Le Web, et le protole HTTP

CSC4101 - Séance 1

Télécom SudParis

15/09/2016

Table des matières

1 Préface

1.1 Rappel séance précédente

— Présentation des objectifs du cours
— Panorama général des concepts
— Historique
— Enjeux
— Modalités
— Exercice : votre consommation du Web en 24h ?

1.2 Objectifs de cette séance

— Comprendre le protocole HTTP
— Structure des URLs
— Bases communication Client - Serveur
— Contenu des messages de requêtes / réponses
— Expérimenter avec les outils

1.3 Résultat exercice hors présentiel

Alors, une diette numérique s'impose-t-elle ?

2 Historique

2.1 Avant le Web

— Minitel

— Systèmes *hypertextes* locaux
— FTP, Usenet, Gopher

> Gopher fournit un système hypertexte.
> Gopher est à peu près contemporain de WWW, mais semble avoir subi la concurrence du Web, du fait d'une tentative de gestion de la propriété intellectuelle associée par l'université d'origine, mais aussi plus probablement du fait de l'ajout du support des images dans WWW, alors que Gopher était essentiellement textuel.

2.2 Hypertexte

— *Memex* de Vannevar Bush, dans *As We May Think* (*Atlantic Monthly*, 1945)
— *Xanadu* de Ted Nelson : `https://fr.wikipedia.org/wiki/Projet_Xanadu` (1965-...)
— *HyperCard* de Bill Atkinson (Apple, 1987)
— *World Wide Web* de Tim Berners-Lee (CERN, 1989)
Plus de détails dans `https://fr.wikipedia.org/wiki/Hypertexte`

2.3 World Wide Web

— Le Web est né au CERN en 1989-1990.
— Tim Berners-Lee a défini une architecture pour accéder à des documents liés entre eux et situés sur des serveurs reliés par Internet (*Web* = toile d'araignée)
— le W3C (*World Wide Web Consortium*) mis en place rapidement (1994) pour définir des standards (ouverts).

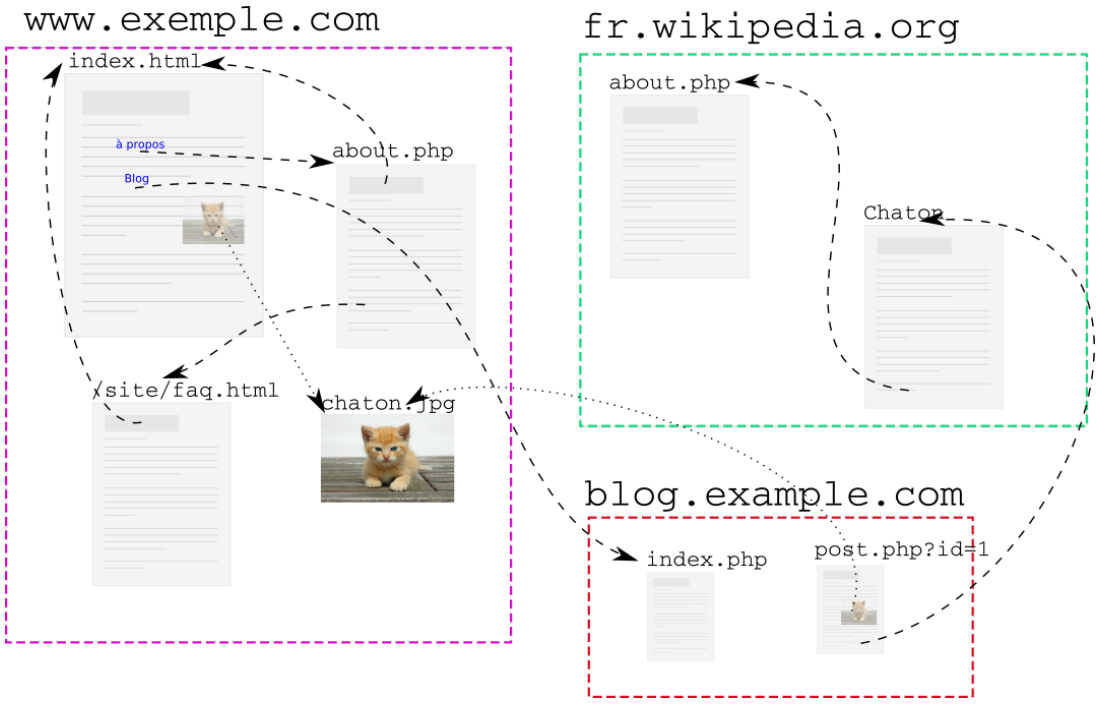

"Tim Berners-Lee"

Objectif : répondre à leur besoin d'échanges de documents (rapports, croquis, photos...) entre des équipes internationales.
Note : l'IMT est membre du W3C.
Important : ouverture des protocoles, standards ouverts.

