

Introduction aux applications Web

Daniel Hagimont

<https://www.google.fr/search?q=daniel+hagimont+home+page>

1

Ce cours est une introduction aux applications Web. Le but est de vous donner les grands principes qui jalonnent toutes les technologies du domaine.

Le but n'est pas de vous initier à la dernière techno à la mode, parce que :

- il n'y en a pas qu'une
- elle peut changer tous les deux ans
- le but n'est pas de connaître une techno, mais d'être capable de s'adapter aux technos (et pour ça il faut connaître les principes)

Principe



- Système client-serveur
- Trois aspects
 - ◆ Désignation et localisation des documents (URI/URN/URL)
 - ◆ Codage des documents (HTML, CSS)
 - ◆ Protocole de requête et transfert de documents (HTTP)

2

Le Web repose sur un modèle client-serveur (que vous avez pu découvrir dans le cours Intergiciels). Ici le client est un navigateur (comme Firefox) et le serveur est un serveur Web (comme Apache).

Il y a trois aspects principaux dans le Web :

- la désignation et localisation des documents avec des URI/URN/URL
- l'encodage des documents avec les langages HTML et CSS
- le protocole de communication (HTTP) entre le navigateur et le serveur Web

Désignation et localisation

- URI : identifie une ressource par nom (URN) ou par localisation (URL)
- URL : spécifie la localisation d'une ressource et le moyen d'y accéder
 - ◆ <http://www.inria.fr>
 - ◆ <mailto:hagimont@enseeiht.fr>
- URN : ne suppose pas la disponibilité de la ressource
 - ◆ URN:ISBN:0-395-36341-1
 - ◆ Motivation : éviter les URL cassées
 - ◆ Nécessite une traduction / peu utilisé

3

Initialement, le W3C (World Wide Web Consortium), organisme de standardisation, a défini des notions de URN (Uniform Resource Name) qui sont des noms de documents et de URL (Uniform Resource Locator) qui sont des localisations de documents. L'idée était que la localisation (URL) d'un document peut changer, son nom (URN) restant inchangé. Cependant, les URN n'ont été que très peu utilisées et les URL sont utilisées pour désigner et localiser les documents.

La notion de URI (Uniform Resource Identifier) est une super-classe des classes URN et URL : un URI est soit un URN, soit une URL.

Langage HTML

- HTML : Hyper Text Markup Language
 - ◆ Langage de description d'information structurée portable
 - ◆ Organisme de standardisation : WWW consortium (W3C)
- Histoire
 - ◆ 2000-2004 : orientation vers XML (XHTML) (W3C)
 - ▶ syntaxe plus rigoureuse
 - ◆ 2004 : contestation des fabricants de navigateurs
 - ▶ Web Hypertext Application Technology Working Group (WHATWG)
 - ◆ 2008 : HTML5 (W3C)
 - ▶ reprend les productions des 2 groupes
 - ◆ 2012 : HTML Living Standard (WHATWG)
 - ▶ standard en évolution constante

4

La structuration du contenu des documents repose sur le langage HTML (HyperText Markup Language), standardisé par le W3C.

Historiquement, HTML n'était pas très bien structuré, par exemple certaines balises n'étaient jamais fermées (langage mal parenthésé). Il a évolué vers XHTML, une version compatible avec XML (et bien parenthésée).

Les fabricants de navigateurs ont contesté cette version, créant le groupe de travail WHATWG.

Cette divergence a mené à la définition de HTML5 qui a repris les contributions du W3C et du WHATWG. Puis le WHATWG a annoncé le HTML Living Standard, un standard en évolution constante qui est devenu le standard "de facto" de HTML (car proposé par les fabricants de navigateurs).

En 2019, le W3C a annoncé que le standard de HTML serait celui du WHATWG.

Bref, on trouve encore pleins de choses pas très propres dans HTML.

Langage HTML

- `<balise attributs> contenu </balise>`
 - ◆ balise (tag), ex : `<html>`
 - ◆ attributs : représente des options, ex : `<table width="60%">`
 - ◆ contenu : texte, images ou d'autres balises
 - ◆ `</balise>` : fin de la balise

```
<HR>
<IMG SRC="monimage.gif" WIDTH=100 HEIGHT=120>
<H1>Ceci est un titre</H1>
<h2>Ceci est un sous-titre</h2>
<A HREF="http://host/dir/file.html">lien</A>
<!-- Commentaire -->
```

5

Le langage HTML est composé de balises qui encadrent un contenu (comme des parenthèses). Par exemple, les balises `<html>` et `</html>` encadrent en entier une page HTML.

Une balise peut avoir des attributs (paramètres), par exemple ici la balise `<table>` (pour définir un tableau) prend un attribut "width" pour définir sa largeur.

Le contenu, entre une balise ouvrante et une balise fermante, peut être du texte ou d'autres balises. Une balise peut également faire référence à une image ou une autre page HTML

Dans cet exemple, on a respectivement :

`<HR>` correspond à une ligne horizontale

`` correspond à l'inclusion d'une image dans la page, avec des attributs de largeur et hauteur. `monimage.gif` est une URL relative (le fichier est au même endroit que la page courante), mais on aurait pu mettre une URL absolue vers un autre site.

