Le protocole HTTP

URI vs URL?

https://google.com

- URI ou URL?

URI

- *Uniform Resource Identifier*
- Un URI identifie une ressource par nom, emplacement, ou les deux
- Un URI peut être un nom par lui-même, ou un nom avec un protocole
- Exemples :
  - `google.com`
  - `https://google.com`
  - `urn:isbn:0-486-27557-4`

URL

- *Uniform Resource Locator*
- Un URL est un sous-type d'URI qui inclut un protocole, comme FTP ou HTTP
- Exemples :
  - `https://google.com`
  - `http://machine.local/fichiers/un_fichier.txt`
  - `ftp://ftp.mozilla.org`
  - `mailto:`
  - `file:///home/user/fichier`

URN

- *Uniform Resource Name*
- Un URN est un sous-type d'URI qui utilise le schéma `urn:`
- Exemples :
  - livre : `urn:isbn:0-486-27557-4`
  - RFC (*Request For Comments*) : `urn:ietf:rfc:791`
  - UUID (*Universally Unique Identifier*) : `urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66`
  - Espace de nom (*namespace*, en XML par exemple) : `urn:example:animal:ferret:nose`

Donc...

- Les URI rassemblent les URN et les URL
- Ainsi, **tous les URL sont des URI** (mais pas l'inverse)
- URL et URI sont des moyens de trouver quelque chose en ligne (si on exlut les URN)
- Un URI peut être un nom par lui-même, ou un nom avec un protocole
- Un URL doit inclure un protocole
- Quand on vous parle d'un URI en informatique, il y a de fortes chances qu'on parle d'un URL

Structure d’un URL

```
https://p.gah:Xud876543@www.exemple.com:8082/chemin/encore/index.php?search=to
```

Structure d’un URL (1)

- `https` : *scheme*, protocole utilisé
- `p.gah:Xud876543` : *userinfo*, nom d’utilisateur et mot de passe
- `www.exemple.com` : *host*, nom du domaine et sous-domaines (ou bien IP)
- `8082` : *port*
  - *userinfo* + *host* + *port* = *authority*, autorité (cible)

Structure d’un URL (2)

- `/chemin/vers/index.php` : *path*, chemin sur le serveur vers la ressource demandée
- `param1=categorie3&param2=4` : *query* ou *query string*, paramètres passés à l’application web
  - `?` sépare le *path* du *query*
  - `&` sépare les paramètres
  - `=` sépare le nom du paramètre de sa valeur
- `section2` : *fragment*, cible à atteindre dans la page (ancre en HTML)

Protocoles et ports par défaut

- HTTP (*HyperText Transfer Protocol*) : 80
- HTTPS (*HTTP Secure*) : 443
- FTP (*File Transfer Protocol*) : 20/21
- SSH (*Secure Shell*) : 22
- SFTP (*Secure FTP* via SSH) : 22
- FTPS (*FTP Secure* via SSL/TLS) : 989/990
- Telnet : 23
- MAILTO : pas de port (pour client messagerie)
- FILE : pas de port (pour fichiers locaux)
- SMB (*Server Message Block*) : 445
- IRC (*Internet Relay Chat*) : 6667
- ...

HTTP

HTTP ?

- *HyperText Transfer Protocol*
  - texte qui contient des liens vers d’autres textes
  - toute la navigation sur le Web est basée sur ce concept
- Créé par Tim Berners-Lee, CERN, 1989, avec HTML et le concept d’URI/URL
  - *HTML* : structure d’une page web
  - *HTTP* : comment la page est transmise du serveur au client
  - *URL* : comment une ressource doit être adressée sur le Web

Rôle de HTTP

- HTTP est le « langage » dans lequel votre navigateur Web (notamment) parle au serveur Web
- Il s’agit plus précisément d’un **protocole de communication client-serveur**
  - le client est le navigateur, le serveur est le serveur Web

Fonctionnement de HTTP (1)

- Navigateur : « http://exemple.com »
- **Requête HTTP** vers le serveur
  - requête = ***header***, ou **en-tête HTTP** + ***body***, ou **corps de la requête**
  - *header* : méthode HTTP (GET, POST…), URI, version HTTP, *User-Agent*, cookies…

Fonctionnement de HTTP (2)

- Serveur reçoit la requête et la traite
  - peut impliquer, en fonction de la ressource demandée, des calculs, des requêtes à BDD, des accès à des fichiers, à d'autres serveurs/services…
- Serveur construit une **réponse HTTP** et la renvoie au client
  - réponse = ***header*** + ***body***
  - *header* : code de statut, type de contenu, cookies…
  - *body* : HTML, fichier, JSON…
- Client reçoit la réponse et l'utilise (par exemple, affiche le HTML)

Exemple de requête HTTP

```
GET /search?q=exemple+recherche HTTP/1.1
Host: www.lesite.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTM
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
Accept-Language: fr-FR,fr;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Exemple de réponse HTTP

```
HTTP/1.1 200 OK
Date: Thu, 22 Feb 2025 00:50:03 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: text/html; charset=UTF-8
Content-Length: 483
Connection: keep-alive

<!DOCTYPE html>
<html lang="fr">
<head>
...
```

HTTP - Utilisation

- Origine : demander un document HTML
- Plus généralement : tout type de média
- *Streaming* (audio, vidéo)
- Applications
- WebSockets (communication bidirectionnelle sur le web)
- Web services (SOAP, XML-RPC, JSON-RPC)
- API REST (contrôle de services web)

HTTP - Architecture Client-Serveur

- Client : navigateur, application mobile, application *desktop*…
- Le *software* destiné au client et au serveur doivent être indépendants
  - au sens où ils peuvent être développés et déployés séparément

HTTP - Protocole *Stateless*

- HTTP = protocole *stateless* (sans état)
- Chaque requête est traitée indépendamment des autres
  - le serveur ne garde pas en mémoire les requêtes précédentes
- C'est pourquoi on a besoin de stocker des informations côté client et côté serveur pour garder un état
  - côté client : cookies, *localStorage*, *sessionStorage*...
  - côté serveur : gestion de **session client**