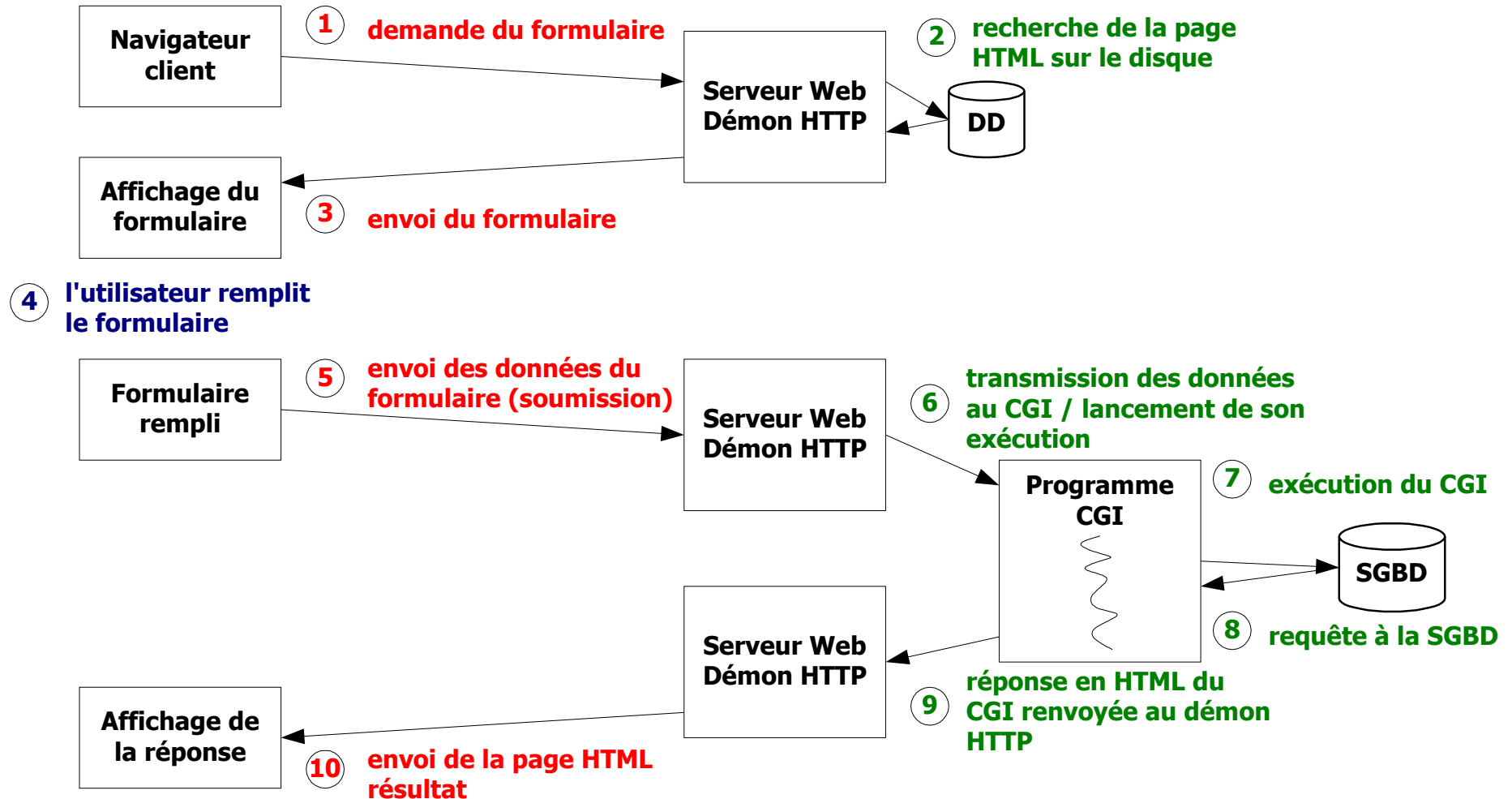
Pages statiques et dynamiques

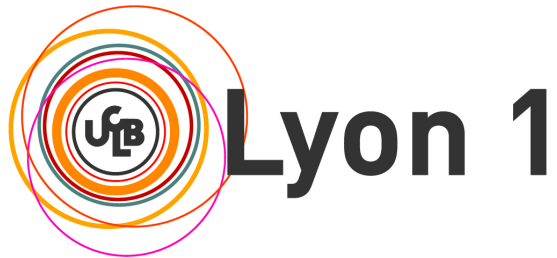
- Le HTML ne permet pas d'interactivité avec l'utilisateur
 - Les pages visualisées sont "statiques"
- Pages statiques
 - La page visualisée N fois sur le même navigateur donnera toujours le même résultat
- Pages dynamiques
 - La page visualisée dépend des manipulations de l'utilisateur
 - Elle est générée dynamiquement selon les actions de l'utilisateur dans la page
 - Nécessite de la programmation pour prendre en compte les actions
 - Programmation Web côté client : principalement Javascript
 - Programmation Web côté serveur : principalement PHP

Interactions navigateur/serveur web

Poste client

Site serveur





Lyon 1



département
Informatique

Université Claude Bernard Lyon 1

Le protocole HTTP

- Une transaction typique
- Format des requêtes/réponses
- Durée de vie des connexions, Cookies
- Différentes versions de HTTP, Proxy et caches web
- Les requêtes clientes : GET, POST et HEAD
- Les réponses du serveur
- Les en-têtes, les types MIME

Qu'est-ce que HTTP ?