2.3.1 Premier site

`http://info.cern.ch/hypertext/WWW/FAQ/Bootstrap.html`

2.4 Graphe de ressources liées décentralisé



Liens entre différents documents, du même site ou de sites différents.
Certains liens sont navigables, d'autres correspondent à l'inclusion de ressources externes (images).
Note : les liens sont uni-directionnels, du point de vue des serveurs : les documents ne savent pas qui leur pointe dessus. Aucune coordination : pas de permission à demander avant de tisser des liens, mais pas de garantie de disponibilité des ressources distantes.

2.5 Grandes étapes de l'évolution du Web

2.6 1. Naissance du Web

(début des années 1990)
— Accès à des documents structurés via des liens hypertextes
— Protocoles et langages simples
— Technologies de base HTML, HTTP, MIME, formats GIF...

2.7 2. Ouverture, homogénéisation et programmation

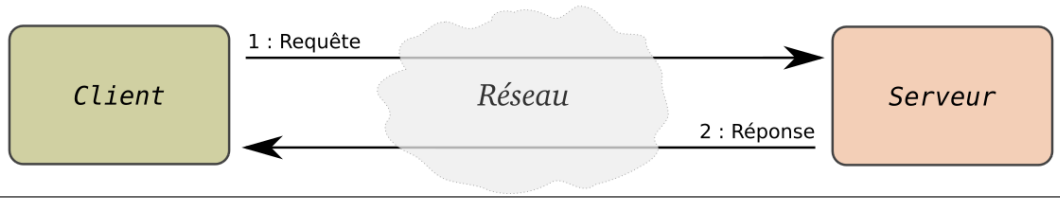
(fin des années 1990)
— Interactions avec les applications et programmation Web
— Langages plus riches, manipulation d'objets, développement des styles
— Evolution des technologies : XML, CSS, DOM, Server Pages, JavaScript ...
— Standardisation difficile (guerre des navigateurs)

2.8 3. Evolution des usages et de l'interface utilisateur

(depuis 2005)
— Partage d'informations, édition collaborative, sites communautaires
— Réseaux sociaux, mondes virtuels
— Technologie AJAX, HTML 5
— Intégration de flux RSS, de vidéos, de podcasts
— Personnalisation des accès
— *User-generated* content

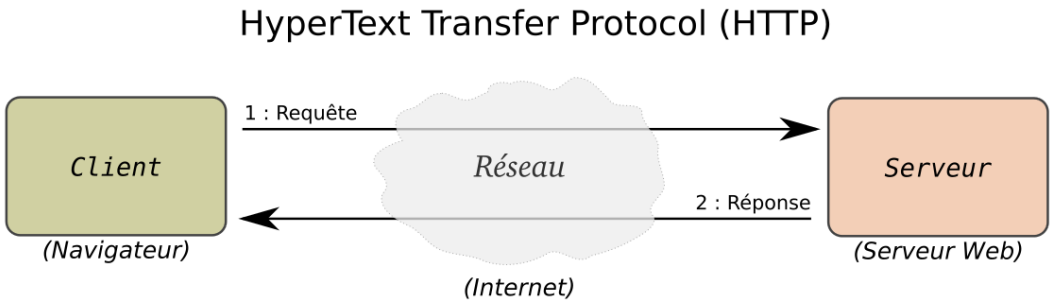
3 Clients et serveurs HTTP

3.1 Architecture Client-Serveur



Cette architecture est très classique et n'est pas née avec le Web.

3.2 Client et serveur Web



Client aussi appelé *user agent* dans les spécs

Serveur aussi appelé *origin server* dans les spécs

Le client peut être tout type de programme (navigateur, robot, etc.).
La notion d'*origin server* (serveur d'origine) permet de distinguer le serveur d'un éventuel *proxy* (serveurs mandataire) présent entre ce serveur et le client. Les détails d'HTTP concernant les *proxies* ne seront pas abordés dans ce cours.

3.3 Dialogue entre client et serveur

— Communication en 3 étapes simples :

1. Le **client** (navigateur) fait une **requête** d'accès à une **ressource** auprès d'un serveur Web selon le protocole **HTTP**
2. Le **serveur** vérifie la demande, les autorisations et transmet éventuellement l'information demandée
3. Le client interprète la **réponse** reçue et l'affiche à l'utilisateur (directement ou via un plug-in ou une application).

— Le client itère pour récupérer éventuellement des ressource complémentaires (images, . . .).

