Typage des ressources Internet

- Types MIME (*Multi-purpose Internet Mail Extension*)
  - RFC 2046. Composé par un type et un sous-type
  - Les types principaux sont les suivants
    - text
    - image
    - audio
    - video
    - message
    - multipart
    - application
  - Ex: text/html, image/gif, application/octet-stream, etc.

Types MIME (2)

- Possibilité de spécifier un paramètre du sous-type
  - Après le sous-type et un point-virgule
  - Par exemple, codage des caractères pour le type texte
    - text/html; charset=iso-8859-1
- Pour les documents composites contenant différents types MIME
  - **multipart** avec **boundary** pour délimiter les parties
    - Sous-types:
    - mixed (par défaut),
    - alternative (la même information dans différents formats)
- La syntaxe des types MIME est ouverte

Les URI

- ***Uniform Resource Identifier*** (RFC 2396)
  - URL (Locator): localisation sur le réseau
  - URN (Name): indépendant de la localisation (site mirroir)
- URI identifie une ressource à partir d'un **schéma**
  - http, ftp, mailto, etc.
  - Attention: tous les URI ne sont pas nécessairement associés à des protocoles de communication
    - Exemple: URN:ISBN:2-7117-8689-7
  - Syntaxe générale:
    - **[<schéma>:]<partie dépendante du schéma>[#fragment]**

Syntaxe dans les URI

- Le schéma
  - doit commencer par une lettre minuscule, puis être composé de lettres minuscules, de chiffres et des signes « + », « . » et « - »
- La partie dépendante du schéma
  - Composée de caractères ASCII ou de caractères codés en hexadécimal de la forme « **%xx** »
  - Certains caractères peuvent avoir une interprétation spécifique (ils sont en général traduits automatiquement, si besoin, par les navigateurs. Il s'agit de l'espace et de
    " # < > ` | % { } \ ^ [ ] + ; / ? : @ = &

URI hierarchiques

- Certains URI sont dits « **hierarchiques** » (par opposition à « **opaques** »): leur partie dépendante du schéma peut être décomposée comme suit:
- **[<schéma>:][//<authority>][<path>][?<query>][#<fragment>]**
  - **<authority>**: autorité responsable du nommage de la ressource
  - **<path>**: chemin d'accès à la ressource: segments séparés par des « **/** » et suivis d'un ou plusieurs paramètres séparés par des « **;** »
  - **<query>**: chaîne de caractères interprétée par les deux champs précédents
  - Si l'autorité est liée à un serveur, elle a la forme suivante: **[<user-info>@]<machine>[:<port>]**
  - Si port n'est pas spécifié, port par défaut associé au schéma

Schémas http et https

- Schéma **http**
  - http://<machine>:<port>/<path>?<query>
  - Port par défaut **80**
  - <path> peut être interprété comme un chemin d'accès dans un système de fichiers, mais pas forcément
  - <query> informations supplémentaires (paramètres pour l'exécution d'une commande, par exemple)
- Schéma **https**
  - Protocole HTTP au dessus du protocole SSL
  - Même format, mais port par défaut **443**

Schéma ftp

- Transférer un fichier ou lister un répertoire
  - **ftp://<user>:<passwd>@<machine>:<port>/<r1>/<r2>/...**
    .../name;type=<typecode>
  - Si ni **<user>** ni **<passwd>** ne sont spécifiés, équivaut à ftp anonyme avec l'adresse e-mail en passwd
  - Port par défaut **21**
  - Suite **<r1>/<r2>/.../name** correspond au chemin d'accès relatif au répertoire de **<user>**.
  - **;type=<typecode>** est optionnel (typecode vaut « d » pour lister un répertoire, « i » pour binaire, « a » pour ASCII)
  - Exemples : **ftp://ftp.inria.fr/rfc**
    ftp://duris:monmo2pas@igm.univ-mlv.fr/W3/img.gif;type=i

Schéma mailto

- Désigne les boîtes aux lettres, courrier électronique
  - RFC 822
  - Historiquement URL, mais qqfois considéré comme URN
  - **mailto:<adr1>,<adr2>?<header1>=<value1>&<header2>=<value2>**
  - **<adr i>** sont les adresses destinataires du courrier
  - **<header i>=<value i>** sont des en-têtes à placer dans le courrier. Si **<header>** vaut **body**, c'est le corps du message
  - Seuls les en-têtes **body** et **subject** doivent être reconnus
  - Exemples:
    mailto:roussel@univ-mlv.fr
    mailto:roussel@univ-mlv.fr,bedon@univ-mlv.fr?subject=test&body=test