- HTTP : Hyper Text Transfer Protocol
- Protocole régissant le dialogue entre des clients Web et un serveur (c'est le langage du Web !)
- Fonctionnement en mode Client/Serveur
- Une transaction HTTP contient
 - le type de la requête ou de la réponse (commande HTTP)
 - des en-têtes
 - une ligne vide
 - un contenu (parfois vide)
- Très peu de type de requêtes/réponses
- Port standard côté serveur : 80

Format des requêtes/réponses

- **Format des requêtes**

- type de la requête (METHOD, URL, version HTTP)

- en-têtes sur plusieurs lignes

- une ligne vide

- un contenu éventuel

- **Format des réponses**

- un code de réponse (version HTTP, code, description)

- en-têtes sur plusieurs lignes

- une ligne vide

- le contenu de la réponse

Une transaction typique (1)

- 1 - le client contacte le serveur pour demander le document index.html

GET /~ogluck/index2.html HTTP/1.1

- 2 - le client envoie des informations d'en-tête pour informer le serveur de sa configuration et des documents qu'il accepte

User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.1)

Host: www710.univ-lyon1.fr

Accept: image/gif, image/jpeg, */*

- 3 - le client envoie une ligne vide (fin de l'en-tête) et un contenu vide dans cet exemple

Une transaction typique (2)

- 4 - le serveur répond en commençant par indiquer par un code, l'état de la requête

HTTP/1.1 200 OK

- 5 - le serveur envoie un en-tête qui donne des informations sur lui-même et le document demandé

Date: Sun, 23 May 2004 17:46:01 GMT

Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18

Last-Modified: Sun, 23 May 2004 17:42:12 GMT

Content-Length: 90

Content-Type: text/html; charset=iso-8859-1

- 6 - puis une ligne vide (fin de l'en-tête) et le contenu du document si la requête a réussi

Une transaction typique (3)

```
xterm
ogluck@lima:~$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /~ogluck/index2.html HTTP/1.1
Host: localhost
Accept: */*

HTTP/1.1 200 OK
Date: Sun, 23 May 2004 17:46:01 GMT
Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
Last-Modified: Sun, 23 May 2004 17:42:12 GMT
ETag: "a805a-5a-40b0e274"
Accept-Ranges: bytes
Content-Length: 90
Content-Type: text/html; charset=iso-8859-1

<html><head><title>
index2.html
</title></head><body>
<h1>Bienvenue !</h1>
</body></html>
Connection closed by foreign host.
ogluck@lima:~$
```

Exemple 1-index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>
    1-index.html
  </title>
  <!-- style.css n'existe pas -->
  <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
  Bonjour Olivier !
</body>
</html>
```

Regarder avec Firebug « Réseaux »

- le 404 Not found pour style.css
- le format des deux requêtes/réponses HTTP

```
$ telnet localhost 80
```

```
GET /Sites/Exemples/CM4-HTTP/1-index.html HTTP/1.0
```

Exemple 2-index.php

```
<!DOCTYPE html>
<html> <head>
    <meta charset="utf-8" />
    <title> 2-index.php </title>
</head>
<body>
    <?php
        $url = $_SERVER['REQUEST_URI'];
        echo $url;
    ?>
    <br />
    <?php
        $tab = explode("/", $url);
        echo $tab[1];
    ?>
</body>
</html>
```

Regarder la réponse avec Firebug
« Réseaux » pour voir que le code
PHP a disparu

```
$ telnet localhost 80
GET /Sites/Exemples/CM4-HTTP/2-index.php HTTP/1.0
```

HTTP est un protocole sans état

- HTTP 1.0 (RFC 1945)
 - dès que le serveur a répondu à une requête, il ferme la connexion TCP
- HTTP 1.1 (RFC 2068)
 - par défaut, la connexion est maintenue tant que le serveur ou le client ne décide pas de la fermer (`Connection: close`)
- HTTP est un protocole **sans état**
 - aucune information n'est conservée entre deux requêtes successives : les requêtes sont indépendantes les unes des autres
 - permet au serveur HTTP de servir plus de clients en un temps donné (gestion légère des transactions)
 - **pour conserver des informations entre deux requêtes, il faut utiliser un *cookie*, des champs cachés d'un formulaire, ...**

Utilisation d'un champ HIDDEN

- Un champ de type HIDDEN permet de propager une variable d'une page à une autre.
- Souvent on propage un identifiant, un login, ...
- Par exemple, on aurait pu utiliser

```
<INPUT type="HIDDEN" NAME="score" VALUE="1024" />
```

pour transmettre le score du jeu 2048 entre deux actions de jeu plutôt que de le stocker dans le fichier `score.txt` sur le serveur.

Pour récupérer la valeur du score au début du programme PHP :

```
$score = $_GET['score']; // $score contient 1024, l'ancien score
```

```
$score = $score + 8; // nouveau score : deux 4 ont fusionné
```

Pour propager la nouvelle valeur du score vers la page suivante :

