TP1 - ETUDE DU PROTOCOLE HTTP

Objectif :

- Comprendre le fonctionnement du protocole HTTP basé sur TCP/IP.
- Comprendre le modèle client-serveur.
- Connaitre les commandes netstat, curl, autres.
- Connaitre quelques fonctionnalités courantes d'un serveur WEB.

INTRODUCTION

Nous passons une grande partie de notre temps (loisirs et/ou travail) sur le WEB, c'est-à-dire naviguer sur internet.

Mais quand nous tapons dans un navigateur l'URL suivante http://www.google.fr , que ce passe-t-il réellement entre le moment de la saisie et l'affichage de la page ?

La plupart des personnes ne se posent pas cette question. Mais quand on veut devenir développeur d'application Web il faut le savoir.

Que signifie le WWW placé devant la plupart des URL ? que signifie URL ? Que veut dire web ?

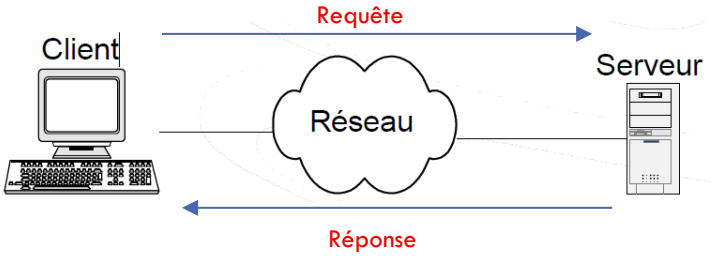
Pour le comprendre, il est nécessaire d'étudier les protocoles réseaux associés au WEB.

ANALYSE D'UNE TRAME HTTP :

La plupart des protocoles réseaux sont basés sur le modèle Client-Serveur.

On peut très illustrer le modèle **Client-Serveur** avec un restaurant.

- **Le client** : c'est une personne qui vient au restaurant et qui fait une **demande** ou **requête** : commander un plat. Dans le réseau, le client est l'ordinateur ou l'application qui envoie une requête (par exemple, demander une page web).
- **Le serveur** : c'est le restaurant et son personnel en cuisine qui **reçoit la commande**, prépare le plat, et le **renvoie au client**. Dans le réseau, le serveur est l'ordinateur ou le logiciel qui reçoit la requête du client et fournit la réponse (par exemple, le serveur web qui envoie la page demandée).
- **La commande** : c'est la **requête** envoyée par le client.
- **Le plat servi** : c'est la **réponse** du serveur à la requête.
- **La salle du restaurant** : c'est le **réseau** qui transporte la commande et la réponse entre le client et le serveur.



Pour résumé, le principe du Client-Serveur est relativement simple :

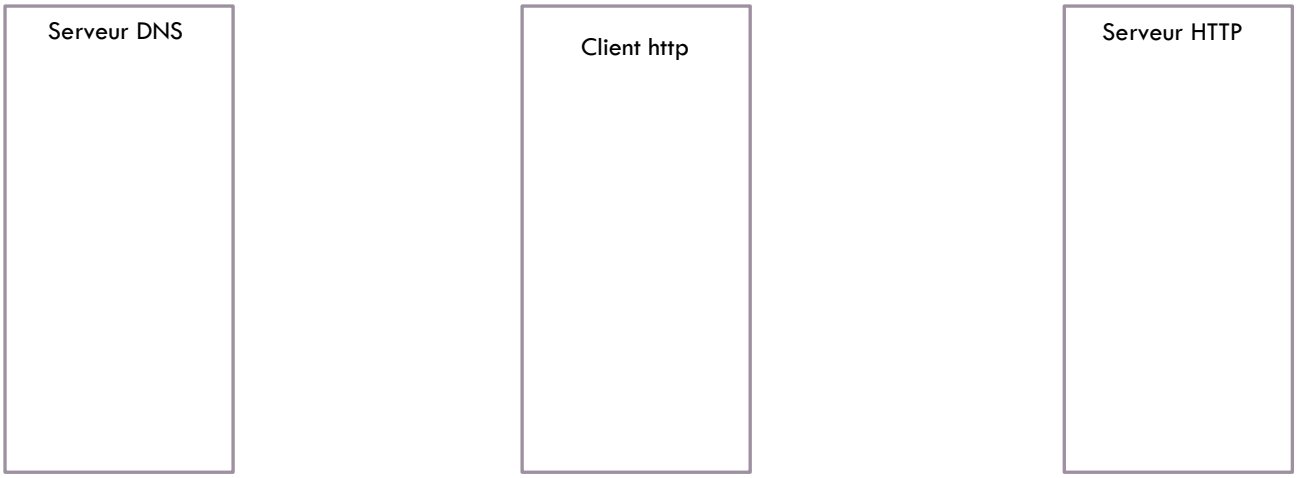
- Le client formule des demandes ou **requêtes** au serveur.
- Le serveur formule des **réponses** aux demandes des clients.