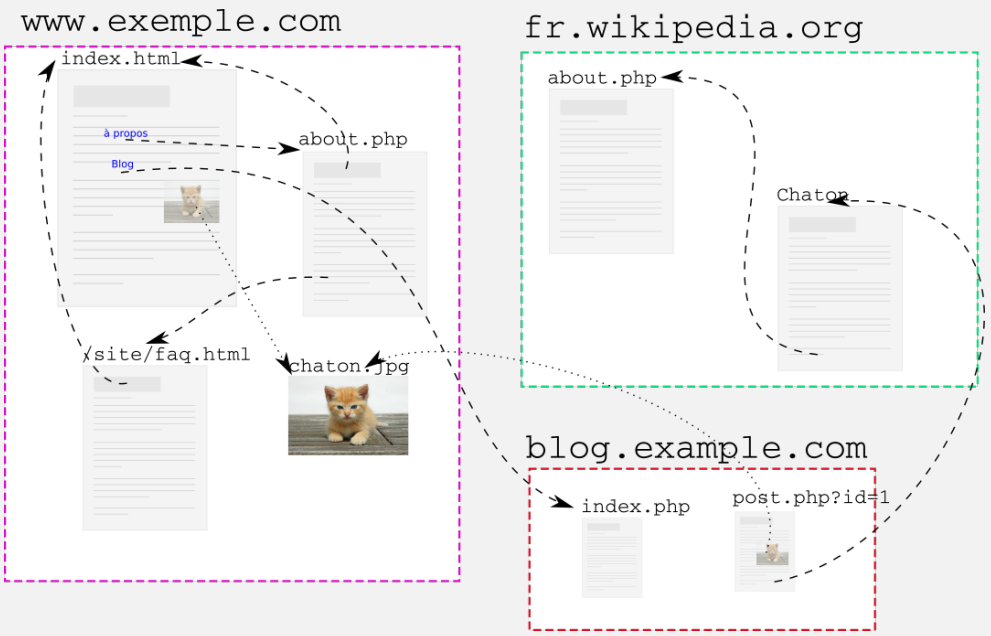
3.4 Concepts de base d'HTTP

— **Interaction** avec des ressources hypertexte :
  — *Quoi* : **Ressources** accessibles via Internet (documents hypertextes, images, etc.)
  — *Où* : **URL** pour localiser les ressources sur des serveurs Internet
  — *Comment* : **HTTP** comme protocole de communication pour un client souhaitant interagir avec les ressources hébergées sur un serveur

On parle de resssources. Historiquement, ces ressources ont d'abord été des documents, mais par la suite, ces ressources correspondent à des entités gérées par des applications.
Depuis l'émergence du Web sémantique, ces ressources correspondent désormais aussi à des concepts abstraits, qui sont des instances sur lesquelles on peut raisonner.

3.5 Construction de la toile (*Web*)

— **Graphe** de ressources hypertexte/hypermedia **décentralisé**
— Les ressources référencent d'autres ressources (attribut `href` dans HTML)
— Les liens sont unidirectionnels
— Aucune garantie de disponibilité, cohérence
— *Follow your nose* :
  — trouver les liens,
  — accéder aux ressources liées
  — recommencer. . .



Hyper-reference, *hyper-link*
Historiquement, **HTML** comme langage de mise en forme pour la visualisation des ressources de type "document hypertexte"
Le navigateur permet de suivre les liens dans le graphe : le graphe n'existe pas (à part dans un cache) tant qu'on n'a pas essayer de naviguer dessus.
Seul point "stable" : le système DNS, comme pour tout Internet.

- Composants de ce graphe :
  - 3 sites/serveurs Web : `www.example.com`, `blog.example.com`, `fr.wikipedia.org`
    - (peut-être sur 2 serveurs (2 adresses IP) : `example.com` et `fr.wikipedia.org` ?)
  - Documents/ressources :
    - `http://www.example.com/index.html` (HTML)
    - `http://www.example.com/site/faq.html` (HTML)
    - `http://www.example.com/about.php` (HTML)
    - `http://www.example.com/chaton.jpg` (image)
    - `http://blog.example.com/index.php` (HTML)
    - `http://blog.example.com/post.php?id=1` (HTML)
    - `http://fr.wikipedia.org/about.php` (HTML)
    - `http://fr.wikipedia.org/Chaton` (HTML)
  - Liens :
    - Inclusions images
      - Même site
      - Autre site
    - Hyper-liens navigables
      - Document du même site
      - Document sur un autre site
      - Fragment à l'intérieur d'un document

3.6 Localisation des ressources

- Objectif : nommer, localiser et accéder à l'information
- Solution : **URL (*Uniform Resource Locator*)** : identification universelle de ressource composée de 3 parties :
  1. le **protocole** (*comment*)
  2. l'identification du serveur Web, le **nom DNS** (*où*)
  3. l'**emplacement de la ressource** sur le serveur (*quoi*)

- plus tard, on raffine avec le concept d'URI

3.7 Structure des URLs

— URL type :

— Composantes :

1. protocole : en majorité `http` ou `https`
2. adresse / nom du serveur : `www.monsite.fr`
3. chemin d'accès à la ressource / document :
   `/projet/doc.html` (ne garantit pas que le résultat est un document HTML)

3.7.1 URLs plus détaillées

Exemple d'URL plus complexe :

1. protocole : `http` ou `https` (mais aussi `ftp`, `file`, `mailto`, `gopher`, `news`,...)
2. autorité : nom du serveur : `www.monsite.fr` + port : `8000`
3. chemin d'accès à la ressource : `/projet/doc`
4. requête : deux paramètres : `id` et `f`
5. fragment : `num42` à l'intérieur du document

> Il existe aussi un sur-ensemble des URLs, les URI (*Uniform Resource Identifier*). On se préoccupe plus d'identifier une ressource que de savoir comment y accéder. En pratique, cette subtilité ne change pas grand chose dans le cadre de ce cours.
> Pour plus de détails, voir la spécification générale des URIs (RFC 3986).

