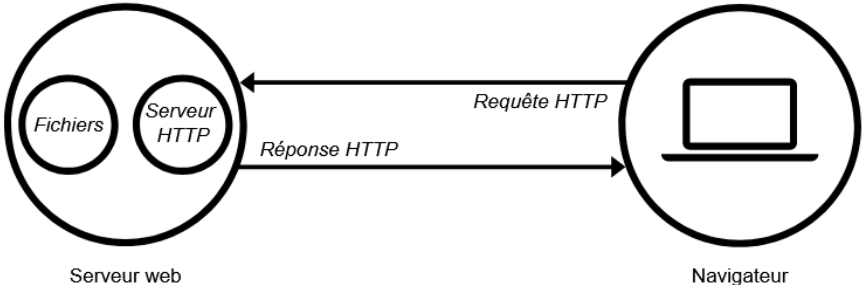
TD 2 - Serveurs Web et HTTP

I. Serveurs Web

Un « serveur web » peut faire référence à des composants logiciels (*software*) ou à des composants matériels (*hardware*) ou à des composants logiciels et matériels qui fonctionnent ensemble.

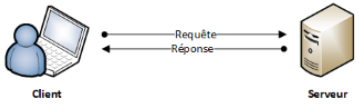
1. Au niveau des composants matériels, un serveur web est un ordinateur qui stocke les fichiers qui composent un site web (par exemple les documents HTML, les images, les feuilles de style CSS, les fichiers JavaScript) et qui les envoie à l'appareil de l'utilisateur qui visite le site. Cet ordinateur est connecté à Internet et est généralement accessible via un nom de domaine tel que `mozilla.org`.
2. Au niveau des composants logiciels, un serveur web contient différents éléments qui contrôlent la façon dont les utilisateurs peuvent accéder aux fichiers hébergés. On trouvera *a minima* un serveur *HTTP*. Un serveur HTTP est un logiciel qui comprend les URL et le protocole HTTP (le protocole utilisé par le navigateur pour afficher les pages web).

Au niveau le plus simple, à chaque fois qu'un navigateur a besoin d'un fichier hébergé sur un serveur web, le navigateur demande (*on dit qu'il envoie une requête*) le fichier via HTTP. Quand la requête atteint le bon serveur web (*matériel*), le serveur HTTP (*logiciel*) renvoie le document demandé, également grâce à HTTP.



Les types d'architecture client-serveur

- Si toutes les ressources nécessaires sont présentes sur un seul serveur, on parle d'architecture à deux niveaux ou 2 tiers (1 client + 1 serveur).



- Si certaines ressources sont présentes sur un deuxième serveur (par exemple des bases de données), on parle d'architecture à trois niveaux ou 3 tiers (1 client interroge le premier serveur qui lui-même interroge le deuxième serveur).

- Au delà de 3 acteurs, on parle d'architecture à n tiers.

L'invention du World Wide Web

En 1989, alors qu'il travaillait au CERN, Tim Berners-Lee proposa la création d'un système hypertexte sur internet. Initialement nommé *Mesh*, il prit le nom de World Wide Web lors de sa mise en place en 1990. Bâti sur les protocoles existants TCP et IP, il consistait en quatre éléments de base :

- Un format textuel pour représenter les documents hypertextes, l'*HyperText Markup Language* (HTML).
- Un protocole simple pour échanger ces documents, l'*HyperText Transfer Protocol* (HTTP).
- Un logiciel client pour exposer (et modifier) ces documents, le premier navigateur web nommé *WorldWideWeb*.
- Un serveur pour garantir l'accès au document, version initiale du *httpd*.

Pour publier un site web, vous aurez besoin d'un serveur web **statique** ou **dynamique**.

Un serveur web **statique** (*aussi appelé stack*) est composé d'un ordinateur (*matériel*) et d'un serveur HTTP (*logiciel*). Il est appelé « statique » car le serveur envoie les fichiers hébergés « tels quels » vers le navigateur.