Schéma telnet

- Terminal interactif
- **`telnet://<user>:<passwd>@<machine>:<port>/`**
- Port par défaut **23**
- Ouvre un terminal sur la machine spécifiée, avec comme nom d'utilisateur **`<user>`**, à condition que le mot de passe soit valide

Schéma file

- Accès à un système de fichier (en général local)
- **`file://<machine>/<path>`**
- On n'utilise pas, en général, la partie machine, mais le « **`/`** » du path doit, en principe, être présent
- **`file:///home/duris/COURS/JAVA/http/progJavaHTTP.sdd`**
- Même pour des fichiers Windows, les séparateurs sont des « / »
- **`file:///C:/Program%20Files/`**

Les URI relatifs

- Concerne les URI hierarchiques
  - Parler d'une ressource « relativement » à un URI de base. Forme la + complète d'un URI relatif:
  - **`//<net_loc>/<path1>/.../<pathn>;<params>?<query>`**
  - Chacun des préfixes peut être omis et sera remplacé par l'URI de base (s'il est moins précis)
  - **`http://igm.univ-mlv.fr/~roussel/test/index.html`**
  - **`toto.html`** signifie alors **`http://igm.univ-mlv.fr/~roussel/test/toto.html`**
  - **`/~duris/index.html`** signifie alors **`http://igm.univ-mlv.fr/~duris/index.html`**
  - **`//etudiant.univ-mlv.fr`** signifie alors **`http://etudiant.univ-mlv.fr/`**

Le protocole HTTP

- (*Hyper Text Transfert Protocol*) RFC 2616 (version 1.1)
- Protocole **sans état**, **rapide**, **léger**, de niveau application permettant le **transfert de données** au dessus d'une **connexion** de type TCP (sûr)
  - Utilise les URLs
- Port par défaut **80**
- Basé sur le paradigme de **requête / réponse**
  - Chaque requête et chaque réponse est un « **message HTTP** »
- Largement utilisé sur le World Wide Web

Principe Requête/Réponse

- Connexion TCP établie entre un client et le port 80 (en général) du serveur HTTP
- Envoi par le client d'un message HTTP, dit *requête*
- Traitement de cette requête et envoi par le serveur d'un message, dit *réponse*, au client.
- En HTTP 0.9 et 1.0, la connexion est fermée par le serveur après chaque réponse
- En HTTP 1.1, il est possible de conserver la même connexion TCP pour plusieurs échanges de requêtes/réponses.

Format des messages HTTP

- **`Ligne de début`**
  `Champ d'en-tête_1: valeur_1`
  `...`
  `Champ d'en-tête_n: valeur_n`
  `¶` *// ligne vide « \r\n »*
  `[ corps éventuel ]`
  - La présence de zéro, un ou plusieurs champs d'en-tête permettent de spécifier des paramètres
  - La présence d'un corps de message dépend de la ligne de début de ce message

Message de requête

- Constitution de la première ligne de requête
  - ***`Méthode Ressource Version`***
    Exemple:
    ```
    GET /index.html HTTP/1.1
    Host: www.univ-mlv.fr
    ```
- ***Méthode***: `GET, HEAD, POST, PUT, TRACE,...`
  - Représente le type d'opération à effectuer
- ***Ressource***: identifie la ressource recherchée
  - HTTP/1.1 exige que le champ « *Host* » soit spécifié
- ***Version***: identifie la version du protocole utilisée
  - Si rien n'est spécifié, la version HTTP 0.9 est considérée

Message de réponse

- Constitution de la ligne dite « **de statut** »
  - ***`Version Code Message`*** Exemple:
    ```
    HTTP/1.1 200 OK
    Server: Netscape-Enterprise/4.1
    Date: Wed, 18 Sep 2002 09:30:24 GMT
    ...
    Content-type: text/html
    Content-length: 13298
    ¶
    <!DOCTYPE HTML PUBLIC ....
    <html><head>
    ...
    </html>
    ```

Les codes de statut

- Organisés en 5 catégories
  - **1xx « informatifs »**: acceptés uniquement dans les versions 1.1. Réponse « intermédiaire » du serveur.
    Ex: `100 Continue`
  - **2xx « de succès »**: la réception, la compréhension ou l'acceptation de la requête est réussie. Ex: `200 OK`
  - **3xx « de redirection »**: d'autres actions sont requises pour traiter la requête. Ex: `301 Moved Permanently`
  - **4xx « erreurs du client »**: erreur de syntaxe ou de spécification. Ex: `404 Not Found`
  - **5xx « erreurs du serveur »**: serveur incapable de répondre. Ex: `501 Method Not Implemented`