Le protocole http est basé sur ce modèle ou architecture Client – Serveur.

- Le client http cela peut être le NAVIGATEUR → Un logiciel
- Le serveur http, c'est par exemple un serveur Apache, Nginx, IIS, autres. → Un logiciel aussi

Pour qu'un **client HTTP** puisse dialoguer avec un **serveur HTTP**, les 2 doivent respecter les règles de conversations du protocole HTTP. C'est ce que nous allons étudier au travers de capture de trame, mais également en "parlant" ce langage

A faire : ouvrir le fichier « trameHTTP.pcap » dans le logiciel WireShark et compléter les échanges de trames dans le schéma suivant :



En résumé nous avons la chaine suivante : DNS → TCP/IP → HTTP

LE PROTOCOLE DNS

DNS (Domain Name System) : service qui traduit des noms lisibles par l'humain (noms de domaines, machines ou services) en adresses réseau compréhensibles par les ordinateurs.

C'est un protocole qui utilise UDP (User Datagram Protocol) sur le port 53. C'est un protocole de niveau transport.

LE PROTOCOLE TCP /IP

C'est un protocole de niveau transport.

On distingue à ce niveau deux modes de communication :

- Le mode connecté (comparable à une communication téléphonique), utilisant le protocole TCP (Transmission Control Protocol). Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que l'adresse de destination n'est pas nécessaire à chaque envoi de données. Le Client ou le Serveur peut mettre fin à la connexion.
- Le mode non connecté ou Datagram (analogue à une communication par courrier), utilisant le protocole UDP (User Datagram Protcol). Ce mode nécessite l'adresse de destination à chaque envoi, et aucun accusé de réception n'est donné.

TCP signifie « Transmission Control Protocol ».

C'est ce protocole qui est utilisé pour établir la connexion entre le navigateur et le serveur Web.

Au niveau programmation, la connexion entre 2 machines se fait par l'utilisation d'une « socket ». Pour qu'une connexion puisse se faire entre 2 Logiciels/Applications il faut 2 informations essentiels :

- Un numéro de port (0-65535). Les numéros de port de 0 à 1024 sont réservés. Chaque protocole utilise un numéro de port d'écoute qui lui est propre :
  - HTTP:80, DNS:53(UDP), FTP:21, TELNET:23....
  - D'autres programmes qui ne sont pas des protocoles implémentent également le principe de socket. C'est le cas des serveurs SQL. Par exemple MySQL écoute sur le port 3306.
- Une adresse IP.

LE PARE FEU

Son rôle est de bloquer des numéros de port (0-65535), ou bien une IP, ceci pour la sécurité d'une machine.