`<H1>` correspond à un titre de premier niveau. Le texte sera formaté en gros.

`<h2>` correspond à un titre de niveau 2.

`` correspond à un hyperlien. Le texte "lien" sera présenté en souligné et lorsqu'on clique dessus, la page dont l'URL est donnée sera chargée dans le navigateur.

Langage HTML

- Structure page HTML

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Page title</title>  
  </head>  
  <body>  
    <h1>This is a heading</h1>  
    <p>This is a paragraph.</p>  
    <p>This is another paragraph.</p>  
  </body>  
</html>
```

```
HTML5 : <!DOCTYPE html>  
HTML4.01 : <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
XHTML1.0 : <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Une balise en début de page permet d'indiquer le standard de HTML qui est utilisé.

Principales balises

- `<a>` lien hypertexte attribut : *href* pour une URL ou *name* pour une ancre
- `` met en emphase une portion de texte
- `` inclut une image dans le document, attributs : *alt* (texte alternatif) et *src* (chemin vers l'image)
- `<div>` conteneur générique de type bloc
- `<p>` paragraphe de texte
- `<table>` écriture d'un tableau voir aussi `<tr>`, `<td>`, `<th>`
- `<h1>`, `<h2>`, . . . `<h6>` niveaux de titres
- ``, `` listes ordonnées ou à puces simples, chaque élément (item) sera écrit ``
- `<form>` formulaire interactif

7

Les principales balises de HTML sont :

`<a>` pour définir un lien (cliquable) vers une autre page ou une ancre (une section de la page courante)

`` pour mettre en emphase un texte

`` pour inclure une image

`<div>` pour définir un bloc (et contrôler ses attributs)

`<p>` pour définir un paragraphe de texte

`<table>` pour définir un tableau

`<h1><h2>` ... pour définir des titres de différents niveaux

`` pour définir des listes

`<form>` pour définir des formulaires de saisie de données (détaillé plus loin)

Mise en page



- CSS : Cascading Style Sheet
 - ◆ Feuilles de style
 - ▶ HTML : contenu / CSS : formattage
 - ◆ Egalement une guerre des standards
 - ◆ <http://www.w3.org/TR/CSS2>
- Documentations
 - ◆ Valider son code HTML
 - ▶ <http://validator.w3.org/>
 - ◆ Valider son code CSS
 - ▶ <http://jigsaw.w3.org/css-validator/>
 - ◆ Tutoriels HTML - CSS
 - ▶ <http://www.w3schools.com/>
 - ◆ Bootstrap ...

8

HTML permet de définir la structure et le contenu d'une page. On peut également définir sa présentation (sa mise en page) en utilisant les attributs des balises, afin de définir la taille, la fonte, la couleur du texte.

Mais ce n'est pas très bien de faire comme cela, car on mélange le contenu et la présentation. Il est intéressant de les séparer, car on peut ainsi changer la présentation en appliquant un style sans changer le document (comme on le fait avec Latex).

Le langage CSS permet de définir des feuilles de style décrivant la présentation associée à chaque type de balise ou à chaque balise. Il y a également une guerre des standards pour CSS.

Quelques références sont données, à aller regarder si vous vous voulez en savoir plus.

Syntaxe CSS

```
/* les titres de types h1 seront en bleu et centrés */
h1 {
  color: blue ;
  text-align: center;
}
/* on peut appliquer des propriétés à des classes d'objet :
ici seuls les paragraphes de classe 'note' auront cette propriété */
p.note {
  font-family: Arial, sans-serif;
  font-style: italic;
}
/* les balises html peuvent aussi être dotées d'un attribut
id qui doit alors être unique dans la page */
p#menu {
  font-weight: bold;
}
```

→ `<p class="note">Hello</p>`

→ `<p id="menu">Call</p>`

9

Quelques exemples de styles CSS.

Syntaxe CSS

- Dans l'entête du document

```
<head >
...
<style type="text/css">
  h1 {color: blue;}
</style>
...
</head>
```

- Dans les balises

```
<h1 style=" color: blue;">
  Titre de la page
</h1>
```

- Référence à une feuille externe (le mieux)

```
<head>
...
<link rel="stylesheet" type="text/css" href="styles.css" />
...
</head>
```

10

On peut inclure directement ces règles CSS dans la section `<head>` de la page. C'est pas mal (c'est séparé du contenu), mais pas idéal.

On peut mettre de telles règles sous la forme d'attributs dans les balises. Là c'est carrément pas bien.

On peut mettre dans la section `<head>` de la page une référence à une feuille de style CSS qui est dans un fichier séparé (ça c'est le mieux).