Quelques champs d'en-tête

- Spécifiques (requêtes/réponses) ou bien généraux:
- **Server** (réponse): informations sur le serveur HTTP
- **Date** (général): estampille les messages HTTP
- **Last-modified** (serveur): date de dernière modification
- **ETag** (): identifiant de la ressource géré par le serveur
- **Content-type** (général): type le corps du message
- **Content-length** (général): nombre d'octets du corps
- **Accept: text/plain** (requête): ce qu'accepte le client
- **Accept-charset**, **Accept-language**, **Accept-encoding**
  Content-encoding, **Transfert-encoding...**

Quelques requêtes

- `GET`: récupérer une ressource
- `HEAD`: identique à `GET`, mais sans corps de msg
  - Peut servir à savoir si une ressource existe et si elle à changé, sans nécessairement la récupérer
- `TRACE`: écho de la requête
  - le serveur renvoit, dans le corps du message, le message qu'il a reçu du client (diagnostique)
- `OPTIONS`: demande d'information sur les méthodes
  - Le serveur renvoit toutes les méthodes qu'il implante dans le champ d'en-tête « `Allow` »

Connexion persistante ou "keep alive"

- Avant 1.1, champ **`Content-length`** pas obligatoire:
  - Comme le serveur fermait la connexion à la fin de la réponse, ce champ servait simplement à vérifier que OK
- Avec la **connexion persistante** (***keep alive***)
  - Serveur peut garder la connexion après la fin de réponse
  - Plus efficace (délais d'ouverture et fermeture de connexion)
  - Possibilité d'exploiter le full-duplex de la connexion TCP (plusieurs transactions peuvent se recouvrir)
- Toutes les réponses doivent contenir **`Content-length`**
  (sauf réponses sans corps de message ou `chunked`)
  - Indispensable pour déterminer la fin d'un message

```
% telnet www.w3.org 80
Trying 193.51.208.67...
Connected to www.w3.org.
Escape character is '^]'

GET /Overview HTTP/1.1
Host: www.w3.org

HTTP/1.1 200 OK
Date: Sun, 30 Jun 2002 15:33:43 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.9
Last-Modified: Sat, 29 Jun 2002 16:08:59 GMT
ETag: "2d2c96-2c69-3778ef9b"
Accept-Ranges: bytes
Content-Length: 11369
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC ...
<html lang="en"> ... le code du document html ...
</html>
```

La connexion n'est pas fermée...


```
HEAD /Overview HTTP/1.1
Host: www.w3.org

HTTP/1.1 200 OK
Date: Sun, 30 Jun 2002 15:33:58 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.9
Last-Modified: Sat, 29 Jun 2002 16:08:59 GMT
ETag: "2d2c96-2c69-3778ef9b"
Accept-Ranges: bytes
Content-Length: 11369
Content-Type: text/html; charset=iso-8859-1

OPTIONS / HTTP/1.1
Host: www.w3.org
Connection: close

HTTP/1.1 200 OK
Date: Sun, 30 Jun 2002 15:34:17 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.9
Content-Length: 0
Allow: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, PATCH,
  PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, TRACE
Connection: close

Connection closed by foreign host.
%
```

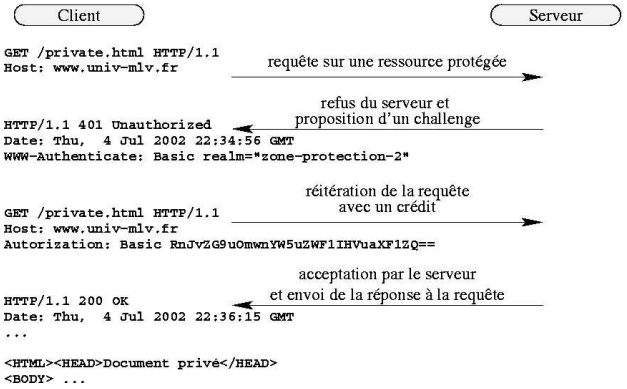

L'authentification

- Pour limiter l'accès de ressources à qq clients
  - Mécanisme de *challenge/crédit* pour l'authentification des requêtes
  - Utilise les champs **`WWW-Authenticate`** et **`Autorization`**
- Lors d'une requête sur une ressource protégée
  - Serveur répond **`401 Unauthorized`** avec, un champ **`WWW-Authenticate`** qui contient au moins un *challenge* applicable à la ressource concernée (une liste)
  - Un challenge est constitué d'un schéma (mot, ex: Basic), suivi de paires attribut=valeur. Exemple:
    `Basic realm= "zone-protection-2"`