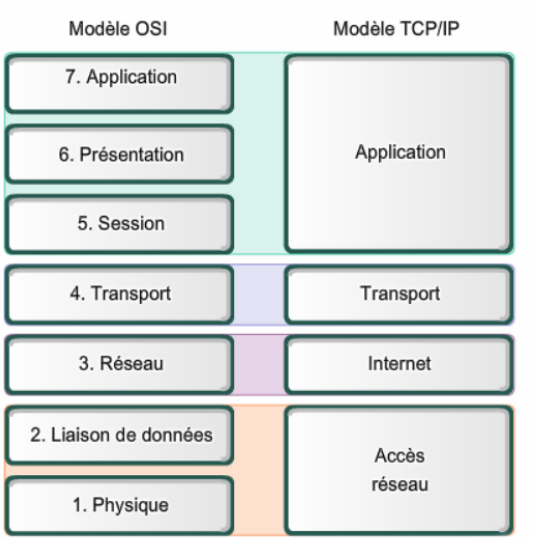
LE PROTOCOLE HTTP

Qu'est-ce qu'un protocole ? Un protocole **est un ensemble de règles qui définissent comment se produit une communication dans un réseau.**

Celui de http est définie dans une RFC :https://datatracker.ietf.org/doc/html/rfc2616

Pour plus d'information sur RFC : https://fr.wikipedia.org/wiki/Request_for_comments

A faire : Associer chacun de ses protocoles à un niveau du modèle OSI : FTP, TCP, UDP, Telnet, HTTP, IP, Ethernet, DNS ...



A faire : si WireSkark est installé, capturer une trame http à partir de votre navigateur sur le site http://b1.sgibert.net.local

LES DIFFERENTS MODES

MODE FICHIER LOCAL

A faire : dans le lecteur H : ou à un autre endroit créer un dossier TP1-Web ➔ ce sera la racine (/) de votre site web.

A faire : lancer VsCode. Choisir Open Folder, ouvrir le dossier TP1-Web. Ajouter un fichier nommer **« index.html »**

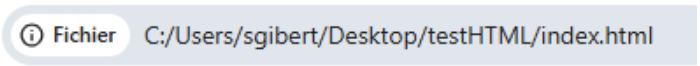
A faire : VSCode utilise des outils d'auto-complétions comme Emmet (antif à vscode) pour ajouter du code automatiquement. Taper **« html »** puis choisir **« html : 5 »**, ou bien simplement **« ! »**

Bien d'autres raccourcis de code sont possibles, voir par exemple ce lien : https://laconsole.dev/blog/raccourcis-emmet-indispensables

Attention : avant de vouloir développer rapidement, il faut d'abord savoir coder simplement !

A faire : ajouter à la balise <body> la balise <h1> Hello World ! </h1>

A faire : dans la zone terminale taper : **« start index.html »**, votre page s'ouvre dans votre navigateur, avec une URL commençant par C:\....



Ce qu'il faut retenir de ce mode :

- Le navigateur ouvre le fichier **directement depuis le disque**
- **Il n'y a pas de serveur web.**
- Le protocole http n'est pas utilisé, donc les protocoles DNS, TPC, UDP ne le sont pas eux aussi.

MODE LOCALHOST

A faire : dans le terminal taper la commande suivante :

```
TP1-WEB > python -m http.server
```

Nous venons de lancer un serveur Web très léger. Ce serveur écoute par défaut sur le port 8000 et sert les fichiers se trouvant dans le dossier courant de VSCode à savoir dans notre cas « **TP1-Web** »

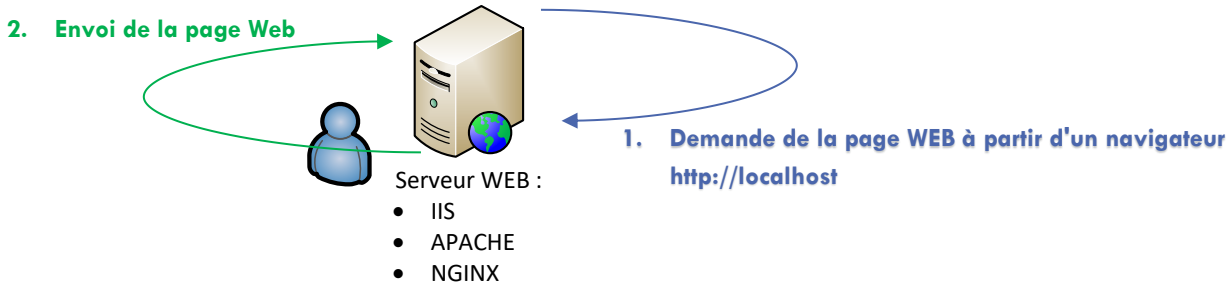
A faire : dans VSCode ouvrir un nouveau terminal et taper la commande :

```
start chrome http://localhost:8000
```