Syntaxe CSS

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <style type="text/css">
      h1 {color: blue;}
      p.note {font-style: italic;}
      p#menu {font-weight: bold;}
    </style>
  </head>
  <body>
    <h1>Un titre</h1>
    <div>Un bloc.
    <p id="menu">Un paragraphe.</p>
    <p class="note">Un deuxième paragraphe (accent utf8).</p>
    <p>Un troisième paragraphe (accent html).</p>
    </div>
  </body>
</html>
```

Un titre

Un bloc.

Un paragraphe.

Un deuxième paragraphe (accent utf8).

Un troisième paragraphe (accent html).

11

Voici un petit exemple complet, avec le style CSS dans la section <head>.

La figure illustre le rendu dans le navigateur.

Différents styles d'une même page

The image displays four variations of a web page titled "Welcome to My Homepage". Each variation demonstrates a different CSS style sheet applied to the same underlying HTML content.

- Top Left (Default):** Features a white background with a yellow brushstroke at the top. The navigation menu is a simple list. The "Same Page Different Stylesheets" section has a white background with blue links.
- Top Right (Light Green):** The background is a solid light green. The navigation menu is a list with a light grey background. The "Same Page Different Stylesheets" section has a white background with blue links. A blue sidebar is on the right.
- Bottom Left (Red):** The background is a solid red. The navigation menu is a list with red buttons. The "Same Page Different Stylesheets" section has a white background with red buttons for the links.
- Bottom Right (Dark Green):** The background is a solid dark green. The navigation menu is a list with a dark grey background. The "Same Page Different Stylesheets" section has a white background with green links. A dark sidebar is on the right.

On voit sur cet exemple que le même contenu peut être présenté de manières très différentes avec des feuilles de style différentes.

Le protocole HTTP

- HTTP : HyperText Transfer Protocol
 - ◆ Modèle client-serveur pour le transfert des documents hypertextes
 - ◆ Protocole utilisé par les serveurs Web depuis 1990
 - ◆ Protocole minimaliste basé sur TCP/IP et utilisant des messages sous forme de chaînes de caractères
- URL : Uniform resource Locator
 - ◆ method://machine[:port]/fichier[#ancre]?param]
 - ◆ Method: file/ftp/http/telnet/news/mailto
- Transaction HTTP
 - ◆ initialement one-shot, mais keep-alive + pipeline en HTTP 1.1
- HTTPS sur SSL/TLS

13

HTTP (HyperText Transfer Protocol) est le protocole de communication utilisé pour les requêtes (du navigateur vers le serveur) et les réponses (du serveur vers le navigateur). Il décrit le format des messages échangés à l'aide de TCP/IP. Les messages échangés contiennent des chaînes de caractères.

Une URL désigne une page qui peut être chargée par un navigateur depuis un serveur (typiquement en cliquant sur un lien (balise)).

Une URL est composée de différents champs : la méthode d'accès (protocole), la machine (nom DNS du serveur), le numéro de port (par défaut 80 pour HTTP), le fichier (un path sur le serveur), une ancre (repère dans le document) et des paramètres (nous verrons ça dans la partie Web dynamique). HTTP est une des méthodes d'accès, mais il y en a pleins d'autres (par exemple FTP).

Une transaction HTTP est composée d'une requête et une réponse. Initialement, HTTP était implanté "one-shot" ce qui signifie qu'une transaction ouvrait une connexion TCP, envoyait la requête et recevait la réponse, et fermait la connexion (donc une seule transaction par connexion TCP). Plus tard, le "keep-alive" a été introduit, ce qui permet au navigateur de conserver une connexion

Enfin, HTTPS est l'implantation de HTTP sur SSL/TLS, donc chiffrant les communications entre le navigateur et le serveur Web.

Requête HTTP

Méthodes	method URL HTTP_version	GET, HEAD, POST ...
Destination	Host :	Adresse électronique du client
Descriptif	If-Modified-Since :	Accès à la ressource si modification depuis
	Referer :	URL du document ayant émis la requête
	User-Agent :	Information sur le navigateur
	Accept :	Types MIME supportés par le client
	Content-Encoding :	Type de codage du corps de la requête (compression)
	Content-Length :	Taille du corps de la requête
Fin-Entête	Content-Type :	Type MIME du corps de la requête
	...	
	CRLF	Ligne blanche
Corps	...	Surtout POST et PUT

