Python : programmer des Websockets

Présentation

Les Websockets permettent, dans un réseau, de gérer en temps réel la communication entre un serveur et un client :

- la communication s'effectue dans les deux sens (full-duplex) avec une connexion TCP unique :
  - le client peut envoyer des messages au serveur,
  - le serveur peut envoyer des messages au client.
- De cette manière, le client n'est pas obligé d'interroger le serveur pour obtenir des mises à jour d'information ou du nouveau contenu. Il est écoute simplement les nouveaux messages provenant du serveur.

Ressources

https://websockets.readthedocs.io/en/stable/intro.html

Installation de la bibliotèque Websockets

```
$ pip3 install websockets
```

Création d'un serveur

```python
#!/usr/bin/env python

import asyncio
import websockets

async def hello(websocket, path):
    # le serveur est en attente du message d'un client qui se connecte
    message_recu_du_client = await websocket.recv()
    print(message_recu_du_client)

    # le serveur envoie un message au lient qui s'est connecté
    message_pour_le_client = "Message bien reçu"
    await websocket.send(message_pour_le_client)
    print(message_pour_le_client)

start_server = websockets.serve(hello, 'localhost', 8765)

loop = asyncio.get_event_loop()
loop.run_until_complete(start_server)
loop.run_forever()
```

Explications :

- l'instruction **websockets.serve()** créer et démarre le serveur, un **WebSocketServer**, qui écoute les connexions clientes
- **hello()** est le gestionnaire (**handler**) qui sera appelé à **chaque connexion cliente** ; cette fonction reçoit en paramètre :
  - une instance du protocole websocket
  - le chemin URI
- si un client se connecte, la **coroutine hello()** est appelée et dès que la fonction hello() est **terminée**, la connexion est **fermée (close)** en utilisant les intructions suivantes :
  - instruction **websocket.recv()** : le serveur attend un message du client
  - isntruction **websocket.send(message)** : le serveur envoie un message
  - puis la connexion de ce client est close
- la méthode ***runforever() permet de faire tourner le programme indéfiniment pour écouter d'éventuels clients. </WRAP> ===== Avoir des informations sur la communication ===== Pour avoir des informations sur le Websocket créé lors de la connexion d'un client : * websocket.remote_address fournit l'adresse du client sous la forme d'un tuple (host, port) (None - rien- si la connexion n'a pas encore été établie. * websocket.open : cette propriété est à vrai (True) quand la connexion est***

utilisable. ===== Envoyer ou bien recevoir des messages ===== * Pour envoyer des messages aux clients (à mettre dans une boucle): <code python> while True: … instructions pour générer le message … ou bien si on souhaite utiliser une fonction message = await fonctionQuiGenereLeMessage() await websocket.send(message) </code> * Pour recevoir des messages des clients (à mettre dans une boucle): <code python> while True: message = await websocket.recv() … instructions pour traiter le message … ou bien si on souhaite utiliser une fonction await fonctionQuiTraiteDuMessage(message) </code> Pour la version de Python >= 3.6 la réception des messages peut s'écrire de la manière suivante : <code python> async for message in websocket: … instructions … ou bien si on souhaite utiliser une fonction await fonctionQuiTraiteDuMessage(message) </code> ===== Envoyer ET recevoir des messages en parallèle ===== ===== Création d'un client Web ===== Une page Web peut contenir des instructions javascript et utiliser les websockets pour pourvoir communiquer avec le serveur. Voici un exemple de code à mettre dans une page HTML. <code html> <!DOCTYPE html> <html> <head> <title>Client utilisant un WebSocket</title> </head> <body> <script> var ws = new WebSocket("ws:adresssipduserveur:8765/"), ws.onmessage = function (event) { alert(event.data); }; </script> </body> </html> </code> ==== Retour au cours : Les instructions du langage Python ====

- Cours : Les instructions du langage Python