Chapitre XIII : Le Web côté serveur

I - Requête HTTP

a) Principe de l'échange client-serveur

Tout accès à un site Web commence par la saisie d'une URL sous la forme :

$$protocole\ ://nom\_ou\_adresse/document$$

où *protocole* peut être http ou https (lorsqu'on consulte une page Web). La partie *nom_ou_adresse* est le nom de domaine ou l'adresse IP de la machine faisant office de serveur (dans le cas d'un nom de domaine, un serveur DNS donnera l'adresse IP du serveur).
Le *document* est optionnel, ce peut être le nom d'un fichier stocké sur le serveur, en cas d'absence, ce sera le fichier index.html.

Par exemple si on saisit l'URL `https://icnlauregatet.github.io/1NSI/index.html`, les actions suivantes se produisent :

1. Le navigateur isole le nom de domaine `icnlauregatet.github.io` et effectue une requête DNS pour obtenir l'adresse IP de serveur Web hébergeant le site (185.199.111.153 dans cet exemple).
2. Le navigateur se connecte alors à la machine d'adresse IP 185.199.111.153 en utilisant le port TCP.
3. Une fois connecté à la machine, le navigateur envoie un message avec le protocole HTTP pour demander la ressource index.html du répertoire 1NSI.
4. Le serveur Web envoie en réponse le contenu du fichier au navigateur.
5. Le navigateur parcourt ensuite le fichier et envoie une nouvelle requête au serveur Web à chaque fois qu'il identifie un élément nouveau. Le serveur les lui renvoie au fur et à mesure.

Le protocole HTTP définit les messages envoyés entre le navigateur (nommé *client*) et le serveur Web. Les messages envoyés par le client sont appelés des **requêtes**, ceux envoyés par le serveur sur appelés des **réponses**. La majorité des requêtes d'un client sont des demandes d'une ressource (fichier HTML, PDF, image, ...). Par exemple, en écrivant l'URL `https://icnlauregatet.github.io/1NSI/index.html`, le navigateur Web envoie au serveur la requête :

```
Méthode de la requête :GET
Version :HTTP/2
URL de la requête :https://icnlauregatet.github.io/1NSI/index.html
```

Le navigateur (client) souhaite communiquer avec la version 2 du protocole HTTP et lui demande la ressource /1NSI/index.html. Il le fait avec la méthode GET, qui permet uniquement de demander des informations au serveur. La méthode POST permet également d'envoyer au serveur des informations, ce qui est souvent le cas lors d'une réponse à un formulaire.

Le serveur lui répond :

```
HTTP/2 200 OK
server: GitHub.com
content-type: text/html; charset=utf-8
content-length: 6229
last-modified: Sat, 23 May 2020 12:42:02 GMT
date: Sat, 23 May 2020 20:34:46 GMT
```

Le serveur informe le client qu'il accepte de communiquer avec la version 2 du protocole HTTP et que la ressource demandée est disponible et peut ainsi entre envoyée (le code 200 traduit le succès de la requête - le code étant 304 si le fichier se situe déjà dans le cache du navigateur).
On voit également quand a été effectuée cette demande et de quand date la dernière version de la page demandée. Cette réponse précise également la taille du fichier (6229 octets).

Ces réponses permettent au navigateur de gérer la réponse pour l'afficher en étant certain d'avoir reçu l'intégralité du fichier.

Remarque : Lorsqu'on demande une page qui n'existe pas comme :
`https://icnlauregatet.github.io/1NSI/inexistant.html`, alors le serveur répond :

```
HTTP/2 404 Not Found
date: Sat, 23 May 2020 20:44:31 GMT
content-length: 5232
```

Le code 404 signifie que le serveur n'a pas trouvé la ressource demandée.

Échange sécurisé avec HTTPS

Le protocole HTTP est un protocole en clair : n'importe qui ayant des droits suffisants pour intercepter des paquets échangés sur le réseau pourra lire le contenu des messages (par exemple un administrateur sur l'une des machines se trouvant entre le client et le serveur). Ceci pose des problèmes en termes de sécurité et de confidentialité des données, en particulier si l'utilisateur renseigne un formulaire avec des données sensibles comme un mot de passe. Pour y remédier, le protocole HTTPS a été introduit : les informations échangées sont alors chiffrées, un utilisateur interceptant les paquets ne pourra alors n'extraire qu'une suite d'octets.