```
echo '<input type="hidden" name="score" value="' . $score . "' />';
```

Qu'est-ce qu'un cookie ?

- C'est un moyen pour le serveur de stocker des informations dans le navigateur client pour palier au caractère sans état du protocole HTTP
- Un cookie est une chaîne de caractères url-encodée de 4ko maximum stockée sur le disque dur du client
- Un cookie stocke des informations associées à un ensemble d'URL qui sont envoyées lors de chaque requête vers l'une de ces URL
- Les *cookies* permettent de
 - donner un identifiant au client (permet par exemple au serveur de ne pas compter deux fois le même navigateur pour un comptage du nombre de visites sur le site)
 - propager un code d'accès (évite une authentification lors de chaque requête)
 - propager une identification dans une base de données
 - propager l'identifiant d'une session PHP

Installation d'un cookie sur le client

- Directive Set-Cookie **dans l'en-tête de la réponse HTTP** (envoyé lors de la première connexion)

Set-Cookie: nom=valeur; expires=date; path=chemin_accès; domain=nom_domaine; secure

- le couple nom/valeur est le contenu du cookie (seul champ obligatoire), sans espace ; et ,
- le cookie devient invalide après la date indiquée
- `path=/pub` signifie que le cookie est valable pour toutes les requêtes dont l'URL contient `/pub`
- `domain` indique le nom de domaine (associé au serveur) pour lequel le cookie est valable
- `secure` : le cookie n'est valable que lors d'une connexion sécurisée

Utilisation d'un cookie par le client

- Chaque fois qu'un client va effectuer une requête, il vérifie dans sa liste de *cookies* s'il y en a un qui est associé à cette requête
- Si c'est le cas, le client utilise la directive Cookie **dans l'en-tête de la requête HTTP**

Cookie: nom1=valeur1; nom2=valeur2; ...

- Le serveur peut insérer plusieurs directives Set-Cookie
- Dans la première spécification des *cookies* :
 - un client peut stocker un maximum de 300 *cookies*
 - un maximum de 20 *cookies* par domaine est permis
 - la taille d'un *cookie* est limitée à 4Ko

Différentes versions du protocole HTTP

- Version d'origine : HTTP 0.9
 - Une seule méthode : GET
 - Pas d'en-tête
 - Une requête = une connexion TCP
- Amélioration en deux étapes
 - HTTP 1.0 :
 - Introduction des en-têtes pour échanger des **métadonnées** entre le navigateur et le serveur web
 - Nouvelles fonctionnalités : utilisation de caches, méthodes d'authentification (`.htaccess`)...
 - HTTP 1.1 :
 - Mode **connexions persistantes** par défaut
 - Introduction des serveurs virtuels -> l'en-tête `Host` est obligatoire dans la requête

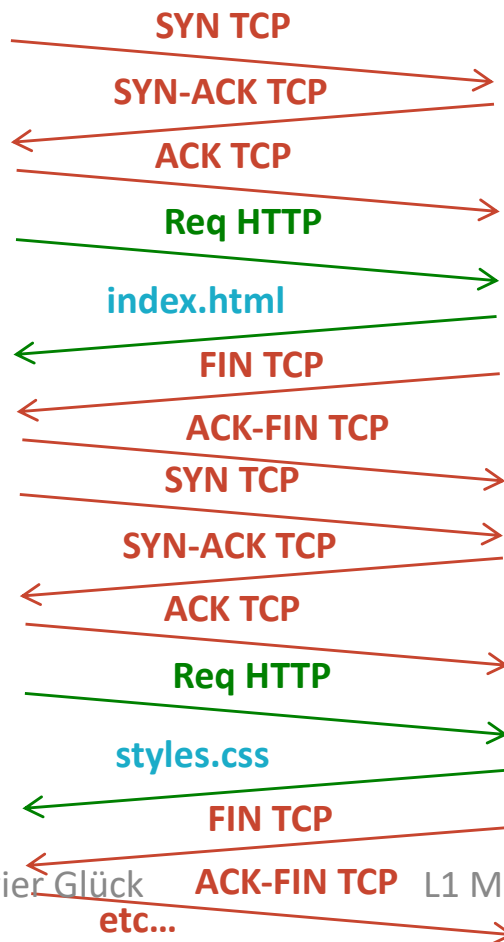
Intérêt des connexions persistantes

Exemple d'une page HTML contenant 1 feuille de styles et 2 images :

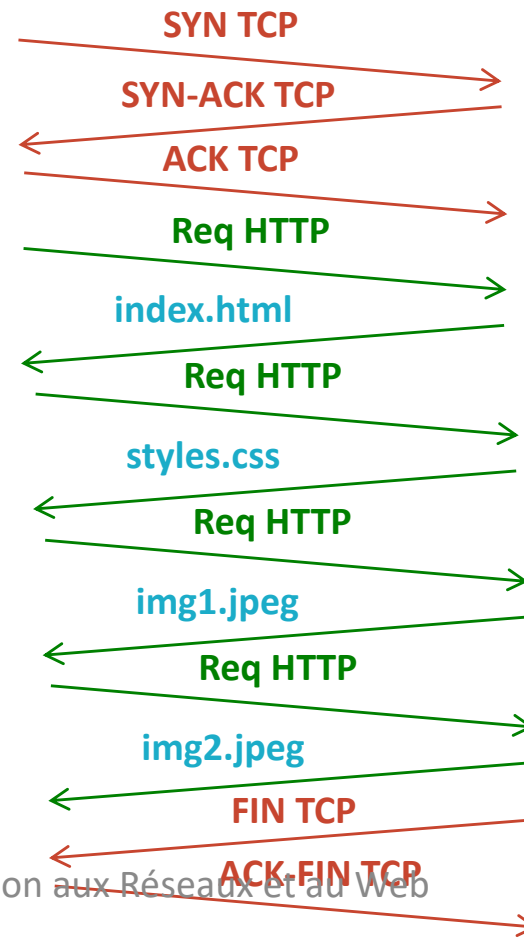
Avec HTTP 0.9, trois connexions/déconnexions TCP/IP

Avec HTTP 1.1, une seule connexion TCP/IP

Les connexions persistantes améliorent les performances.

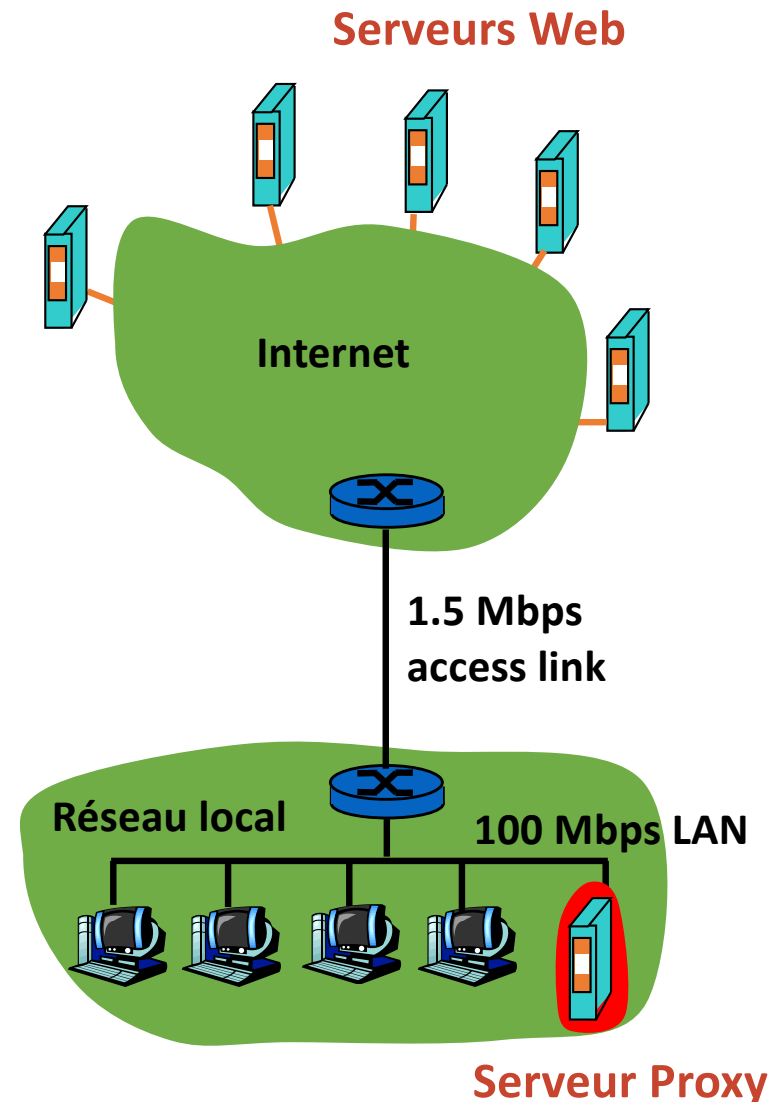


Laquelle est persistante ?



Intérêt d'un cache Web

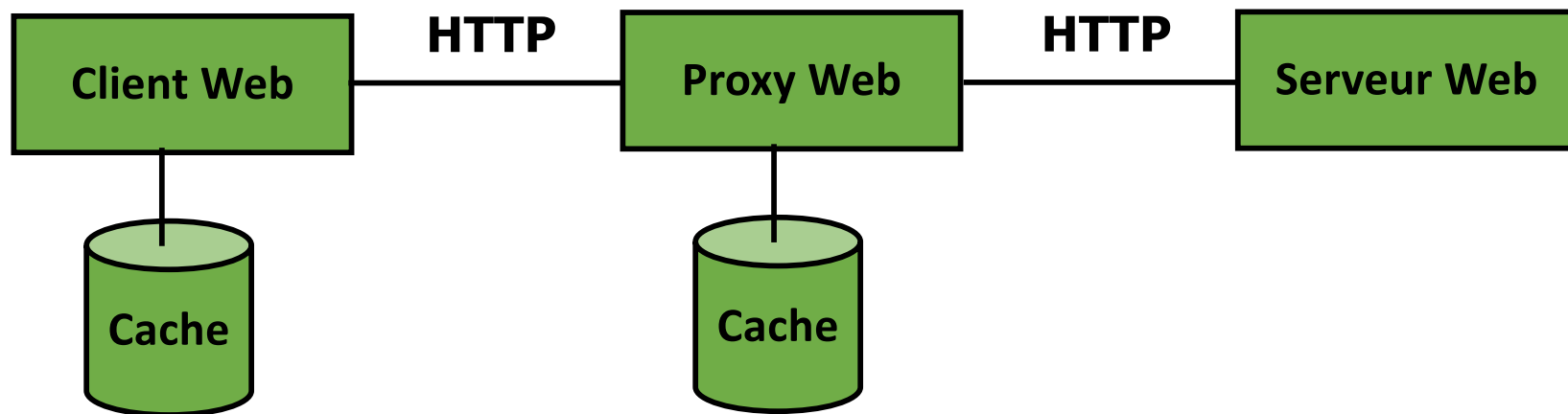
- Les pages qui sont demandées plusieurs fois sont conservées dans un cache.