3.8 Rappel : Services sur Internet (TCP/IP)

— Un service sur Internet est caractérisé par :
  — une **application TCP/IP**
  — un **protocole**
  — un **numéro de port**
— Fonctionnement en mode Client/Serveur au dessus des couches réseau (IP, TCP ou UDP, ...)
— Ports qui nous intéressent :
  — 80 par défaut (HTTP)
  — 443 par défaut (HTTPS)
  — 8000, par exemple si serveur configuré pour écouter sur ce port

> Le Web n'est qu'une application parmi d'autres : ne pas confondre Internet le le Web
> Services nombreux et variés.
> Dans ce cours, on suppose une connaissance de base de TCP/IP et des réseaux.

3.9 Rappel : adresses IP

— Les URLs peuvent contenir l'adresse IP (v4 ou v6) du serveur. Ex. : `http://192.168.0.1/index.html`
  — Le serveur doit être lancé et écouter sur cette même IP
  — Attention aux problèmes de routage des paquets (firewall, NAT, ...)
— Dans les TPs on utilisera souvent `localhost` : `127.0.0.1` (en IPv4), dans l'environnement de développement Symfony, car client et serveur sur la même machine.

En TP on se connectera à un serveur Web tournant en local, démarré depuis l'environnement de test de Symfony, donc tout se fait sur `http://localhost:8000/`
Pour tester "comme en vrai", ça peut devenir un peu plus compliqué au niveau réseau, selon la configuration de l'OS. Mais ceci dépasse les limites du présent cours.
Le serveur peut connaître l'adresse apparente d'un client, mais ne peut pas toujours en faire quelque chose (proxy, NAT, etc.). Dans HTTP, le serveur n'en tient en général pas compte pour déterminer son comportement.

3.10 Construction de la requête

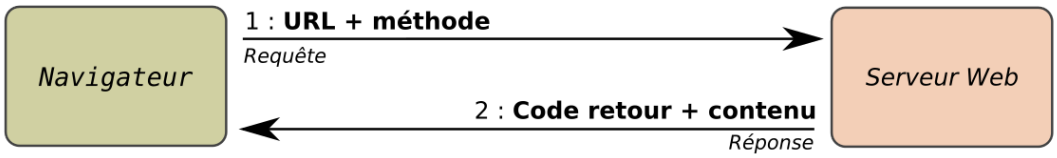
1. Le client (navigateur) interprète l'**URL**
2. Il demande au service de nom **une adresse IP** corrrespondant au nom du site (`www.monsite.fr`)
3. Il établit une **connexion TCP** avec le serveur à cette adresse, sur le numéro de port associé au protocole : 80 pour HTTP (ou 443 pour HTTPS, ...)
4. Il formule sa **requête** d'action (par exemple via la méthode `GET`) sur une ressource (`/projet/doc.html`), en écrivant un message (ex. : `"GET /projet/doc.html"`)

Et, depuis HTTP 1.1 il indique au serveur qu'il souhaite parler au site en question avec l'en-tête `Host`, pour le cas où plusieurs sites Web seraient gérés par le même serveur qui écoute sur cette adresse IP.

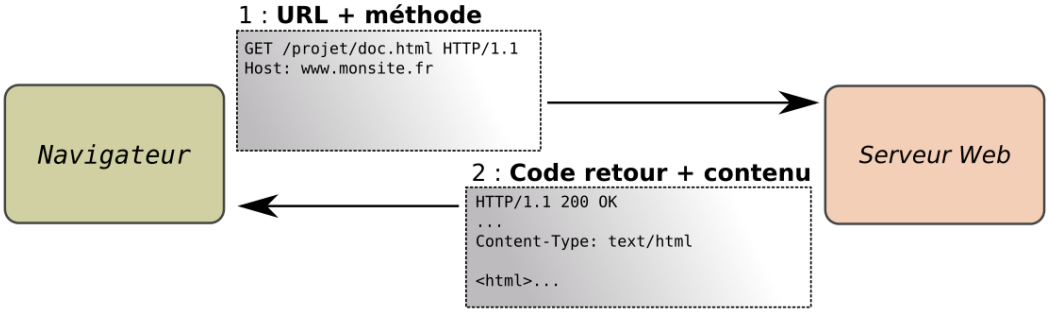
3.11 Traitement de la requête par le Serveur Web

1. Analyse la requête reçue (`GET /projet/doc.html`)
2. Vérification de sa validité (rien à cet emplacement ?), des autorisations d'accès...
3. Envoi d'une **réponse** au client avec :
   — un **code de retour**
   — du **contenu** (document existant ou résultat d'une exécution)
   ou :
   — un message d'erreur, ou une demande d'authentification, ou une adresse de redirection
4. Fermeture de la connexion

3.12 Requête - Réponse



3.12.1 Contenu des messages



On verra plus loin le détail de ce qui circule dans les messages de requêtes et de réponses.