A faire : taper la commande :

```
start chrome http://127.0.0.1:8000
```

Dans ce mode votre machine est à la fois CLIENT et SERVEUR. Elle joue les 2 rôles. C'est très pratique pour un développement local sur votre poste.



Avantages :

- Ce type d'installation est pratique, car elle simule au mieux les conditions réelles d'un serveur d'hébergement Web.
- Ce type d'installation permet de travailler sans que l'on dispose d'un réseau local, ou même d'une connexion Internet.

Inconvénients :

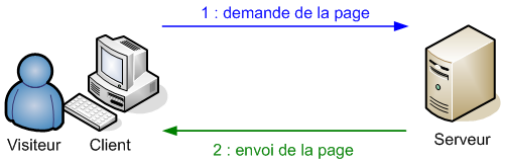
- Une fois développé en local il faut transférer votre site vers votre hébergeur via un client FTP. Se pose alors le problème des versions (celles en local et celles hébergées)
- Bien souvent plusieurs personnes travaillent en même temps sur le même site. Dans ce cas, la solution locale n'est plus possible.

MODE CLIENT-SERVEUR

Avec la commande

```
python -m http.server
```

Votre serveur écoute à la fois en local mais aussi à distance.



A faire en binôme : noter l'adresse IP du site de votre binôme et taper l'URL suivante : http://@IP_SERVEUR_DU_BINOME:8000

Dans votre navigateur.

A faire sur le serveur et sur le client : à l'aide de la commande suivante repérer les ports en écoute (LISTENNING) et les ports connectés (ESTABLISHED).

```
netstat -a -n
```

ETUDE DU PROTOCOLE HTTP

Comme nous l'avons vu le protocole http est basé sur l'architecture Client-Serveur :

- Le client formule des demandes ou **requêtes** au serveur.
- Le serveur formule des **réponses** aux demandes des clients.

LA REQUETE :

Une fois la connexion établie entre le client et le serveur, le client émet une requête HTTP composé de 3 parties distinctes :

- **Ligne de requête** constituée de 3 éléments Nom_méthode URL Version_http
- **L'en-tête** optionnel communique des informations complémentaires à la requête. Elles sont aux nombres de 16 et ont la forme suivante :

  Nom_en_tête: valeur

- **Le corps** de la requête : il n'est là que lorsque le client envoie des données au serveur en utilisant la méthode POST par le biais de formulaire dans la majorité des cas.

A faire : noter les **lignes de requêtes** envoyées par le client au serveur. Quel est le nom de la méthode ? Son URL ? La version du protocole http utilisé. On retrouve ses informations dans le terminal de VSCode.

A faire : pour voir en détail une requête http, nous allons utiliser la commande suivante. Repérer la ligne de requête, l'en-tête de la requête, le corps de la requête.

```
curl -v http://@IP_DU_SERVEUR_DU_BINOME:8000
```

A faire : quels sont les méthodes du protocoles http ? Expliquer en une phrase chacune des méthodes

A faire : que signifie http ?