 - Soulage le réseau car permet de ne pas redemander au serveur une page qui est déjà dans le cache.
 - L'accès a une page déjà dans le cache est plus rapide.
- Le cache peut être dans le navigateur ou sur un serveur relais (*proxy*) présent dans le réseau local de l'entreprise.
- Le cache améliore les performances.



Qu'est-ce qu'un proxy Web ?

Le proxy sert de cache. Il permet aussi de filtrer toutes les requêtes/réponses qui rentrent et sortent de l'organisation

Les filtres peuvent se faire sur des mots-clés présents dans l'URL ou dans le contenu des pages web.



Les requêtes HTTP faites par le client

- Format d'une requête HTTP (du client vers le serveur)
 - une commande HTTP (**METHOD**), une URL qui identifie la ressource demandée, la version de HTTP
 - l'en-tête et une ligne vide
 - éventuellement un contenu (corps de la requête)
- Méthode **GET**
- Méthode **POST**
- Méthode **HEAD**
- D'autres méthodes qui ne sont pas souvent supportées par les serveurs

La méthode GET

- La méthode standard de requête d'un document
 - Permet de récupérer un fichier HTML, une image, une feuille de styles, un fichier pdf...
 - Permet d'activer l'exécution d'un script PHP en lui transmettant des données en provenance d'un formulaire
- **Avec GET, le contenu de la requête est toujours vide**
- Le serveur répond avec une ligne décrivant l'état de la requête, un en-tête et le contenu demandé
- Si la requête échoue, le contenu de la réponse décrit la raison de l'échec (fichier non présent, non autorisé, ...)

Transmission des paramètres avec GET

- Comme le contenu d'une requête GET est vide, les données du formulaire sont transmises via l'URL après un ?

- Les champs sont séparés par un &

`GET /cgi-bin/prog.php?email=toto@site.fr&pass=toto&s=login HTTP/1.1`

- Ici, trois champs du formulaire sont transmis dans la requête : `email`, `pass` et `s`
- Attention : le mot de passe est transmis en clair !
- Permet de conserver l'URL dans un **favori** avec les données saisies dans le formulaire
- L'URL a une taille limitée qui dépend du navigateur et du serveur (conseillé de ne pas dépasser 2000 caractères)

La méthode POST

- Elle permet de transmettre des données au serveur dans le corps/contenu de la requête
- Exemple

POST /cgi-bin/prog.php HTTP/1.1

User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.1)

Host: localhost

Accept: */*

Content-type: application/x-www-form-urlencoded

Content-length: 36

email=toto@site.fr&pass=toto&s=login

- Le mot de passe est toujours transmis en clair !

La méthode HEAD (1)

- Identique à **GET** mais permet uniquement de récupérer les en-têtes relatifs à un document
- Permet de récupérer
 - la date de dernière modification du document (important pour les caches, JavaScript)
 - la taille du document (estimation du temps d'arrivée du document)
 - le type du document (le client peut sélectionner le type de documents qu'il accepte)
 - le type du serveur (permet de faire des requêtes spécifiques selon le type du serveur)
- Remarque : le serveur ne fournit pas nécessairement toutes ces informations !

La méthode HEAD (2)