L'authentification (suite)

- Le client doit alors réémettre sa requête avec un *crédit* dans le champ **`Autorization`**. Exemple:
  - **`Autorization: Basic blablabla`**
    où **`blablabla`** représente le codage en base 64 de la chaîne « login:passwd »
  - Attention, base 64 est réversible, et le crédit passe en clair
- Autre schéma: ***Digest***
  - Requiert une valeur d'autorisation obtenue par hachage cryptographique utilisant, en plus de user et passwd de client, des informations liées à la requête et un valeur aléatoire tirée par le serveur

Authentification (exemple)

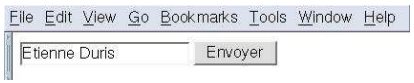


Envoyer des données

- Une requête peut servir à envoyer des informations vers le serveur HTTP
  - Méthode POST, dans le corps du message, mais aussi
  - Méthode GET, dans l'URL sous la forme de « *query string* »: permet de transmettre quelques arguments
- Ces données sont principalement utilisées pour « générer » une ressource dynamiquement
  - Utilise des « programmes » satellites au serveur
  - CGI, Servlettes/JSP, PHP, ASP...

Les formulaires HTML (GET)

- Permet de définir des zones de saisie pour le client
  - <form enctype="application/x-www-form-urlencoded"
    action="http://www.serv.fr/path"
    method="**GET**">
    <input type="text" name="user" value="votre nom">
    <input type="submit" name="ok" value="Envoyer">
    </form>



  - Un click sur Envoyer crée une requête GET à l'URL suivant: http://www.serv.fr/path?user=Etienne+Duris

Les formulaires HTML (POST)

- <form action="http://www.serv.fr/path" method="**POST**">
  <input type="text" name="user" value="votre nom">
  <input type="submit" name="ok" value="Envoyer">
  </form>
- Même apparence dans un navigateur
- Un click sur Envoyer crée une requête POST
  - à l'URL http://www.serv.fr/path,
  - Avec comme champs d'en-tête
    Host: www.serv.fr
    Content-Type: application/x-www-form-urlencoded
    Content-Length: 14
  - dont le **corps du message** contient
    user=votre+nom

Formulaire plus riche (illustration)


```
<form enctype="application/x-www-form-urlencoded"
  action="http://www.serv.fr/path" method="GET">
  Zone de texte (nom) :
  <input type="text" size="30" name="user" value="votre nom"><hr>
  <textarea name="srctext" rows="6" cols="40">
   Tapez votre message ici.
  </textarea><hr>
  Zone de boutons (age) : Quel âge avez vous?
  moins de 18 ans <input type="radio" name="age" value="-18">
  de 19 à 35 ans <input type="radio" name="age" value="19-35">
  de 36 à 50 ans <input type="radio" name="age" value="36-50" checked>
  plus de 50 ans<input type="radio" name="age" value="50-"><hr>
  Zone de menu (disponibilité) : Vous êtes
  <select name="dispo">
   <option value="ok">disponible
   <option selected value="ko">indisponible
  </select><hr>
  <input type="hidden" name="client" value="a1234-567-8901">
  <input type="submit" name="choix1" value="Accepter">
  <input type="submit" name="choix2" value="Refuser">
  <input type="reset" value="Réinitialiser">
  <hr>
</form>
```


Corps de message « par morceaux »

- Lorsqu'une requête génère une réponse longue
  - On devrait attendre toute la génération de la réponse afin de connaître sa taille AVANT de créer l'en-tête de la réponse et son champ **`Content-Length`**
  - On peut alors « découper » la réponse en morceaux (***chunked***) et commencer à envoyer les premiers avant d'avoir fini de générer la totalité du document
  - Utilise le champ **`Transfert-Encoding: chunked`**
  - Il n'y a plus de champ d'en-tête **`Content-length`**
  - Chaque morceau est préfixé par sa taille
  - Le premier morceau de taille nul indique la fin du corps

Chunked (exemple)

```
GET /search?q=test HTTP/1.1
Host: www.google.fr

HTTP/1.1 200 OK
Server: GWS/2.0
Date: Mon, 23 Sep 2002 11:58:56 GMT
Transfer-Encoding: chunked
Content-Type: text/html

b3f
<html><head>
..... Premier morceau ...
3ce2
..... Deuxième morceau ...
</body></html>
0
```

Envoyer des fichiers

- Si le formulaire spécifie un nom de fichier, c'est le contenu du fichier qu'on désire envoyer
  - le type (enctype) ne doit alors plus être "**application/x-www-form-urlencoded**", mais "**multipart/form-data**", introduit en HTML 4 pour cela