```
GET /chemin/vers/fichier.html HTTP/1.0
Host: enseiht.fr:80
If-Modified-Since: Monday, 19-Jan-96 14:30:36 GMT
User-Agent: Mozilla/2.0b6a (X11; I; OSF1 V3.2 alpha)
```

14

Voici le format d'une requête HTTP.

C'est ce qui est envoyé en ASCII (chaîne de caractères) sur la connexion TCP (bien sur, le document peut contenir autre chose que de l'ASCII).

La première ligne contient la méthode (GET, POST ...), le path de la ressource adressée, et la version de HTTP utilisée.

Suivent un ensemble d'attributs. Notons les attributs Content-Length indiquant la taille du Corps pouvant contenir un document qu'on envoie ou bien des paramètres, et Content-Type qui indique le type MIME du Corps (par exemple une image JPG ou un document HTML).

Après ces attributs, il y a une ligne blanche, puis le Corps (document ou paramètres).

Les méthodes HTTP

GET	récupérer une ressource
POST	envoyer des données à une ressource
HEAD	obtenir des informations sur une ressource
OPTIONS , TRACE	interroger le serveur sur son état
CONNECT	tunneler un serveur (proxy)
PUT	ajouter ou remplacer une ressource
DELETE	supprimer une ressource

15

Les méthodes définies par HTTP sont données ici.

Les plus utilisées sont GET et POST.

GET permet de récupérer une ressource (par exemple une page HTML).

POST permet d'envoyer des données à une ressource (généralement un programme s'exécutant dans le serveur Web).

HEAD permet de récupérer des informations (attributs) d'une ressource, par exemple la dernière date de modification pour savoir si la version dans le cache client est toujours valide.

TRACE permet de tester la connexion.

OPTIONS permet de consulter des caractéristiques de communication avec le serveur Web.

CONNECT permet de créer un tunnel TCP sur un proxy.

PUT permet de déposer une ressource sur le serveur Web.

DELETE permet de détruire une ressource.

Toutes ces méthodes ne sont pas forcément implantées par un serveur Web ou les navigateurs (par exemple PUT et DELETE le sont rarement).

Réponse HTTP

Status	HTTP_version code phrase	
En-tête	Location :	URL exacte de la ressource demandée
Descriptif	Server :	Informations sur le serveur
	Content-Encoding :	Type de codage du corps de la réponse
	Content-Length :	Taille du corps de la réponse
	Content-Type :	Type MIME du corps de la réponse
	Date :	Date de la génération de la réponse
	Expires :	Date d'expiration du document
	Last-Modified :	Dernière modification du document
	...	
Fin-Entête	CRLF	Ligne blanche
Document	...	Contenu du document demandé

```
HTTP/1.1 200 OK
Date: Fri, 09 Jan 1998 09:49:11 GMT
Server: Apache/1.3b2
Last-Modified: Tue, 19 Aug 1997 11:57:17 GMT
Content-Length: 118
Content-Type: text/html
<html> ...
```

16

Voici le format d'une réponse HTTP.

La première ligne contient la version d'HTTP, puis un code de retour (par exemple 200 veut dire OK), puis un message qui peut être affiché (si problème).

Suivent un ensemble d'attributs. Notons Content-Length et Content-Type comme précédemment. Notons également Expires qui donne une indication sur le temps où on peut garder la ressource dans le cache du navigateur.

Après une ligne blanche, il y a le Document retourné, qui doit être affiché par le navigateur.

Code de retour



- Classe 1 : information (pas utilisé)
- Classe 2 : succès
 - ◆ 200 (OK), 201 (created), 204 (no content), . . .
- Classe 3 : redirection
 - ◆ 301 (moved permanently), 304 (not modified), . . .
- Classe 4 : erreur client
 - ◆ 400 (bad request), 401 (Unauthorized), 404 (not found), . . .
- Classe 5 : erreur serveur
 - ◆ 500 (internal serveur error), 501 (not implemented), 503 (service unavailable), . . .

17

Voici les codes de retour prévus par HTTP.

Le plus connu est 200 qui signifie que tout s'est bien passé et qu'on a récupéré la page Web qui peut être affichée par le navigateur.

404 est aussi très connue, c'est le code affiché quand on ne trouve pas une page.

Web statique

- Limitations
 - ◆ Requête limitée à la demande de ressources statiques
 - ◆ Besoin d'exécuter une application sur le serveur HTTP
 - ◆ Besoin de fournir des données
- Solution
 - ◆ Formulaires HTML pour la saisie
 - ◆ Scripts CGI pour l'exécution

18

Initialement, le Web était statique, ce qui veut dire qu'on ne pouvait que charger des pages HTML (ou des images, vidéos) existantes et stockées sur les disques des serveurs Web. Le web n'était alors qu'un simple graphe de documents, un document pouvant contenir un ensemble de liens (HTML) vers d'autres documents.

Le besoin s'est très vite fait sentir d'être en mesure d'exécuter des programmes dans les serveurs Web, ces programmes générant des pages Web. On dit alors qu'on a des serveurs Web dynamiques : dont les pages sont générées dynamiquement (par des programmes).

L'autre besoin qui s'est fait sentir très vite et de passer des paramètres à ces programmes, ces paramètres étant saisis et transmis par l'utilisateur.

Pensez à une application Web de réservation de billet d'avion. L'utilisateur doit pouvoir saisir des paramètres (ville de départ, d'arrivée, horaires) et le serveur web doit pouvoir générer la page Web montrant les vols disponibles.

Pour répondre à ces 2 besoins, on a ajouté :

- des formulaires dans HTML. Ces formulaires permettent une interaction avec l'utilisateur (qui peut saisir du texte ou choisir des options avec des boutons) et ces données sont envoyées comme paramètres dans la requête HTTP.

- des scripts CGI (Common Gateway Interface). Ce sont des programmes installés dans le serveur Web (comme des documents, mais exécutables).

Lorsque le serveur reçoit une requête adressant un tel programme, au lieu de retourner le document (le programme), le programme est exécuté et c'est le résultat de l'exécution du programme (son STDOUT) qui est retourné en réponse au client (navigateur).

Formulaires HTML