Un serveur web **dynamique** possède d'autres composants logiciels, dont certains qu'on retrouve fréquemment dont un *serveur d'applications* et une *base de données*. Il est appelé « dynamique » car le serveur d'applications met à jour les fichiers hébergés avant de les envoyer au navigateur via HTTP.

Par exemple, afin de produire la page web que vous voyez sur votre navigateur, le serveur d'applications serveur peut utiliser un modèle HTML et le remplir avec des données. Ainsi, des sites comme MDN ou Wikipédia ont des milliers de pages mais il n'existe pas un document HTML réel pour chacune de ces pages. En fait, il y a quelques modèles (ou gabarits) HTML qui sont utilisés avec une gigantesque base de données. Cette organisation permet de mieux mettre à disposition le contenu et de maintenir plus efficacement le site.

Un serveur web dynamique implique toujours (au minimum) une architecture 3-tiers : serveur d'applications/web *(qui interprète le langage de contrôle des données, comme PHP)* et serveur de données *(BDD, comme MySQL)*

II. HTTP

Communiquer via HTTP

Un serveur web supporte le protocole HTTP (pour *HyperText Transfer Protocol* en anglais soit Protocole de transfert hypertexte). Comme son nom l'indique, HTTP définit comment transférer des fichiers hypertextes (c'est-à-dire des documents web liés entre eux) entre deux ordinateurs.

Ici, un *protocole* est un ensemble de règles définissant la communication entre deux ordinateurs. HTTP est un protocole textuel, sans état.

Textuel

Toutes les commandes qui sont échangées sont du texte pouvant être lu par un humain.

Sans état

Ni le serveur, ni le client (l'ordinateur sur lequel est le navigateur) ne se souviennent des communications précédentes. Par exemple, si on utilisait uniquement HTTP, un serveur ne pourrait pas se souvenir si un mot de passe a été saisi ou si une transaction est en cours (pour gérer cela, il faut utiliser un serveur d'applications).

HTTP fournit des règles claires qui indiquent comment un client et un serveur communiquent.

Sur un serveur web, le serveur HTTP est responsable du traitement des requêtes reçues et de leurs réponses.

1. Une fois qu'il a reçu une requête, le serveur HTTP vérifie que l'URL demandée correspond à un fichier existant.
2. Si c'est le cas, le serveur envoie le fichier vers le navigateur du client. Sinon, le serveur d'applications génère le fichier nécessaire.
3. Si le fichier n'existe pas ou que le traitement est impossible, le serveur web renvoie un message d'erreur au navigateur. Le message d'erreur le plus fréquemment rencontré est `404 Page non trouvée`

HTTP flow

When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:

1. Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
2. Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same. For example:

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

3. Read the response sent by the server, such as:

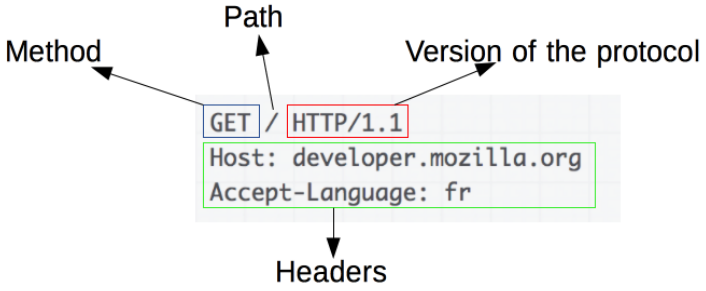
```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested we
```

4. Close or reuse the connection for further requests.

If HTTP pipelining is activated, several requests can be sent without waiting for the first response to be fully received. HTTP pipelining has proven difficult to implement in existing networks, where old pieces of software coexist with modern versions. HTTP pipelining has been superseded in HTTP/2 with more robust multiplexing requests within a frame.