b) Programmation réseau en python

Le module `socket` en python permet de programmer des applications client-serveur.
Par exemple la fonction `gethostbyname` prend en argument un nom de domaine et renvoie son adresse IP. Par exemple :

```
>>> from socket import socket, gethostbyname
>>> gethostbyname("icnlauregatet.github.io")
'185.199.110.153'
>>> gethostbyname("www.qwant.fr")
'217.70.184.56'
```

Une *socket* est une interface qui représente la connexion entre notre machine et une machine distante sur le réseau. Par défaut, une socket utilise les protocoles IPv4 et TCP. Nous allons les utiliser pour programmer un client d'un côté et un serveur de l'autre. Dans l'exemple ci-dessous, le serveur attend que le client lui envoie un message (sous la forme d'une chaîne de caractères). Le client servira juste à envoyer une phrase d'un utilisateur au serveur.

Programme côté serveur TCP :

```
from socket import socket

serveur = socket()
serveur.bind( ('', 12345) )
serveur.listen(5)

while True:
    connexion, adresse = serveur.accept()
    print("connexion de", adresse)
    donnees = connexion.recv(1024)
    while donnees.decode() != "Fin" :
        print(donnees.decode(), end='')
        donnees = connexion.recv(1024)
    connexion.close()
```

Programme côté client :

```
from socket import socket

serveur = socket()
serveur.connect( ('127.0.0.1', 12345) )

phrase = input("Saisis un texte : ")
while phrase != "Fin":
    phrase = phrase + '\n'
    serveur.send(phrase.encode())
    phrase = input("Saisis un texte : ")
serveur.send(phrase.encode())
```

Pour fonctionner, il faut écrire ces deux programmes dans deux éditeurs différents (Edupython, Spyder, Thonny, ...).

L'idée est juste de comprendre le comportement d'un serveur à l'aide d'un programme assez simple. Côté serveur, on crée une socket et on place le serveur en position d'écoute. Quand on lance le programme, la boucle `while` place le serveur en mode actif, il est donc en attente d'une connexion.
Côté client, on lance le programme qui se connecte au serveur à l'adresse IP 127.0.0.1 (c'est l'adresse *localhost*, c'est-à-dire l'adresse locale de l'ordinateur) avec le même port que celui déclaré dans le serveur (12345). Le client demande à l'utilisateur de saisir un texte qui sera envoyé au serveur (le serveur écrit alors ce texte dans la console). Quand l'utilisateur saisit le mot `'Fin'`, le client envoie un dernier message au serveur lui

indiquant de fermer la connexion et le programme s'arrête. On peut relancer le programme et le serveur établira une nouvelle connexion.

Remarque 1 : Les données échangées entre un serveur et un client sont des octets. Les textes devront donc être encodés depuis le client pour être envoyés au serveur, puis décodés par le serveur pour être affichés correctement. Ceci explique la présence des méthodes encode et decode.

Remarque 2 : Dans un réseau local (LAN), comme celui qui est lié à une box, il est possible de lancer le programme sur une première machine dont il faudra connaitre l'adresse IP (une commande permet de l'obtenir facilement, retrouvez-là). Sur une deuxième machine, ouvrir le programme côté client et modifier l'adresse '127.0.0.1' par l'adresse IP du serveur. Le client pourra alors envoyer des messages au serveur (son adresse IP apparaîtra côté serveur).

c) URL complète

Nous n'avons parlé que d'URL relativement simples où la cible était systématiquement un document. Or certains formulaires nous permettent d'avoir des URL plus précises.
Par exemple, sur le site `http://revue.sesamath.net`, il est possible de remplir le champ "Rechercher" en haut à droite. Si on saisit "python", alors nous sommes redirigés vers une nouvelle URL :
`http://revue.sesamath.net/spip.php?page=recherche&recherche=python`.
Cette URL est sous la forme *protocole* ://*nom_ ou_ adresse*/*document* ?*n1*=*v1*&*n2*=*v2*.

Le document dans notre exemple est un fichier php (nous en parlerons plus tard), il contient les variables $page et $recherche (en php, les variables sont toujours précédées du symbole "$" - le symbole & permet d'en placer plusieurs - dont les valeurs respectives sont "recherche" et "python". Pour séparer le nom du document des différentes variables, on utilise le symbole " ?".

Cette adresse peut alors être envoyée sous la forme complète, les arguments de la recherche étant directement dans l'adresse. La preuve en cliquant sur le lien :
http://revue.sesamath.net/spip.php?page=recherche&recherche=python.
Nous verrons dans la deuxième partie comment on crée cette URL à l'aide d'un formulaire.