```
<form action="url" method="methode ">
```

```
...
```

```
</form>
```

- url : identifie le programme utilisé pour traiter le formulaire
- method : méthode à utiliser pour transmettre l'information au serveur
 - ◆ GET : données ajoutées à l'URL
 - ◆ POST : données envoyées dans le corps du message

19

Dans une page HTML, il est possible de définir des formulaires avec la balise `<form>`.

Dans cette balise, les attributs :

- action : donne l'URL du programme qui sera appelé (avec une requête HTTP) lorsque le formulaire est validé
- method : donne la méthode HTTP qui sera utilisée lorsque le formulaire est validé, pour appeler le serveur Web

Éléments de formulaire

■ Éléments INPUT

```
<input type="text" name="nom" size="size" maxlength="max" value="val" />
```

■ Différents types possibles

TEXT	Champ de saisie de texte
PASSWORD	Champ de saisie de texte caché
SUBMIT	Bouton de soumission du formulaire
RADIO	Bouton à cocher de type radio (1 unique)
CHECKBOX	Bouton à cocher (plusieurs)
HIDDEN	Champ invisible

First name:
Mickey
Last name:
Mouse
Submit

Male
 Female

- On envoie dans la requête un ensemble de paires nom=val

20

A l'intérieur du formulaire, on peut mettre des éléments graphiques interactifs permettant à l'utilisateur de saisir des choses.

Un premier exemple est la balise `<input>` qui permet de placer dans le formulaire un élément de saisie. L'attribut `name` définit le nom de la variable qui sera envoyée au serveur Web lorsque le formulaire sera validé. Dans l'exemple donné, un `<input>` de type "text" est une zone de saisie d'une chaîne de caractères et lorsque le formulaire est validé après avoir tapé "bonjour", un paramètre sera passé à la requête HTTP : `nom=bonjour`

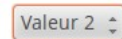
Les différents types de `<input>` sont :

- `type="text"` : on a une zone de saisie d'une chaîne de caractères
- `type="password"` : on a une zone de saisie d'un mot de passe (caché lors de la frappe)
- `type="submit"` : on a un bouton de validation du formulaire. Quand on clique dessus, une requête HTTP est envoyée au serveur Web (à l'URL définie dans la balise `<form>`) en passant les paramètres de tous les éléments dans le formulaire.
- `type="radio"` : chaque `<input type="radio" ...>` correspond à une option et l'utilisateur doit en choisir une.
- `type="checkbox"` : chaque `<input type="checkbox" ...>` correspond à une option et l'utilisateur peut en choisir plusieurs.
- `type="hidden"` : n'affiche rien et envoie une valeur cablée. Avec `name="x"` et `value="y"`, ça envoie `x=y` en paramètre de la requête HTTP.

Éléments de formulaire

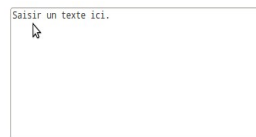
■ Éléments SELECT

```
<select name="choice">  
  <option value="value1">Valeur 1</option>  
  <option value="value2" selected>Valeur 2</option>  
  <option value="value3">Valeur 3</option>  
</select>
```



■ Éléments TEXTAREA

```
<textarea name="saisie" rows="10" cols="50">  
Saisir un texte ici.  
</textarea>
```



21

Dans un formulaire, on peut aussi mettre un élément `<select>`.

On a alors dans le formulaire un menu déroulant contenant ici les valeurs Valeur 1, Valeur 2, Valeur 3 et lors de la validation du formulaire, si on a choisi Valeur 1, ça envoie en paramètre de la requête HTTP : `choice=value1`

On peut aussi mettre dans le formulaire un élément `<textarea>` permettant de saisir un texte composé de plusieurs lignes.

Il y a beaucoup d'autres éléments. Consultez la doc.

Exemple