LA REPONSE

À partir de HTTP/1.0, la réponse du serveur a le même format qu'une requête, en se composant en 3 parties :

- **La ligne statut de la réponse,** qui sert à préciser la manière dont s'est passé le traitement de la requête. En fonction du déroulement de la requête, le code statut vaut :

2xx	Requête reçue, traitée
3xx	Redirection sur une autre URL, traitement incomplet
4xx	Erreur client
5xx	Erreur serveur

- **L'en-tête de lé réponse,** qui contient des informations sur la réponse et sur le serveur du type :

Nom_en_tête: valeur

- **Le corps de la réponse** : correspond au code HTML de la page demandé. C'est ce qui sera interprété par le navigateur. Vous notez que ce n'est que du texte qui circule

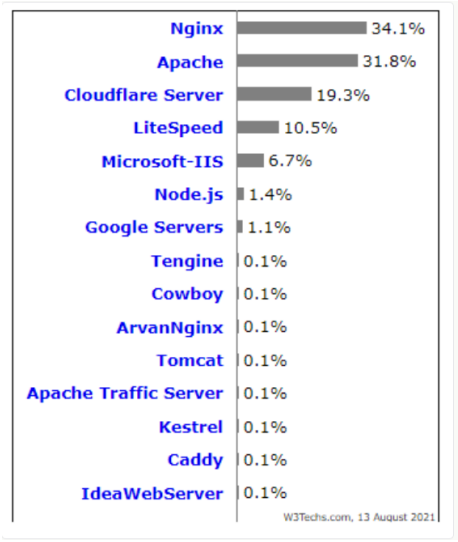
A faire : dans le terminal, noter les statuts des réponses envoyés au CLIENT. Des requêtes sont en statuts 200, expliquez. Une autre requête est en statut 404. Expliquer l'erreur 404 ? Pourquoi cette erreur ? Comment la corriger ?

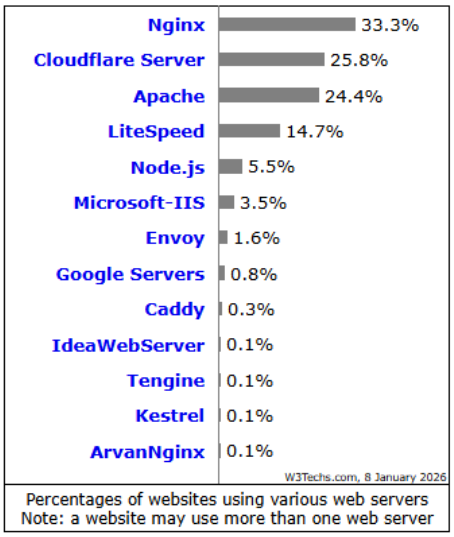
ETUDE DES FONCTIONNALITES D'UN SERVEUR WEB

Dans le cadre du module B1 Programmation Web nous allons apprendre les bases du développement WEB. Plus précisément, nous allons produire des pages Web. Ces pages en elle-même ne servent pas à grand-chose si elles ne sont pas hébergées sur un serveur WEB.

À quoi sert un serveur WEB ? Quelles sont les fonctionnalités et options proposées ?

Il existe de nombreux serveur WEB. Voici une liste non exhaustive avec un pourcentage d'utilisation parfois non identique selon les sources, ici en copie d'écran les chiffres entre janvier 2026 et Aout 2021





Tous ces serveurs présentes plus ou moins les mêmes fonctionnalités :

- **Écoute réseau**
  - Écoute sur un ou plusieurs **ports** (80, 443, autres)
  - Liaison à une ou plusieurs **adresses IP**
- **Service HTTPS**
  - Gestion des **certificats TLS**
  - Chiffrement des communications
  - Redirection HTTP ⟶ HTTPS
- **Gestion des ressources**
  - **Répertoire racine (DocumentRoot)**
    - Dossier servant de base aux ressources web
    - Correspondance URL ↔ fichiers
  - **Documents par défaut**
    - index.html, index.php, etc.
    - Servis automatiquement sans préciser le nom
- **Types MIME**
  - Association extension ⟶ type (text/html, image/png, etc.)
  - En-tête Content-Type
- **Exécution de scripts :** PHP, Python, ASP.NET, etc.
- **Contrôle d'accès**
  - restrictions par :IP, utilisateur, authentification (Basic, Digest, etc.)
- **Droits sur les ressources :** lecture / écriture / exécution /protection de dossiers
- **Limitation :** nombre de connexions / taille des requêtes / débit
- **Journaux d'accès :** IP client, ressource demandée, code statut http, date / heure
- **Journaux d'erreurs :** erreurs serveur / scripts en échec /problèmes de configuration
- **Performances**
  - **Cache :** cache côté serveur, contrôle via en-têtes http
  - **Compression :** gzip / brotli, réduction du volume transféré
  - **Gestion des connexions :** keep-alive, files d'attente, threads / workers