```
xterm
ogluck@lima:~$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
HEAD /~ogluck/index2.html HTTP/1.0
Accept: */*

HTTP/1.1 200 OK
Date: Sun, 23 May 2004 18:14:14 GMT
Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
Last-Modified: Sun, 23 May 2004 17:42:12 GMT
ETag: "a805a-5a-40b0e274"
Accept-Ranges: bytes
Content-Length: 90
Connection: close
Content-Type: text/html; charset=iso-8859-1

Connection closed by foreign host.
ogluck@lima:~$
```

Autres requêtes HTTP

- PUT : permet de stocker le corps de la requête sur le serveur à l'URL spécifiée
- DELETE : suppression du document spécifié par l'URL
- OPTIONS : renvoie la liste des méthodes autorisées par le serveur
- TRACE : la corps de la requête entrante est renvoyée au client (utilisé pour faire du debug)
- ...

Les réponses HTTP renvoyées par le serveur

- Une réponse contient trois éléments sur la 1^{ère} ligne :
version HTTP, code de statut, description textuelle du code
HTTP/1.1 200 OK
HTTP/1.1 404 Not Found
- Le code est un entier sur 3 chiffres classé selon des catégories
 - 100-199 : message d'information
 - 200-299 : succès de la requête cliente
 - 300-399 : la requête n'est pas directement serviable, le client doit préciser certaines choses
 - 400-499 : échec de la requête qui incombe au client
 - 500-599 : échec de la requête qui incombe au serveur (par ex. erreur d'exécution d'un programme sur le serveur)

Quelques en-têtes de requêtes

- Identification du client

`From` (adresse mail du client), `Host` (serveur, **obligatoire en HTTP1.1**), `Referer` (URL d'où l'on vient), `User-Agent`

- Préférences du client

`Accept` (liste des types MIME acceptés), `Accept-Encoding` (`compress|gzip|...`), `Accept-Language`, `Accept-Charset`

- Information pour le serveur

`Autorization` (`username:passwd` encodé en base64), `Cookie`

- Conditions sur la réponse

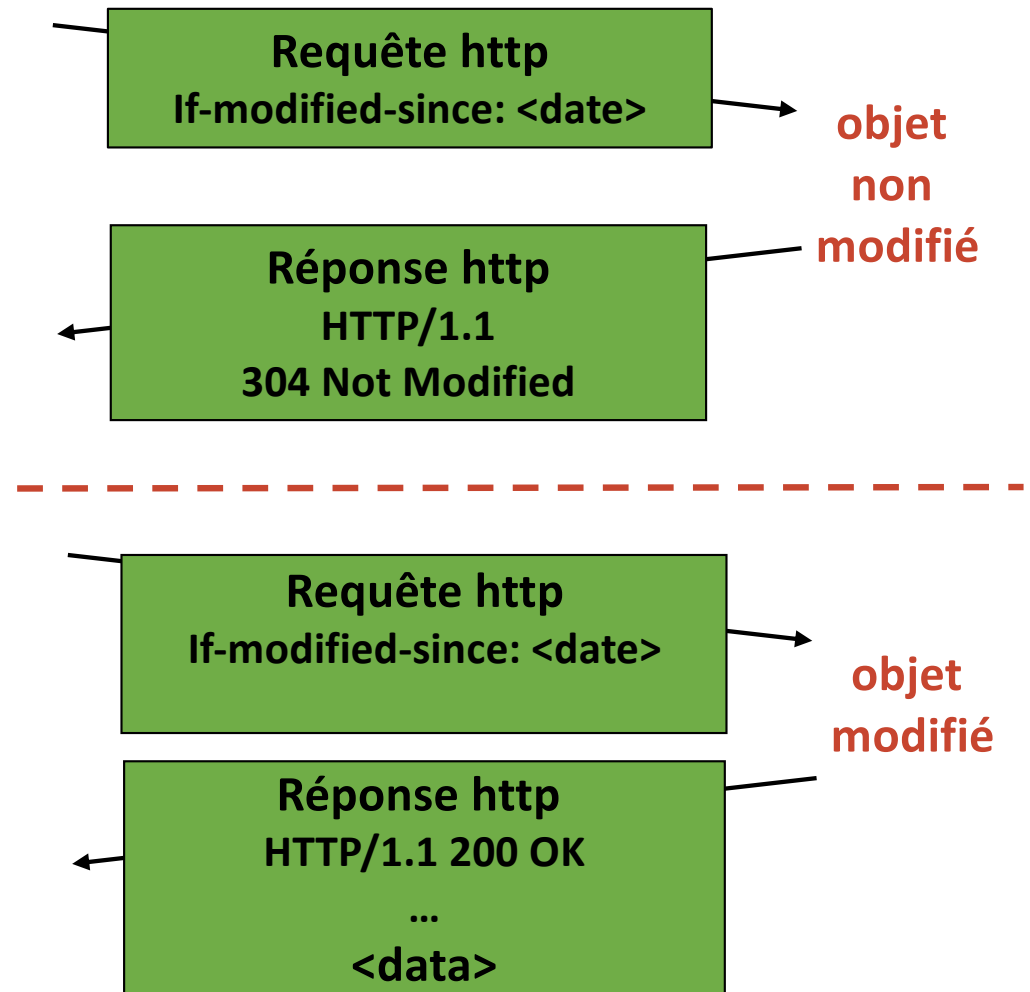
`If-Modified-Since` (utile pour les caches),
`If-Unmodified-Since`, `If-Match` (Etag)

L'en-tête If-Modified-Since

- Objectif : ne pas envoyer un objet que le client a déjà dans son cache
- Problème : les objets contenus dans le cache peuvent être obsolètes
- Le client spécifie la date de la copie cachée dans la requête http
If-modified-since: <date>
- La réponse du serveur est vide si la copie du cache est à jour

Client

Serveur



Quelques en-têtes de réponses

- Informations sur le contenu du document

- Content-Type (type MIME du document),
 - Content-Length (barre de progression du chargement),
 - Content-Encoding, Content-Location,
 - Content-Language

- Informations sur le document

- Last-Modified (date de dernière modification),
 - Allow (méthodes autorisées pour ce document),
 - Expires (date d'expiration du document)

- En-tête générales

- Date (date de la requête), Server (type du serveur)

Les types MIME

- MIME : *Multi-purpose Internet Mail Extensions*
- Permet l'échange de fichiers multimédias entre machines quelconques en spécifiant le type du fichier
- Les commandes MIME ont été intégrées dans HTTP1.0
- Un type MIME est composé
 - d'un type général (text, image, audio, video, application...)
 - et d'un sous-type (image/gif, image/jpeg, application/pdf, application/rtf, text/plain, text/html)
- En perpétuelle évolution
- La machine cliente doit ensuite associer l'exécution d'une application à chaque type MIME
- Le serveur positionne **Content-type** à partir de l'extension du document demandé (/etc/mime.types)



Lyon 1



département
Informatique

Université Claude Bernard Lyon 1

La programmation CGI

Qu'est-ce que la programmation CGI ?

Un premier exemple

Récupérer les données du formulaire avec GET/POST

Format URL encodé

Format de la sortie standard

Les variables d'environnement

CGI - Common Gateway Interface

- Interface de base qui définit la communication entre le serveur HTTP et un programme d'application
- CGI spécifie comment des navigateurs clients peuvent communiquer avec des programmes qui s'exécutent sur le serveur Web et qui génèrent des pages HTML dynamiques **créées à la volée** à partir du résultat des exécutions

Qu'est ce qu'un programme CGI ?

- Un programme
 - qui s'exécute sur la machine hébergeant le serveur HTTP
 - en langage compilé (binaire) ou interprété (script)
 - qui permet de
 - récupérer les données du formulaire à l'aide d'un *parser* :
pour chaque champ, un couple NAME/VALUE est transmis au serveur
 - effectuer des traitements sur le serveur
 - lecture/écriture dans une base de données
 - stockage d'une info (compteurs, identifiant de connexion, ...)
 - recherche d'une info
 - pied de page automatique (ex : date de dernière modification)
 - générer un résultat qui est renvoyé au client
 - page HTML, image, document postscript, ...

Avantages/inconvénients

- Puissant mais dangereux
 - Permet d'exécuter tout et n'importe quoi par le démon HTTP du serveur
- Un CGI doit s'exécuter rapidement
 - Risque de surcharge du serveur
 - Utilisateurs impatients : pendant que le CGI s'exécute, le client attend la réponse sans savoir pourquoi elle n'arrive pas...
 - Possibilité d'envoyer dès le début de l'exécution une page qui permet d'indiquer à l'utilisateur que le résultat va arriver (transfert par morceaux en HTTP 1.1)

Un premier exemple (1)