3.13 Production du contenu sur le serveur

— URL reçue : identifie la **ressource** demandée
— En fonction du contexte de la requête, le serveur peut renvoyer différents **documents** pour la même URL
— La façon dont le serveur fabrique le contenu renvoyé lui appartient : **le client ne peut rien supposer**, juste suivre les liens hypertextes.
— Arborescence de chemins d'accès aux ressources accessibles (dans les URL) **différente** d'une éventuelle arborescence de stockage effectif à l'intérieur du serveur.
  — Peut-être envoi d'un document stocké sur un système de fichier ?
  — Peut-être une application générant des documents dynamiquement

Le contexte dépend uniquement de la méthode HTTP, de l'URL accédée et des en-têtes, mais pas des requêtes précédentes (sans état)
Le serveur Web va faire un certain nombre de choix qui dépendent de la façon dont il est configuré, avec des règles à appliquer relativement élaborées.
On ne couvre pas ces aspects de configuration dans le présent cours.

3.14 URLs absolues ou relatives

— Les URL permettent d'établir des **liens** entre documents
  — sur le même serveur
  — entre différents serveurs
— Liens sur le même serveur :
  — chemin absolu / chemin relatif
  — analogie avec chemin de fichier Unix : `.`, `..`, `/`
  — lien intra-document : fragments/ancres : `doc.html#conclusion`
  — convention : lien dans l'espace d'un utilisateur : `~taconet/menu.html`

3.14.1 Exemple

Document source	Ressource liée	Emplacement réel
`http://w.e.c/index.html`	`about.php`	`http://w.e.c/about.php`
`http://w.e.c/about.php`	`/`	`http://w.e.c/` (contenu de `index.html`)
`http://w.e.c/site/faq.html`	`../index.html`	`http://w.e.c/index.html`
`http://w.e.c/index.html`	`./chaton.jpg`	`http://w.e.c/chaton.jpg`
`http://w.e.c/index.html`	`http://b.e.c/`	`http://b.e.c/` (contenu de `index.php`)
`http://w.e.c/about.php`	`/site/faq.html`	`http://w.e.c/site/faq.html`
`http://b.e.c/post.php?id=1`	`http://w.e.c/chaton.jpg`	`http://w.e.c/chaton.jpg`
`http://b.e.c/post.php?id=1`	`http://f.w.o/Chaton`	`http://f.w.o/Chaton`
`http://f.w.o/Chaton`	`/about.php`	`http://f.w.o/about.php`
`http://f.w.o/Chaton`	`/about.php#terms`	`http://f.w.o/about.php` (`<div id"terms">`)

3.15 Transmettre des données au serveur

— Commment le client peut-il transmettre des données au serveur (applications dynamiques) ?
— L'URL peut contenir des informations variables :
  — des arguments d'un `GET`
  `http://www.monsite.fr/projet/service.php?id=3&nb=42`

nom	valeur
id	3
nb	42

— La requête peut transmettre des données : méthode `POST` (voir ci-après)

3.16 Protocole sans état

— Chaque requête est effectuée de façon indépendante, et le serveur ne garde pas la trace des requêtes précédentes effectuées par le même client.
— De base, pas de gestion de session dans HTTP (mais des solutions existent).

4 Spécifications de base HTTP

4.1 Spécifications (version de juin 2014) :

— RFC 7230 *HTTP/1.1 : Message Syntax and Routing*
— RFC 7231 *HTTP/1.1 : Semantics and Content*
— RFC 7232 *HTTP/1.1 : Conditional Requests*
— RFC 7233 *HTTP/1.1 : Range Requests*
— RFC 7234 *HTTP/1.1 : Caching*
— RFC 7235 *HTTP/1.1 : Authentication*

Les RFC (*Request For Comments)* peuvent avoir valeur de standard de l'Internet (cf. `https://fr.wikipedia.org/wiki/Request_for_comments`).
Les spécifications d'HTTP ont été réécrites récemment, même si HTTP 1.1 est très vieux.

4.2 Versions HTTP historiques

— Version d'origine, notée HTTP 0.9 (1991)
  — Une seule méthode : `GET`
  — Pas d'en-têtes
  — Chaque requête = une connexion TCP
— HTTP version 1.0 (1996)
  — Introduction des en-têtes ==> échange de "méta" informations
  — Nouvelles possibilités :
    — utilisation de caches
    — authentification
    — connexion persistante
  — Ajout de méthodes : HEAD, POST...