```
<html>
  <head><title>MaBanque</title></head>
  <body>
    <form method="post" action="/servlet/BanqueAccount">
      <p>numero de compte<input type="text" name="num">
        </p>
      <p>montant<input type="text" name="val"></p>
      <p><input type="submit" name="operation"
        value="solde">
        <input type="submit" name="operation" value="debit">
        <input type="submit" name="operation" value="credit">
        </p>
    </form>
  </body>
</html>
```

22

Voici un exemple complet.

Lorsque le formulaire est validé, ça envoie une requête HTTP POST à l'URL
/servlet/BanqueAccount

Cette URL est relative (ne commence pas par http://) donc c'est un path sur le même serveur Web que celui d'où vient la page courante.

Cette URL (/servlet/BanqueAccount) correspond sur le serveur Web à un programme écrit en Java (une servlet). Les servlets sont abordées plus tard dans le cours.

Un premier élément dans le formulaire permet de saisir le paramètre num.

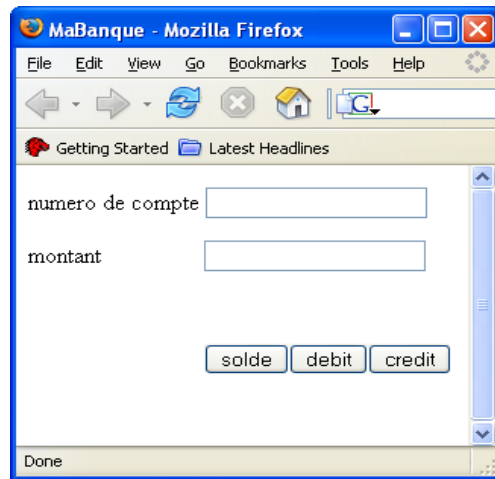
Un second élément dans le formulaire permet de saisir le paramètre val.

Les trois éléments suivants dans le formulaire sont de type submit. Si on clique sur l'un d'eux, le formulaire est validé et la requête HTTP est envoyée avec les paramètres saisis. Notons qu'on profite ici des boutons submit pour passer l'opération choisie (operation=solde, operation=debit, ou operation=credit).

Souvent on définit des boutons submit sans l'attribut name :

<input type="submit" value="OK"> on ne passe rien, on veut juste que l'utilisateur puisse cliquer sur un bouton OK pour valider le formulaire, les paramètres étant saisis par d'autres éléments dans le formulaire.

Exemple - résultat



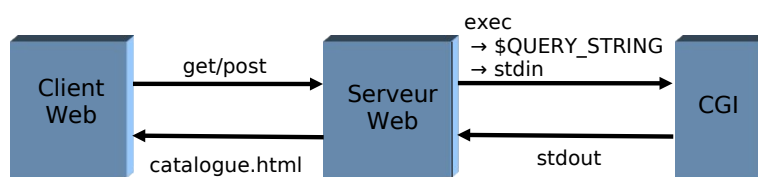
23

Voici ce qui est affiché dans le navigateur avec le formulaire précédent.

Vous pouvez vous amuser à copier le code HTML du slide précédent et à le sauver dans un fichier, puis à ouvrir ce fichier avec firefox.

Scripts CGI

- Programme générant un contenu en réponse à une requête
- Programmé dans n'importe quel langage
 - ◆ Perl, C, C++, Java, shell, ...
- Placé dans un répertoire particulier du serveur Web (cgi-bin)
- Envoie le contenu sur STDOUT
- Doit envoyer entête (type mime) + contenu
 - ◆ Pour HTML: content-type : text/html



24

On a vu que les formulaires HTML permettent de saisir des paramètres qui sont envoyés dans la requête HTTP envoyée au serveur Web.

Cela n'a d'intérêt que si on a un programme dans le serveur Web qui récupère ces paramètres et fait un traitement en fonction de ces paramètres.

La première forme de tel programme a été les CGI. C'est un standard qui spécifie l'interface entre le serveur Web et le programme.

En général, un répertoire (cgi-bin) est dédié à ces programmes et quand une requête HTTP référence quelque chose dans ce répertoire, on considère que c'est un binaire et il est exécuté. Ce programme est sensé générer le contenu (la page Web à retourner) sur la sortie standard (STDOUT). Le contenu doit être précédé de la définition du type (MIME) du contenu retourné (+ une ligne blanche avant le contenu), par exemple :

```
-----  
content-type : text/html
```

```
<html> ..... </html>
```

```
-----
```

Sur la figure, un formulaire qui était dans le client Web (navigateur) a été validé et une requête HTTP (get ou post) a été envoyée, référençant un CGI sur le serveur. A réception, le serveur appelle le CGI en lui passant les paramètres reçus dans la requête HTTP (nous allons voir les 2 modes de passage de paramètre plus loin). Le CGI génère une page HTML sur STDOUT et la page est retournée au navigateur.

Requête à un CGI

- Envoi de paramètres
 - ◆ champ1=valeur1&champ2=valeur2...
- Les requêtes GET et POST
 - ◆ GET : paramètres inclus dans l'URL
 - ▶ http://nom_du_serveur/cgi-bin/script.cgi?champ1=valeur1&...
 - ▶ Limitation à 255 caractères, visible, ...
 - ◆ POST : paramètres inclus dans le corps de la requête HTTP
 - ◆ Utilisation de formulaires (pour interactions)
- Réception des paramètres
 - ◆ GET : variable d'environnement QUERY_STRING
 - ◆ POST : STDIN, et variable d'environnement CONTENT_LENGTH

25

Lorsqu'un formulaire est validé (en cliquant sur un bouton submit), les paramètres des différents éléments du formulaire sont rassemblés. Chaque paramètre a un nom et une valeur (que ce soit un élément <input>, <select> ou autre).

Ces paramètres sont rassemblés en une chaîne de caractères
champ1=valeur1&champ2=valeur2

Lorsque la méthode HTTP GET est utilisée, les paramètres (la chaîne de caractères ci-dessus) sont copiés dans (à la fin de) l'URL. La conséquence est qu'on voit ces paramètres dans la barre du navigateur, et on est limité à la taille d'une URL (255 caractères). Comme l'URL est présente dans l'entête de la requête HTTP, le serveur pourra récupérer les paramètres dans cette entête.

Lorsque la méthode HTTP POST est utilisée, les paramètres sont copiés dans le corps de la requête HTTP.

Du côté du serveur Web, si GET est utilisée, le serveur reçoit la chaîne de caractères des paramètres dans la variable d'environnement QUERY-STRING (variable du processus). Si POST est utilisée, alors la chaîne de caractères des paramètres est reçue sur STDIN et la variable d'environnement CONTENT_LENGTH donne la taille de la chaîne de caractères.

Exemple - serveur CGI