```
#!/bin/sh
# Date.cgi
echo 'Content-type: text/html'
echo ''
#Création du corps du document
echo '<HTML><HEAD><TITLE>'
echo 'Date.cgi'
echo '</TITLE></HEAD><BODY>'
echo '<H1>Date sur le serveur</H1>'
echo -n "On est le `date +%D`, il est "
echo "`date +%H`h `date +%M`m"
echo '</BODY></HTML>'
```

Source du programme CGI

```
ogluck@lima:~/public_html/cgi-bin$
./Date.cgi
Content-type: text/html

<HTML><HEAD><TITLE>
Date.cgi
</TITLE></HEAD><BODY>
<H1>Date sur le serveur</H1>
On est le 11/07/03, il est 11h 30m
</BODY></HTML>
```

Exécution du CGI sur le serveur

Un premier exemple (2)

Exécution du CGI depuis le client



Un premier exemple (3)

- Ce programme CGI n'utilise aucune donnée en provenance du client
- Il récupère simplement la date sur le serveur et affiche sur sa **sortie standard** le code d'une page HTML minimale contenant la date et l'heure
- La ligne "**Content-type: text/html**" est une information destinée au serveur pour la construction de l'en-tête HTTP constituant la réponse renvoyée au client (ici, il s'agit d'indiquer que le type des données générées par le CGI est une suite de commandes HTML)

Méthodes GET/POST (1)

- Voici le code d'un petit script CGI en shell

```
#!/bin/sh
```

```
# Get_Post.cgi
```

```
echo 'Content-type: text/plain'
```

```
echo ''
```

```
echo "QS=$QUERY_STRING"
```

```
read DATA
```

```
echo "Data=$DATA"
```

- Les résultats de l'exécution avec la méthode GET puis POST sont montrés dans les deux transparents suivants

Méthodes GET/POST (2)

```
xterm
ogluck@lima:~$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /~ogluck/cgi-bin/Get_Post.cgi?email=toto@site.fr&pass=toto&s=login HTTP/1.1
Host: localhost
Accept: */*

HTTP/1.1 200 OK
Date: Sun, 23 May 2004 18:25:26 GMT
Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
Transfer-Encoding: chunked
Content-Type: text/plain; charset=iso-8859-1

2e
QS=email=toto@site.fr&pass=toto&s=login
Data=
0

Connection closed by foreign host.
ogluck@lima:~$
```

Méthodes GET/POST (3)

```
xterm
ogluck@lima:~$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
POST /~/ogluck/cgi-bin/Get_Post.cgi HTTP/1.1
Accept: */*
Host: localhost
Content-type: application/x-www-form-urlencoded
Content-length: 36

email=toto@site.fr&pass=toto&s=login
HTTP/1.1 200 OK
Date: Sun, 23 May 2004 18:29:52 GMT
Server: Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
Transfer-Encoding: chunked
Content-Type: text/plain; charset=iso-8859-1