4.3 HTTP version 1.1 (1997 -)

— Mode Connexions persistantes par défaut
— Exemple : page d'accueil avec 5 images
  — HTTP 0.9 ==> 6 connexions/déconnexions TCP
  — HTTP 1.1 ==> 1 SEULE connexion TCP
— Possibilité de transférer des séries d'octets
— Introduction du "*multihoming*" càd possibilité de gérer plusieurs noms de sites par un serveur

Bien que la version 1.1 soit assez ancienne, c'est aujourd'hui encore la version principalement utilisée sur le Web, donc celle avec laquelle vous allez travailler dans ce cours.

4.4 HTTP version 2 (2015)

— Tout nouveau
— Performances (binaire, compression, 1 seule connexion,...)
Spécs : Hypertext Transfer Protocol Version 2 (HTTP/2) RFC 7540

Pas forcément encore très diffusée sur le Web.
Pas utilisé dans le cadre de ce cours.

4.5 Format des messages HTTP

— Très peu de types de requêtes / méthodes (moins d'une dizaine)
— Tout en texte `ASCII` 7 bits (facile à débugger)
— Syntaxe très simple, avec 4 zones

1. *ligne 1* : **requête** (ou **statut réponse**)
2. *plusieurs lignes* : **en-têtes** (optionnels)
3. *ligne vide* (séparateur)
4. *reste du message* : **contenu** (qui peut être vide)

Les en-têtes sont au format RFC822, déjà popularisé pour l'email. Pour transporter autre-chose que de l'ASCII (binaire, accents, etc.) il faudra encoder / décoder le contenu.

4.6 Structure d'une requête client

— Canevas :

```
Méthode Document Version_HTTP
En-têtes
            Ligne vide
Contenu du Message          optionnel
(saisie d'un formulaire, document à publier)
```

— Exemple :

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

4.7 Structure d'une réponse serveur

— Canevas :

```
Version_HTTP Code Signification
En-têtes
             Ligne vide
Contenu du Message
(document HTML, image, ...)
```

— Exemple :