Architecture d'une requête HTTP



En informatique, le **code HTTP** (aussi appelé *code d'état*) permet de déterminer le résultat d'une requête ou d'indiquer une erreur au client. Ce code numérique est destiné aux traitements automatiques par les logiciels de client HTTP. Ces codes d'état ont été définis par la RFC 2616[1], en même temps que d'autres codes d'état, non normalisés mais très utilisés sur le Web. Ils ont été ensuite étendus par la RFC 7231[2].

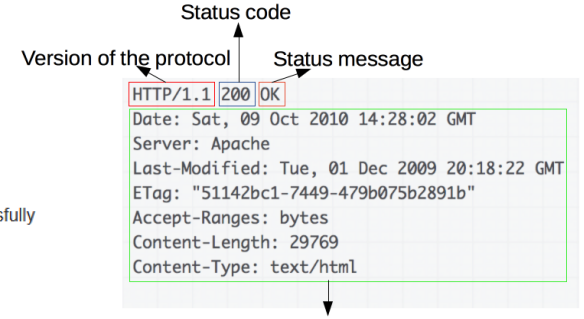
Le premier chiffre du code d'état est utilisé pour spécifier une des cinq catégories de réponse (informations, succès, redirection, erreur client et erreur serveur).

Les codes les plus courants sont :

- 200 : succès de la requête ;
- 301 et 302 : redirection, respectivement permanente et temporaire ;
- 401 : utilisateur non authentifié ;
- 403 : accès refusé ;
- 404 : page non trouvée ;
- 500 et 503 : erreur serveur ;
- 504 : le serveur n'a pas répondu.

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (`100–199`),
2. Successful responses (`200–299`),
3. Redirects (`300–399`),
4. Client errors (`400–499`),
5. and Server errors (`500–599`).



Architecture d'une réponse HTTP

Méthodes HTTP

HTTP définit un ensemble de **méthodes de requête** qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée. Ces méthodes sont souvent appelées *verbes HTTP*. Chacun d'eux implémente une sémantique différente mais certaines fonctionnalités courantes peuvent être partagées par différentes méthodes (e.g. une méthode de requête peut être sûre (*safe*), idempotente ou être mise en cache (*cacheable*)).

`GET`

La méthode GET demande une représentation de la ressource spécifiée. Les requêtes GET doivent uniquement être utilisées afin de récupérer des données.

`HEAD`

La méthode `HEAD` demande une réponse identique à une requête GET pour laquelle on aura omis le corps de la réponse (on a uniquement l'en-tête).

`POST`

La méthode `POST` est utilisée pour envoyer une entité vers la ressource indiquée. Cela entraîne généralement un changement d'état ou des effets de bord sur le serveur.

`PUT`

La méthode `PUT` remplace toutes les représentations actuelles de la ressource visée par le contenu de la requête.

`DELETE`

La méthode `DELETE` supprime la ressource indiquée.

`CONNECT`

La méthode `CONNECT` établit un tunnel vers le serveur identifié par la ressource cible.

`OPTIONS`

La méthode `OPTIONS` est utilisée pour décrire les options de communications avec la ressource visée.

`TRACE`

La méthode `TRACE` réalise un message de test aller/retour en suivant le chemin de la ressource visée.

`PATCH`

La méthode `PATCH` est utilisée pour appliquer des modifications partielles à une ressource.

Ressources, URI et URL

La cible d'une requête HTTP est appelée une "ressource", elle ne possède pas de type particulier. Il peut s'agir d'un document, d'une photo ou de n'importe quoi d'autre. Chaque ressource est identifiée à l'aide d'une *Uniform Resource Identifier* (URI) utilisé au sein de HTTP pour identifier les ressources.