4
QS=
2a
Data=email=toto@site.fr&pass=toto&s=login
0


Connection closed by foreign host.
ogluck@lima:~$
```

Méthodes GET/POST (4)


- Avec la méthode GET
 - les données relatives aux champs du formulaire sont transmises via l'URL (dans le type de la requête)
 - le programme CGI les récupère dans la variable d'environnement **QUERY_STRING**
 - il est possible de cliquer sur "Actualiser" pour retransmettre les données et de définir un *bookmark*
- Avec la méthode POST
 - les données relatives aux champs du formulaire sont transmises dans le corps de la requête HTTP
 - **Content-type** et **Content-length** sont positionnés
 - le programme CGI les récupère sur l'entrée standard
 - "Actualiser" et *bookmark* impossibles, données du formulaire non visibles dans les logs du serveur

Méthodes GET/POST (5)

Formulaire

Adresse  E:\Cours\Lyon1\DESS_Reseaux\C4\CGI\get_post.html

Méthode GET

Adresse  http://lima/cgi-bin/Get_Post.cgi?email=toto@site.fr&pass=toto&s=login

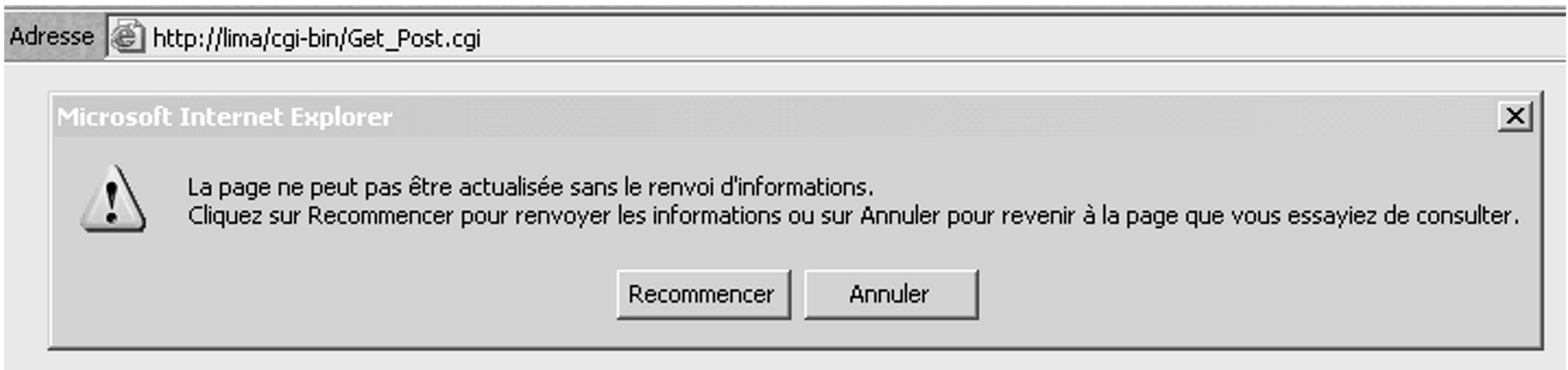
QS=email=toto@site.fr&pass=toto&s=login
Data=

Méthode POST

Adresse  http://lima/cgi-bin/Get_Post.cgi

QS=
Data=email=toto@site.fr&pass=toto&s=login

Méthode POST et "Actualiser"



Récupérer les données du formulaire

- Format de la chaîne CGI
 - `nom_champ1=valeur1&nom_champ2=valeur2&...`
- Cas des champs à valeurs multiples
 - exemple : listes à sélection multiples
 - `nom_liste=valeur1&nom_liste=valeur2&...`
- La chaîne CGI
 - elle est construite par le client au format *URL-encoded* quand la requête est postée
 - elle est transmise au CGI tel quel via la variable d'environnement `QUERY_STRING` avec la méthode `GET`
 - elle est transmise au CGI tel quel via l'entrée standard avec la méthode `POST`
 - en PHP, `$_GET['nom_champ1']` ou `$_POST['nom_champ1']` contient la valeur du champ

Format URL encodé

- Nécessité de coder les données de l'URL (méthode GET) sur le client pour construire la chaîne CGI pour respecter la RFC 2396 qui spécifie la syntaxe des URL
- Les caractères non-alphanumériques sont remplacés par %xx (xx=code ASCII du caractère en hexadécimal)
- Les caractères ; / ? : @ & = + \$ et , sont réservés pour une signification particulière dans l'URL
 - ? : début de QUERY_STRING
 - & : séparateur de champ
 - = : séparation entre le nom du champ et sa valeur
- Les espaces sont remplacés par des +

Exemple 3-compteur-get.php

```
<!DOCTYPE html>
<html> <head> <meta charset="utf-8" />
    <title>3-compteur-get.php </title>
</head>
<body> <form name="compteur" method="get" action="3-compteur-get.php">
    <?php
        if (isset($_GET['compteur']))
            $cpt = $_GET['compteur'];
        else
            $cpt = 0;
        $cpt = $cpt + 1;
        // Pour propager la nouvelle valeur de cpt vers la page suivante
        echo '<input type="hidden" name="compteur" value="'. $cpt. '"/>';
        echo "Le compteur vaut $cpt.\n";
    ?>
    <input type="submit" name="action" value="+1" />
</form></body>
</html>
```

Regarder la réponse avec Firebug
Changer la valeur dans l'URL

```
$ telnet localhost 80
GET /Sites/Exemples/CM4-HTTP/3-compteur-
get.php?compteur=5&action=%2B1 HTTP/1.0
```

Exemple 3-compteur-post.php

```
<!DOCTYPE html>
<html> <head> <meta charset="utf-8" />
    <title>3-compteur-get.php </title>
</head>
<body> <form name="compteur" method="post" action="3-compteur-post.php">
    <?php
        if (isset($_POST['compteur']))
            $cpt = $_POST['compteur'];
        else
            $cpt = 0;
        $cpt = $cpt + 1;
    ?>
    <input type="text" value="" />
</form></body>
</html>
```

Regarder avec Firebug les en-têtes et la réponse après +1

```
$ telnet localhost 80
POST /Sites/Exemples/CM4-HTTP/3-compteur-
post.php HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 22

compteur=3&action=%2B1
```

Exemple 4-form.html

Regarder avec Firebug ce qui est transmis

```
<HTML><HEAD><TITLE>Eléments d'un formulaire</TITLE>
</HEAD><BODY>
<FORM name="f1" METHOD="POST" ACTION="4-form.html">
<INPUT NAME="email" VALUE="entrez votre email ici" SIZE="30" MAXLENGTH="50" />
<INPUT type="PASSWORD" NAME="pass" VALUE="entrez votre passwd ici" SIZE="8" MAXLENGTH="8" /><BR />
<INPUT type="CHECKBOX" NAME="cours1" VALUE="1" CHECKED />HTML
<INPUT type="CHECKBOX" NAME="cours1" VALUE="2" CHECKED />JS
<INPUT type="CHECKBOX" NAME="cours1" VALUE="3" />CGI<BR />
<INPUT type="RADIO" NAME="cours2" VALUE="1" />HTML <INPUT type="RADIO" NAME="cours2" VALUE="2" CHECKED />JS
<INPUT type="RADIO" NAME="cours2" VALUE="3" />CGI<BR />
<INPUT type="SUBMIT" NAME="s" VALUE="login" /> <INPUT type="SUBMIT" NAME="s" VALUE="logout" />
<INPUT type="RESET" NAME="raz" VALUE="Effacer" /><BR />
<INPUT type="hidden" name="MAX_FILE_SIZE" value="1048576" />
<INPUT type="FILE" NAME="fichier" VALUE="Parcourir" /><BR />
<TEXTAREA NAME="t1" ROWS="3" COLS="40"> Entrez vos remarques ici</TEXTAREA>
<SELECT NAME="pop"><OPTION VALUE="v1" />1 <OPTION VALUE="v2" SELECTED />2 <OPTION VALUE="v3" />3
</SELECT>
<SELECT NAME="mul" SIZE="3" MULTIPLE>
<OPTION VALUE="v1" />1 <OPTION VALUE="v2" SELECTED />2 <OPTION VALUE="v3" SELECTED />3 <OPTION VALUE="v4" />4
</SELECT> </FORM></BODY></HTML>
```

entrez votre email ici

.....

HTML JS CGI

HTML JS CGI

login

logout

Effacer

Parcourir...

Aucun fichier sélectionné.

Entrez vos remarques ici

2

1

2

3

État	Méth...	Fichier	En-têtes	Cookies	Paramètres	Réponse
200	POST	4-form.html	Filtrer les paramètres de la requête			
Données de formulaire						
email : "entrez+votre+email+ici"						
pass : "entrez+votre+passwd+ici"						
cours1 : "1"						
cours1 : "2"						
cours2 : "2"						
s : "login"						
MAX_FILE_SIZE : "1048576"						
fichier : ""						
t1 : "++Entrez+vos+remarques+ici"						
pop : "v2"						
mul : "v2"						
mul : "v3"						

Pour afficher tous les champs du formulaire