```
#!/bin/bash
function extract_parameter()
{
  echo you submitted the following key-value pairs "<br>"
  str=$1
  while [ "$str" != "" ];
  do
    echo - `echo $str | cut -f1 -d'&'` "<br>"
    str=`echo $str | cut -s -f2- -d'&'`
  done
}
```

GET

```
echo "Content-Type: text/html"
echo ""
extract_parameter $QUERY_STRING
echo ""
```

Variable
d'environnement

POST

```
echo "Content-Type: text/html"
echo ""
read LINE
extract_parameter $LINE
echo ""
```

Lecture
STDIN

26

Dans ces petits shell scripts, la fonction `extract_parameter()` reçoit la chaîne de caractères des paramètres dans `$1`, puis découpe les paramètres et les affiche.

Si on veut faire un script CGI dans un serveur Web, qui affiche les paramètres reçus :

- à gauche avec la méthode GET, on appelle `extract_parameter()` en lui passant la chaîne reçue dans `QUERY_STRING`

- à droite avec la méthode POST, on lit une ligne sur `STDIN` et on la passe à `extract_parameter()`

Exemple GET - client

```
prompt>> telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /cgi-bin/essai.cgi?nom=toto&prenom=titi HTTP/1.0
Host: localhost

HTTP/1.1 200 OK
Date: Thu, 30 Sep 2010 11:10:01 GMT
Server: Apache/2.2.16 (Debian)
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

you submitted the following key-value pairs <br>
- nom=toto <br>
- prenom=titi <br>

Connection closed by foreign host.
```

27

En ayant placé ce script CGI (test-cours) dans le répertoire cgi-bin d'un serveur Web Apache, on peut se connecter au serveur Web avec telnet (outil permettant de créer une connexion TCP) sur le port 80, port par défaut d'un serveur Web.

Telnet permet de taper du texte qui est envoyé sur la connexion TCP.

On tape une requête HTTP simple :

```
GET /cgi-bin/test-cours?nom=toto&prenom=titi HTTP/1.0
```

Et le résultat est retourné par le serveur Web et s'affiche.

Notez qu'on peut aussi taper dans un navigateur l'URL :

<http://localhost/cgi-bin/test-cours?nom=toto&prenom=titi>

(avec un navigateur, en tapant une URL, la méthode GET est utilisée)

Exemple POST - client

```
prompt>> telnet localhost 80
Connected to localhost.
Escape character is '^]'.
POST /cgi-bin/essai.cgi HTTP/1.0
Host: localhost
Content-length: 20

nom=toto&prenom=titi
HTTP/1.1 200 OK
Date: Thu, 30 Sep 2010 11:17:41 GMT
Server: Apache/2.2.16 (Debian)
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

you submitted the following key-value pairs <br>
- nom=toto <br>
- prenom=titi <br>

Connection closed by foreign host.
```

28

Pour tester avec POST, on peut le faire avec telnet de la même façon, mais on voit cette fois-ci que les paramètres sont passés dans le corps de la requête HTTP.

Si on veut tester avec un navigateur, il faut faire une page HTML avec un formulaire, par exemple :