La forme la plus commune des URI est l'URL (*Uniform Resource Locator* (URL)) que l'on connaît sous le nom d'adresse web.

```
https://developer.mozilla.org
https://developer.mozilla.org/fr/docs/Learn/
https://developer.mozilla.org/fr/search?q=URL
```

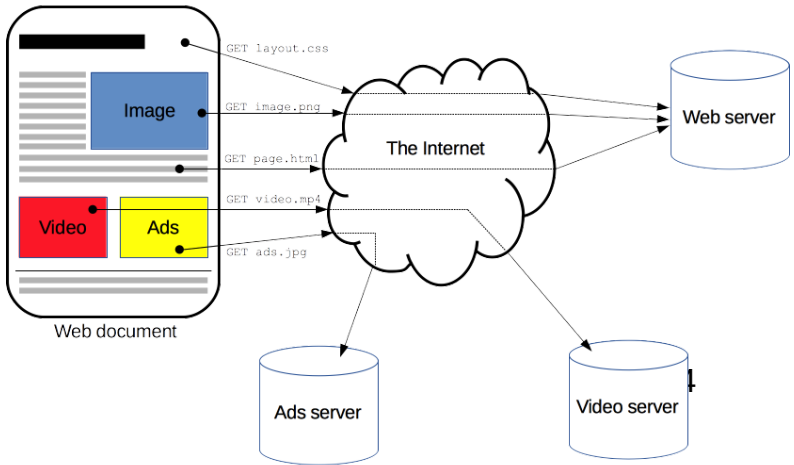
Vous pouvez entrer chacune de ces URLs dans votre navigateur pour lui demander de charger la page associée (il s'agit ici de la ressource).

Une URL est composée de différentes parties, certaines obligatoires et d'autres facultatives.
Voici un exemple plus complet :

```
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument
```

Une page web est un document hypertexte *(== qui dépasse le texte)*, c'est à dire un document qui contient des ressources textuelles, mais aussi d'autres ressources, **mais surtout** un document composés de ressources venant d'endroit divers ! Une page web n'est pas un document fixe, mais un document composé de plusieurs ressources distantes (ou non) et qui se met à jour si ces ressources se mettent à jour. De plus, les documents hypertextes proposent des relations entre eux à partir de liens hypertextes.

Ci-dessous, la récupération d'une page web : on récupère chacune des ressources à l'aide de plusieurs requêtes



Syntaxe des URIs (Uniform Resource Identifiers)

Schéma ou protocole

http://www.example.com:80/path

Protocol

`http://` constitue le protocole, il indique le protocole qui doit être utilisé par le navigateur. Il s'agit généralement de HTTP ou de sa variante sécurisée HTTPS. Le Web nécessite l'un ou l'autre de ces protocoles néanmoins, les navigateurs sont capables de gérer d'autres protocoles tels que `mailto:` (pour ouvrir un client mail) or `ftp:` pour gérer un transfert de fichier. Essayez, lorsque vous naviguez, d'identifier les protocoles utilisés. Les schémas usuels sont :

Schéma	Description
data	URIs de données
file	Fichiers du système hôte sur lequel est installé le navigateur
ftp	File Transfer Protocol
http/https	Hyper text transfer protocol (sécurisé)
mailto	Adresse électronique
ssh	Secure shell
tel	téléphone
urn	Uniform Resource Names
view-source	code source de la ressource
ws/wss	connexions (chiffrées) WebSocket

Autorité

http://www.example.com:80/path/to/my

Domain Name

`www.exemple.com` est le nom de domaine ou l'autorité qui gère cet espace de noms. Il indique quel serveur Web est appelé. Il est aussi possible d'utiliser directement une adresse IP (IP address), néanmoins elles sont moins pratiques à manipuler pour des humains et sont donc moins fréquemment utilisées pour accéder à une ressource sur le Web.

Port

com:80/path/to/myfile.html?key1=valu

Port

`:80` constitue le port. Il indique la "porte" technique à utiliser pour accéder à une ressource sur un serveur web. Il est généralement omis puisque le serveur web utilisera par défaut les ports standards pour HTTP (port 80 pour HTTP et 443 pour HTTPS) pour permettre l'accès aux ressources qu'il héberge. Dans le cas où le port par défaut n'est pas celui utilisé, il est obligatoire de le spécifier.