- **Redirections et réécritures Redirections :** 301, 302, changement d'URL
- **Réécriture d'URL :** URLs « propres », masquage de la structure interne
- **Équilibrage de charge :** répartition du trafic, haute disponibilité
- **Administration et supervision :**
  - **Configuration :** fichiers texte, interface graphique
  - **Démarrage / arrêt :** service système
  - **Supervision :** état du serveur, statistiques, protection contre les attaques (rate limit, headers sécurité)

Avec notre « SimpleHTTP/0.6 Python » nous allons voir quelques fonctionnalités :

1. **Document par défaut : le fichier « index.html » est servi automatiquement par le serveur. Remarquer que lorsque vous êtes sur votre navigateur la page index.html est affiché et cela sans préciser le fichier dans l'URL (http://@IP:8000)
   A faire : renommer votre page **« index.html »** en **« index1.html »**, et actualiser votre page, que ce passe-t-il ? Pourquoi ? Comment faire pour afficher **« index1.html »** ?
2. **Eploration de répertoire :** cette fonctionnalité est associée au Document par défaut. En l'absence de ce dernier, le serveur expose l'ensemble des fichiers qui sont à la racine du site. A noter que cette option peut être bloqué par de « vrai » serveur web pour des raisons de sécurité.
3. **Modifier le Port d'écoute :**

A faire : arrêter votre serveur et relancer le avec la commande « python -m http.server 8080 ». Le serveur est maintenant en écoute sur un port différent. Que faut-il faire pour que la page s'affiche correctement ? Que ce passe-t-il si on choisit un numéro de port comme « 5432 » ? Est-ce que cela fonctionne toujours ? Pourquoi ?

A faire : lancer 2 serveurs python sur 2 ports différents, est-ce possible ? Quel intérêt ?

A faire : remettre le port par défaut (le 8000) pour votre serveur.

4. **Type Mime :** le type Mime est automatiquement reconnu comme dans tous les serveurs.
   A faire : ajouter une page test.css avec le code suivant :

```
body{
    background-color: #f3b8b8;
}
```

➔Tester la commande « curl -v http://localhost:8000/test.css ». Que vos l'en-tête de réponse « content-type » ?

➔ Faire de même avec différents fichiers selon les extensions suivante : .txt, .png, .jpg,.gif, .js, .docx, etc …

5. **La journalisation :** elle reste sommaire, mais vous connaissez les requêtes reçus par votre serveur, et les réponses envoyés. Exemple : Observer et relever toutes les informations pertinentes, notamment lorsque la requête est distante ?

```
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::1 - - [08/Jan/2026 09:17:04] "GET /test.css HTTP/1.1" 200 -
::1 - - [08/Jan/2026 09:18:32] "GET /index1.html HTTP/1.1" 304 -
::1 - - [08/Jan/2026 09:19:23] code 404, message File not found
::1 - - [08/Jan/2026 09:19:23] "GET /index1.html HTTP/1.1" 404 -
::1 - - [08/Jan/2026 09:19:30] "GET /index.html HTTP/1.1" 200 -
::1 - - [08/Jan/2026 09:19:32] "GET /index.html HTTP/1.1" 304 -
::1 - - [08/Jan/2026 09:19:59] "GET /index.html HTTP/1.1" 200 -
::1 - - [08/Jan/2026 09:19:59] "GET /test.css HTTP/1.1" 200 -
::1 - - [08/Jan/2026 09:24:18] "GET /a.png HTTP/1.1" 200 -
::1 - - [08/Jan/2026 09:24:36] "GET /a.docx HTTP/1.1" 200
```