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.
```

Le contenu du message (ce qui suit la ligne vide) est particulièrement important, mais devra être interprété en fonction des en-têtes. Comme il s'agit en général d'autre chose qu'un texte simple en ASCII, les en-têtes relatifs aux formats et méthodes d'encodage sont particulièrement importants.
Le contenu du message, les données est appelé *payload* en anglais.

5 Méthodes

5.1 Actions sur les ressources (CRUD)

— Le client souhaite effectuer des **actions/opérations** sur les ressources du serveur
— Il invoque en conséquence différents types de requêtes, **méthodes** :

	Opération	**Méthode HTTP**	SQL
C	*Create* (création)	**PUT** / **POST**	INSERT
R	*Read* (accès)	**GET**	SELECT
U	*Update* (mise-à-jour)	**PUT** / **PATCH**	UPDATE
D	*Delete* (suppression)	**DELETE**	DELETE

RFC7231 : HTTP/1.1 Semantics and Content

5.2 Exemple de requête GET

— Demande du document `premier.html` par le navigateur (firefox)
Regardez les en-têtes dans les outils pour le développeur de vos navigateurs

5.3 Exemple de réponse à un GET

— Envoi du document HTML par le serveur Web Apache
La première ligne contient le code de statut/retour : `200 OK`

5.4 Méthode GET

Récupérer la représentation actuelle d'une ressource cible
— Seule méthode à l'origine
–> usage très simple : `GET doc.html`
— Méthode la plus utilisée qui permet de :
— Récupérer le contenu d'un document (fichier, image. . .)
— Activer l'exécution d'un programme sur le serveur
— Et même envoyer des données `programme?nom=paul`
— Avec GET, le contenu du message dans la requête est toujours vide

5.5 Autres méthodes

— **HEAD** : comme GET, mais récupérer seulement l'en-tête
— **POST** : envoi de données pour traitement coté serveur (ex : contenu d'un formulaire HTML)
— **PUT** : envoi du contenu d'un document, pour enregistrement sur le serveur
— **DELETE** : suppression d'un document
— **CONNECT** : établir un tunnel vers le serveur hébergeant une ressource (serveur proxy)
— **OPTIONS** : demande des options gérées par le serveur pour l'accès à une ressource
— **TRACE** : effectue un test d'accès complet le long du chemin d'accès à une ressource (ping / echo)

5.6 *Keep Calm*

En pratique, au-delà de cette séance, vous utiliserez et gérerez principalement :
— `GET`
— `POST`
L'affaire se corsera pour ceux qui s'intéresseront aux services Web fonctionnant avec des API sur HTTP (REST)

5.7 Codes de réponse du serveur

Codes de retour (*Status codes*)
— Code numérique, complété par un message littéral
— 5 catégories :
— De 100 à 199 : Messages d'information
— De 200 à 299 : Succès de la requête
Exemple : `200 Okay`
— De 300 à 399 : Redirections
— De 400 à 499 : Erreurs du client
Exemples : `404 Not Found`, `403 Forbidden`
— De 500 à 599 : Erreurs du serveur
Exemple : `501 Internal Server Error`

Et 418 *I'm a teapot* . . .

6 En-têtes sans mal de tête

6.1 Rôle des en-têtes (*headers*)

— Ajoute du contexte (propriétés du client, autorisations, . . .)
— Détaille les caractéristiques des flux (encodage, taille . . .)

— Optimisations (cache, ...)
— Gestion d'une session au-dessus d'un protocole sans état (cookies, ...)
— Éléments propres à l'application (extensible)

Il y a beaucoup d'en-têtes, mais seulement certains d'entre-eux sont réellement importants dans le cadre de ce cours (en gras)

6.2 En-têtes généraux (requêtes ou réponses)

— Les caches et proxy :
  — `Cache-Control`
  — `Pragma`
  — `Via`
— La connexion :
  — `Connection`
— La date :
  — `Date`
— Le codage :
  — `MIME-Version`
  — `Transfer-Encoding`

Les en-têtes généraux peuvent être présents aussi-bien dans les requêtes que dans les réponses HTTP.

6.3 En-têtes de requêtes

— Préférences du client : types, caractères, langage...
  — `Accept`
  — `Accept-Encoding`
  — `Accept-Charset`
  — `Accept-Language`
— Identification du client/serveur : site, e-mail, source...
  — `Host`
  — `From`
  — `Referer`
  — `User-Agent`

Ces en-têtes sont définis par le client qui les ajoute à partir de la 2ème ligne du message de requête

`Accept` définit les priorités sur le format des données que le client aimerait recevoir de la part du serveur

`Host` précise au serveur sur quel site exactement les chemins d'URL sont spécifiés

6.4 En-têtes de requêtes (suite)

— Contrôle d'accès et sessions (login, cookies...)
  — `Authorization`
  — `Cookie`
— Conditions pour le serveur (sur la date...)
  — `If-Modified-Since`
  — `If-Match`
  — `If-Range`
  — `If-Unmodified-Since`
— Autres (pour proxy)
  — `Max-Forwards`

`Authorization` permet de transmettre au serveur des *lettres de créance* pour se faire connaître (cf. séance ultérieure)

`Cookie` permet de transmettre (plus ou moins automatiquement) des données au serveur à chaque visite (requête GET). On verra que ça permet notamment de gérer des sessions dans les applications.

6.5 En-têtes de réponses

- Informations sur le contenu : type, taille, URL, encodage...
  - `Content-Type`
  - `Content-Length`
  - `Content-Encoding`
  - `Content-Location`
- Informations sur la date de modification ou la durée
  - `Last-Modified`
  - `Expires`
- Autres :
  - `Allow : méthodes autorisées`
  - `Etag : entité de balisage`

Ces en-têtes précisent notamment des informations (méta-données) relatives au contenu qui va être transmis par le serveur dans la suite de la réponse.

`Content-Type` précise le format de fichier qui sera présent dans les données

6.6 En-têtes de réponses (suite)

- Identification du serveur : nom, formats...
  - `Server`
  - `Accept-Range`
  - `Location`
- Compléments pour le client : durée, âge...
  - `Age`
  - `Retry-After`
  - `Warning`
- Demande d'authentification
  - `WWW-Authenticate`
- Envoi de cookies
  - `Set-Cookie`

`Location` est notamment utilisé en cas de redirection (code de statut 300 et +) pour indiquer la cible de la redirection

`WWW-Authenticate` complémentaire au code de retour indiquant que le contenu demandé est protégé (`401 Unauthorized`) et indique au client qu'il doit s'authentifier (sera vu dans une séance ultérieure)

`Set-Cookie` permet au serveur de transmettre au client des données à conserver dans les cookies associés à cette URL (cf. plus loin).

7 Outils HTTP du développeur Web

7.1 Moniteur réseau dans navigateur

Cf. démonstration du "moniteur réseau" dans les outils du développeur Web intégrés aux navigateurs.
- Firefox
- Chrome

Sera vu en partie TP
Mais aussi Poster pour Firefox et beaucoup d'autres outils

7.2 CURL en ligne de commande

`https://curl.haxx.se/`
Client en ligne de commande

```
man curl
```

On verra dans la partie TP comment s'en servir, par exemple avec `curl -v 2>&1 | less`
Aussi `wget`, ou `get` autres outils populaire en ligne de commande.

7.3 Tester la dispo, caches et archives

— `http://www.downforeveryoneorjustme.com/`
— Internet archive's Wayback Machine
— Cache des moteurs de recherche

8 HTTP avancé

Notions avancées. Pas trop grave si tout ça n'est pas clair dès le départ.

8.1 Sécurité

— HTTP est en clair !
— => **Utiliser HTTPS**
— Mais attention : avoir confiance dans le DNS, ou se fier aux certificats

On verra les aspects de sécurité plus en détail en séance ultérieure.

8.2 Contrôle d'accès

Sera vue en séance 4

8.3 Redirections

Exemple : `http://www.example.com/` -> `http://www.example.org/`