```
<html>
  <body>
    <form method="post" action="/cgi-bin/test-cours">
      <input type="hidden" name="nom" value="toto">
      <input type="hidden" name="prenom" value="titi">
      <input type="submit" value="OK">
    </form>
  </body>
</html>
```

Et en cliquant sur le bouton submit (OK), ça envoie la requête HTTP POST.

Cookies HTTP

- Limitations d'HTTP
 - ◆ Protocole sans mémoire
 - ◆ Pas d'identification simple du client
- Cookies
 - ◆ Mécanisme de stockage d'informations chez le client (pris en compte par le serveur à chaque accès)
 - ◆ Exemples d'utilisation :
 - ▶ sauvegarde d'option
 - ▶ validité d'accès à un serveur payant

29

Une des grandes limitations d'HTTP est que c'est un protocole sans état ou sans mémoire. Toutes les requêtes reçues sont anonymes.

Cela est gênant si dans mon serveur Web, je veux reconnaître le client qui m'envoie une requête (l'identifier). Cela est très utile dans un serveur de commerce électronique, par exemple pour gérer le caddy d'un client qui fait ses achats. Quand un client fait ses courses sur un site marchand et accumule des articles dans son caddy, à la réception d'une requête (correspondant à la sélection d'un article pour achat), il faut être capable d'identifier le client pour pouvoir ajouter l'article dans son caddy et lui retourner une page affichant l'état de son caddy.

Les cookies répondent à ce besoin. Il s'agit d'un mécanisme permettant au programme (un CGI) s'exécutant sur le serveur Web de retourner une information (un cookie) au navigateur. Le navigateur stocke les cookies qu'il reçoit des serveurs Web. Et lorsque le navigateur contacte un serveur Web, il lui renvoie les cookies qu'il a reçu de ce serveur Web.

Dans l'exemple du caddy, lorsqu'un caddy a été créé pour un utilisateur, on peut allouer un numéro identifiant ce caddy et ce numéro est retourné comme cookie au client. Ensuite, chaque requête provenant d'un client contiendra le cookie avec l'identification du caddy de l'utilisateur.

Ce mécanisme peut également servir pour mémoriser les préférences (options préférées) de l'utilisateur ou pour vérifier que l'utilisateur possède des droits d'accès valides au service.

Cookies : création par le serveur

- Dans l'en-tête HTTP

- ◆ Set-Cookie : Nom=Valeur ; expires=Date ; path=Chemin ; domain=NomDomaine ; secure

name	associe une valeur à une variable spécifique (obligatoire)
expires	date d'échéance du cookie
domain	le serveur qui a envoyé le cookie (vide par défaut : affecté par le client avec site courant)
path	association à un ensemble de ressources du serveur
secure	nécessite une connexion sécurisée (https)

30

Lors de la création d'un cookie par le serveur Web, ce cookie est ajouté dans l'en-tête HTTP, dans laquelle on aura une ligne :

Set-Cookie : Nom=Valeur

Le cookie est une paire key=valeur.

On peut spécifier une durée de vie au cookie (expires) ou un path (le cookie n'est envoyé que quand on accède une ressource correspondant à ce chemin).

Cookies : gestion par le client

- A réception d'une réponse
 - ◆ Mémoire les cookies qu'il reçoit
- A émission d'une requête
 - ◆ Recherche parmi les cookies mémorisés ceux s'appliquant au couple domain/path
 - ◆ Ajoute dans la requête une ligne par paire nom/valeur
- A noter
 - ◆ Plusieurs directives Set-cookie insérables par le serveur
 - ◆ Nombre et taille des cookies limités
 - ◆ Effacement d'un cookie possible en précisant pour expires une date de péremption
 - ◆ Un cookie sans date expire à la mort du navigateur

31

A réception d'un cookie, le navigateur mémorise les cookies.

Et lors de l'envoi d'une requête HTTP, les cookies correspondant au domain/path sont ajoutés dans l'entête HTTP.

A noter que :

- le serveur ou le client peuvent envoyer plusieurs cookies
- le nombre et la taille des cookies sont limités

Exemple

- Requête - Client

GET /chemin/vers/fichier.html HTTP/1.0

- Réponse - Serveur

HTTP/1.1 200 OK

Date: Tue, 07 Aug 2001 21:36:13 GMT

Server: Apache-AdvancedExtranetServer/1.3.19

Set-Cookie: id=0xa11111; expires=Friday, 10-March-09 00:00:00 GMT; path=/

Content-Type: text/html

<HTML>

...

- Nouvelle requête - Client

GET /chemin/vers/fichier.html HTTP/1.0

Cookie: id=0xa11111;

32

Voici un exemple d'exécution :

- le client fait une requête HTTP au serveur (pas de cookies)
- le serveur répond en retournant un cookie : id=0xa11111
- plus tard, lors d'une nouvelle requête HTTP du client, le cookie est ajouté dans la requête.

Premier bilan

- Initialement
 - ◆ Web statique
 - ◆ Principalement des serveurs
 - ▶ de page HTML
 - ▶ accédées par HTTP
- Evolution vers des sites dynamiques
 - ◆ Formulaires HTML pour saisie
 - ◆ Scripts CGI pour génération dynamique de pages
 - ◆ Souvent il faut accéder à une BD
- Programmation fastidieuse

33

Un premier bilan.

Le Web était initialement statique, c'est à dire qu'il permettait seulement une navigation avec des hyperliens entre des pages HTML prédéfinies et stockées sur les disques durs des machines hébergeant les serveurs Web.

Très vite, le besoin de faire du Web dynamique s'est fait sentir, et la réponse à ce besoin a été de fournir :

- la possibilité d'interaction (saisie de données) avec l'utilisateur avec des formulaires HTML
- la possibilité d'exécuter des programmes (recevant les paramètres provenant des formulaires) sur les serveurs Web, avec les scripts CGI.

Notons que souvent ces programmes doivent accéder à des données gérées dans des bases de données. Au début, on faisait cela en shell ou en C avec des bibliothèques permettant d'envoyer des requêtes SQL à des bases de données relationnelles.

Tous cela est très bien, mais la programmation de ces applications Web reste fastidieuse. La motivation pour la suite a été de fournir des environnements de plus en plus évolués pour faciliter le développement de ces applications.