Chemin

:80/path/to/myfile.html?key1=value1

Path to the file

`/chemin/du/fichier.html` constitue le chemin d'accès à la ressource sur le serveur web. Au début du Web, le chemin représentait un emplacement physique où le fichier était stocké, à l'heure actuelle il s'agit d'une abstraction gérée par le serveur web sans réelle existence physique..

Requête

?key1=value1&key2=value2#Some

Parameters

`?key1=value1&key2=value2` sont des paramètres additionnels fournis au serveur web. Ces paramètres sont un ensemble de clés/valeurs séparé par le symbole `&`. Le serveur web peut utiliser ces paramètres pour effectuer des tâches avant de retourner une ressource au client. Chaque serveur web possède ses propres règles en ce qui concerne la gestion des paramètres.

Fragment

lue2#SomewhereInTheDocument

Anchor

`#QuelquePartDansLeDocument` est une ancre vers un morceau de la ressource en particulier, elle constitue une sorte de marque-page à l'intérieur de la ressource. Cela permet au navigateur de savoir où aller pour afficher le contenu à l'emplacement de l'ancre. Au sein d'une page HTML par exemple, le navigateur défilera jusqu'à ce point. Pour un document vidéo ou audio, le navigateur essaiera d'accéder au temps indiqué par l'ancre. On notera que la partie située après le caractère #, aussi appelé le fragment, n'est jamais envoyé au serveur avec la requête.

Les types de ressources : MIME

Le **type Multipurpose Internet Mail Extensions (type MIME)** est un standard permettant d'indiquer la nature et le format d'un document. Il est défini au sein de la RFC 6838. L'Internet Assigned Numbers Authority (IANA) est l'organisme officiel responsable du suivi de l'ensemble des types MIME officiels existants. Une liste exhaustive et maintenue est consultable sur la page Media Types de l'IANA.

Les navigateurs utilisent le plus souvent le type MIME et non l'extension d'un fichier pour déterminer la façon dont ils vont traiter ou afficher un document. Il est donc important que les serveurs puissent correctement attacher le type MIME dans l'en-tête de la réponse qu'ils renvoient.

Structure générale

```
type/sous-type
```

La structure d'un type MIME est simple, elle est composée d'un type et d'un sous-type. Les deux chaînes de caractères sont séparées par un `'/'`. Les caractères d'espacement ne sont pas autorisés. Le *type* représente la catégorie et peut être *particulier* ou *composé* lorsqu'il regroupe plusieurs formats. Le *sous-type* est spécifique à chaque type.

Un type MIME est insensible à la casse mais il s'écrit usuellement en minuscule.

PSY - FÉE

Types particuliers

```
text/plain
text/html
image/jpeg
image/png
audio/mpeg
audio/ogg
audio/*
video/mp4
application/octet-stream
…
```

Les types *particuliers* indiquent la catégorie d'un document. Les valeurs possibles sont :

Type	Description	Exemple de sous-type communément associé
`text`	Représente n'importe quel document contenant du texte et qui est théoriquement lisible par un utilisateur.	`text/plain`, `text/html`, `text/css`, `text/javascript`
`image`	Représente n'importe quelle image. Les vidéos ne font pas partie de ce type bien que les images animées tels les GIFs animés) font partie de ce type.	`image/gif`, `image/png`, `image/jpeg`, `image/bmp`, `image/webp`
`audio`	Représente n'importe quel fichier audio.	`audio/midi`, `audio/mpeg`, `audio/webm`, `audio/ogg`, `audio/wav`
`video`	Représente n'importe quel fichier vidéo.	`video/webm`, `video/ogg`
`application`	Représente n'importe quelle donnée binaire.	`application/octet-stream`, `application/pkcs12`, `application/vnd.mspowerpoint`, `application/xhtml+xml`, `application/xml`, `application/pdf`

`text/plain` doit être utilisé pour tous les documents texte sans sous-type spécifique. De la même façon, les documents binaires sans sous-type ou dont le sous-type est inconnu doivent utiliser `application/octet-stream`.