```
curl http://www.example.com/
```

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: curl/7.50.1
Accept: */*

HTTP/1.1 301 Moved Permanently
Location: http://www.example.org/
Content-Type: text/html
Content-Length: 174

<html>
<head>
<title>Moved</title>
</head>
<body>
<h1>Moved</h1>
```

```
<p>This page has moved to <a href="http://www.example.org/">http://www.example.org/</a>.</p>
</body>
</html>
```

Le navigateur Web n'affichera pas ce contenu HTML : par défaut, il suivra le lien et recommencera un `GET` sur l'URL reçue dans l'en-tête `Location`
Utiliser `curl -L` permet de suivre les redirects.
Par défaut les navigateurs effectuent les redirects, et les outils du développeur masquent parfois celà, sauf si on active une option permettant de garder l'historique des requêtes précédentes.

8.4 Proxy et cache

- Serveur *Proxy* : serveur HTTP intermédiaire
  - Diminuer le trafic.
- Cache : stockage d'informations (statiques)
  - Le navigateur dispose d'un cache à deux niveaux : en mémoire pour l'historique et sur disque entre les sessions.

  - Exemple : 50 utilisateurs d'un site accèdent à une même page :
    - directement, elle sera transférée 50 fois
    - via un proxy, un seul transfert entre l'origine et le proxy
  - Un proxy offre aussi d'autres services (autres protocoles, contrôles...)

Dans le cadre du cours, on n'explicite pas tous ces aspects. L'augmentation générale de la bande passante et des applications dynamiques tend à rendre les proxies moins utiles, en général.

8.5 Ressources / Documents

- HTTP agit sur des ressources, qui ne sont pas forcément des documents stockés physiquement sur le serveur.
- Pas uniquement du HTML (XML, JSON, RSS, HTML, PDF, JPG, CSS, JS, etc.).
- Types MIME, encodage.
- Négociation de contenu entre le client et le serveur
- Web Sémantique

8.6 Content-Negociation

- Le résultat de la requête sur une même URL dépend des en-têtes de la requête.

```
Accept: application/json, application/xml;q=0.9, text/html;q=0.8,
      text/*;q=0.7, */*;q=0.5
```

Priorité	Description
`q=1.0`	`application/json`
`q=0.9`	`application/xml`
`q=0.8`	`text/html`
`q=0.7`	`text/*` (ie. tout type de texte)
`q=0.5`	`*/*` (ie. n'importe quoi)

9 Applications Web

9.1 Applications pour l'utilisateur

Utiliser HTTP pour faire fonctionner des applications dans un navigateur.
Cf. séances suivantes.

Il faut bien comprendre HTTP pour être capable de tester / debugger. Sinon on ne sait pas déterminer si les problèmes viennent du navigateur ou de l'application côté serveur (ou des deux).

9.2 Services Web

— Utiliser HTTP pour faire interagir différents programmes :
  — programme client
  — programme serveur
— Services Web, APIs
— Problème : HTTP est vraiment de très bas niveau, donc il faut ajouter pas mal de standardisation par-dessus, ou que les concepteurs des programmes des deux côtés se concertent.

> L'utilisation de HTTP pour faire fonctionner des APIs ne sera pas couvert en détail dans ce cours.

9.3 Principes REST

— *REpresentational State Transfer* : passer de documents hypertextes à des applications Web
— dans la Thèse de Roy Fielding en 2000 :
— **HTTP 1.1** pour des applications (APIs)
  — Verbes/méthodes : `GET`, `POST` + `PUT` ou `PATCH`, `DELETE`, `OPTIONS`
  — Codes de retour HTTP
— Pas de session ?
— **The six constraints of the architectural style of REST** (voir dessous)

9.4 6 contraintes architecture REST

1. Client-serveur
2. Serveur sans mémoire de l'état du client (*stateless*)
3. Cache (client)
4. Système à plusieurs couches
5. Code sur le client *on-demand* (optionnel)
6. Contrainte d'interface uniforme
   (a) **Indentifiants de ressources**,
   (b) Représentations des ressources,
   (c) Messages auto-descriptifs,
   (d) **HATEOAS** : *Hypermedia As The Engine Of Application State*

On reverra tout cela plus en détail plus tard

10 Postface

10.1 Bibliographie additionnelle

— *HTTP: The Protocol Every Web Developer Must Know - Part 1* par Pavan Podila (anglais)

10.2 Crédits photo

— Minitel 2 par Frédéric BISSON - CC BY 2.0, via `https://commons.wikimedia.org/w/index.php?curid=45302325`
— $\text{Tim}_{\text{Berners}}\text{-Lee}_{\text{CP2.jpg}}$ par Silvio Tanaka - CC BY 2.0 via `https://commons.wikimedia.org/w/index.php?curid=6560968`
— chaton.png par Joseph SARDIN - CC BY 2.0

10.3 Et maintenant

— Poursuivre le dernier TP (C) en hors-présentiel
— Surfer avec l'inspecteur réseau, dans un nouveau profil de navigateur, ou en session privée