Envoyer des fichiers (code)

```
<form enctype="multipart/form-data"
  action="/sendfile" method="POST">
  Fichier 1 : <input type="file"
              name="fichier1" size="20"><br><br>
  Fichier 2 : <input type="file"
              name="fichier2" size="20"><br><br>
  <input type="hidden" name="ID" value="roussel">
  <input type="submit" name="submit" value="Envoyer">
</form>
```

- Supposons que le fichier contienne le texte suivant:
  - Test de fichier
- Alors la requête générée par un click sur Envoyer sera la suivante...


```
POST /sendfile HTTP/1.1
Host: server
Content-Type: multipart/form-data; boundary=ABCDEF
Content-Length: 372

--ABCDEF
Content-Disposition: form-data; name="fichier1"; filename="test.txt"
Content-Type: text/plain

Test de fichier

--ABCDEF
Content-Disposition: form-data; name="fichier2"; filename=""
Content-Type: application/octet-stream


--ABCDEF
Content-Disposition: form-data; name="ID"

roussel
--ABCDEF
Content-Disposition: form-data; name="submit"

Envoyer
--ABCDEF--
```


Requête GET ou POST

- Les deux méthodes sont utilisables dans un formulaire
- Différences principales
  - POST transmet les données dans le corps
  - GET transmet les données dans l'URL
  - POST + Ésthétique (arguments n'apparaissent pas)
  - POST non limité en taille (les serveurs limitent une taille maximale pour les URLs). Dans ce cas, code de statut `414 Requested-URI Too Large`
  - Un formulaire GET peut être « rejoué » en conservant l'URL (bookmark du navigateur)

D'autres requêtes

- PUT permet d'envoyer au serveur HTTP des données (fichiers) qu'il devra stocker
  - Différent de POST, dont les données sont destinées à une ressource identifiée par l'URL (qui doit normalement interpréter les données)
  - Dans le cas de PUT, les données **sont** la ressource à placer à l'endroit spécifié par l'URL
  - Mise à jour (`200 OK`) ou création (`201 Created`)
- DELETE permet de supprimer des données
- Requierent le droit d'accès (authentification)

Les relais

- Modèle client/serveur trop simpliste
  - Beaucoup de machines intermédiaires qui jouent le rôle de serveur pour le client et de client pour le serveur
  - Peuvent également stocker temporairement des réponses pour les « rejouer » à la prochaine requête identique: ***proxy cache***
  - Un client peut spécifier explicitement (statiquement) qu'il veut utiliser un proxy
  - Nombreux champs d'en-tête spécifiques
  - `Proxy-Authenticate, Proxy-Authorization, Via, Max-Forwards, Cache-Control (Pragma: no-cache), Vary...`

Session Tracking

- HTTP protocole sans état. Comment savoir quelle était la précédente requête du même client?
- Idée: faire « sauvegarder un état » par le client, qu'il devra renvoyer dans la prochaine requête
  - Champs « `hidden` » des formulaires
  - **Réécriture des URL** des liens hypertextes conten us dans les documents retournés. Exemple:
    `<a href="http://serveur/request?user=duris">`
    Ne requiert aucune participation du client
  - **Cookies** (plus simple pour le serveur)
    introduit par Netscape, puis RFC 2109
    Requiert la participation du client

Cookies

- C'est une paire <clé>=<valeur>
  - Initialement fournie par le serveur dans une réponse
  - Stockée par le client (navigateur)
  - Puis retransmise par le client lorsqu'il émet des requêtes
- Spécifié par un champ d'en-tête dans la réponse
  - `Set-Cookie: <clé>=<valeur> [; expires=<DATE>]`
    `[; path=<PATH>]`
    `[; domain=<DOM>]`
    `[; secure]`
- Rejoué par les clients:
  - `Cookie: <clé1>=<valeur1>; <clé2>=<valeur2> ...`

Cookies (suite)

- Attributs optionnels
  - **expires**: pour la date de péremption. Au delà, le client ne doit pas conserver le cookie
  - **domain** et **path** servent à déterminer les URL pour lesquels ce cookie doit être utilisé
    (par défaut, ceux de l'URL de la requête sont utilisés)
    - Les cookies dont la valeur de domain est différente du domaine du serveur ayant généré la réponse doivent être ignorés
    - Les cookies trop larges (moins de deux « . ») ignorés
  - **secure**: cookie à n'utiliser que si connexion sécurisée
  - Si même path et même domain, le nouveau écrase l'ancien. Si path diffère, on stocke et renvoit les deux.