```
// afficher la chaîne CGI en méthode GET
```

```
echo $_SERVER['QUERY_STRING'];
```

```
echo "<br />";
```

[Voir 4-form.php](#)

```
// afficher tous les champs transmis en méthode GET
```

```
foreach ($_GET as $name => $value) {
```

```
    echo "$name = $value<br />\n";
```

```
}
```

```
// afficher tous les champs transmis en méthode POST
```

```
foreach ($_POST as $name => $value) {
```

```
    echo "$name = $value<br />\n";
```

```
}
```

Format de la sortie standard d'un CGI

- En-tête, ligne vide, Corps

Content-type: type/subtype (type MIME du corps)

Window-target: _blank (fenêtre de réception du résultat)

Location: URL (redirection vers une autre URL)

Status: code msg (code de la réponse HTTP)

<HTML>

...

</HTML>

- Location doit être utilisé seul - peut être utile par exemple pour utiliser un moteur de recherche existant
- En-tête minimale : Content-type
- Les autres en-têtes sont ajoutées par le serveur Web

Les variables d'environnement (1)

- Elles sont positionnées par le serveur HTTP pour fournir au CGI des infos sur le serveur, le client, ...

SERVER_SOFTWARE : nom/version

nom et version du démon HTTP

SERVER_NAME : nom

nom ou adresse IP de la machine serveur HTTP

GATEWAY_INTERFACE : CGI/version (CGI/1.1)

version des spécifications CGI utilisées par le serveur

SERVER_PROTOCOL : protocole/version (HTTP/1.1)

protocole et version de la requête en cours de traitement

SERVER_PORT : port

numéro du port (TCP) vers lequel la requête a été envoyée

Les variables d'environnement (2)

REQUEST_METHOD : method (GET/POST/...)

méthode associée à la requête en cours de traitement

SCRIPT_NAME : nom (/cgi-bin/mon_cgi.cgi)

chemin du CGI à partir de la racine du serveur HTTP

REMOTE_HOST : nom

nom de la machine d'où vient la requête

REMOTE_ADDR : adresse_IP

adresse IP de la machine d'où vient la requête

AUTH_TYPE : authentication

méthode d'authentification de l'utilisateur s'il y a lieu

REMOTE_USER : login

si authentification, nom de l'utilisateur associé à la requête

REMOTE_IDENT : login_os

login de connexion de l'utilisateur (pas souvent supporté)

Les variables d'environnement (3)

CONTENT_TYPE : type/subtype (application/x-www-form-urlencoded)

type MIME des données véhiculées dans la requête

CONTENT_LENGTH : lg (en octets)

longueur des données véhiculées dans la requête (POST)

PATH_INFO : path

chaîne entre SCRIPT_PATH et QUERY_STRING dans l'URL

QUERY_STRING : nom1=val1&nom2=val2...

données transmises au CGI via l'URL (GET)

pour afficher la chaîne : `echo $_SERVER['QUERY_STRING'];`

HTTP_XXX (une variable pour chaque champ contenu dans l'en-tête HTTP de la requête)

HTTP_ACCEPT, HTTP_USER_AGENT, ...

```
DOCUMENT_ROOT --> /var/www
GATEWAY_INTERFACE --> CGI/1.1
HTTP_ACCEPT --> */*
HTTP_ACCEPT_ENCODING --> gzip, deflate
HTTP_ACCEPT_LANGUAGE --> fr
HTTP_CONNECTION --> Keep-Alive
HTTP_HOST --> lima
HTTP_USER_AGENT --> Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
PATH --> /bin:/usr/bin:/sbin:/usr/sbin
QUERY_STRING --> email=entrez+votre+email+ici&pass=toto&s=login
REMOTE_ADDR --> 140.77.13.102
REMOTE_PORT --> 3304
REQUEST_METHOD --> GET
REQUEST_URI --> /cgi-bin/Env.cgi?email=entrez+votre+email+ici&pass=toto&s=login
SCRIPT_FILENAME --> /home/ogluck/public_html/cgi-bin/Env.cgi
SCRIPT_NAME --> /cgi-bin/Env.cgi
SERVER_ADDR --> 140.77.13.131
SERVER_ADMIN --> olivier.gluck@ens-lyon.fr
SERVER_NAME --> lima
SERVER_PORT --> 80
SERVER_PROTOCOL --> HTTP/1.1
SERVER_SIGNATURE -->
Apache/1.3.28 Server at lima Port 80

SERVER_SOFTWARE --> Apache/1.3.28 (Debian GNU/Linux) PHP/3.0.18
UNIQUE_ID --> P7J-K4xNDYMAAAMwBLI
```

Quel(s) formulaire(s) pour le réseau social ?

https://lifasr2.univ-lyon1.fr/prenom.nom/reseau-social/res.php

Rechercher

Trier l'affichage

- Images
- Messages
- Avec commentaires
- Sans commentaires

Nombre de posts ? 10

Afficher

Bonjour prenom.nom

Quoi de neuf ?

Ajouter une image : Parcourir... Aucun fichier sélectionné.

Poster

prenom.nom
2017-02-21 17:31:32

Bonjour à tous !

Aimer 0 Commenter

Votre commentaire ?