INF8007 – Languages de scripts
Serveurs

Antoine Lefebvre-Brossard

Hiver 2018

HTTP

- Protocole de base formant la base du Web
- Fonctionne par un protocole de requête et réponse
- Typiquement, un *client* va envoyer une requête à un *serveur* et celui-ci enverra une réponse en retour
- Par exemple, lorsque un site web est visité, une requête est envoyée au serveur, qui retourne le plus souvent des fichiers HTML, CSS et JavaScript

HTTP (suite)

- Lorsqu'un serveur roule sur un ordinateur (le votre ou celui d'une ferme de serveurs quelconque), il écoute sur un port, par défaut le port 80
- Mais n'importe quel port peut être utilisé, tant qu'aucun autre serveur ne l'utilise déjà
- Par exemple, lorsqu'un notebook Jupyter roule, il utilise le port 8888
- En général le port 8080 peut être utilisé comme serveur web alternatif

HTTP (suite)

Bien qu'il y a plusieurs types de requêtes, les plus utilisées de loin sont **GET** et **POST**

GET : Requête avec les paramètres dans l'appel (le URL). Va normalement être utilisé par convention pour une demande sans effet externe. Est moins sécuritaire puisqu'un URL peut être lu par-dessus l'épaule ou copié à des endroits non sécuritaire, par exemple un log.

POST : Requête avec les paramètres dans le corps. Est utilisé dans les cas où un changement externe veut être fait, par exemple la création d'un usager, l'ajout d'un article de blog ou l'achat d'un item. Est aussi utilisé dans les cas où la sécurité est importante, par exemple au login.

HTTP (suite)

Par exemple, un GET sur le site du cours va retourner :

```
HTTP/1.0 200 OK
Content-Length: 322
Content-type: text/html
Date: Tue, 09 Jan 2018 14:53:58 GMT
Last-Modified: Tue, 09 Jan 2018 14:49:29 GMT
Server SimpleHTTP/0.6 Python/3.6.4

<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="shortcut icon" href="http://www.polymtl.ca/profiles/porta
    <title>INF8007 - Languages de script</title>
  head>
  <body>
    <script src="/app.js"></script>
  </body>
</html>
```

Serveur Python

- En Python, le module de base pour rouler un serveur est `http.server`
  https://docs.python.org/3.6/library/http.server.html
- Celui-ci hérite du module `socketserver`
- De façon très simplifié, un *socket* est un noeud sur un réseau pour envoyer ou recevoir des données
- Dans notre cas, il faut simplement savoir qu'un socket est le port sur lequel roule le serveur

Exemple


```python
from http.server import HTTPServer, BaseHTTPRequestHandler
PORT = 5000
ADDRESS = "localhost"  # same as ""
server = HTTPServer((ADDRESS, PORT), BaseHTTPRequestHandler)
try:
    server.serve_forever()  # start server
except KeyboardInterrupt:
    server.socket.close()  # close socket when server is interrupted
```

Exemple (suite)

Le problème avec ce serveur est que si on essaie d'y faire une requête on aura

```
HTTP/1.0 501 Unsupported method ('GET')
Connection: close
Content-Length: 496
Content-Type: text/html;charset=utf-8
Date: Tue, 09 Jan 2018 15:36:50 GMT
Server: BaseHTTP/0.6 Python/3.6.4
```

puisqu'aucune méthode n'est implémenté pour le gestionnaire (*handler*) de base.

Handler

- Pour créer un handler, il faut créer un objet héritant de `BaseHTTPRequestHandler`
- Par exemple,

```
class MyHandler(BaseHTTPRequestHandler):
    # ...
```

- Les méthodes que l'on veut implémenter sont de la forme `do_`method`()`, par exemple :
  - `do_GET()`
  - `do_POST()`

Exemple


```
import mimetypes
import os
from http.server import HTTPServer, BaseHTTPRequestHandler

PORT = 8080

class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        # ...

def serve():
    # ...

if __name__ == "__main__":
    serve()
```

Exemple (suite)


```
def serve():
    server = HTTPServer(("", PORT), MyHandler)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        server.socket.serve()
```

Exemple (suite)


```
def on_GET(self):
    try:
        with open(os.path.join(os.path.dirname(__file__), self.path),
                  "r") as f:
            self.send_response(200)
            self.send_header("Content-type",
                             mimetypes.guess_type(self.path)[0])
            self.end_headers()
            self.wfile.write(f.read())
    except IOError:
       self.send_error(404, f"Page not found\n{self.path}")
```

Codes HTTP

- Les codes en HTTP sont séparés en 5 classes :
  - 100 Informatif
  - 200 Tout a fonctionné
  - 300 Redirection
  - 400 Erreur de la part du client
  - 500 Erreur de la part du serveur

Codes HTTP (suite)

- Les codes les plus utilisés sont

  200 OK Tout est correct

  400 Bad Request Lorsque la requête est mal formée

  401 Unauthorized Lorsque la requête nécessite une autorisation ou que celle-ci a été refusée

  403 Forbidden Lorsque le serveur indique que la requête est refusée

  404 Not Found Lorsque la ressource n'est pas trouvée

  418 I'm a teapot Il y a beaucoup de codes non-officiels de plus qui existent

Frameworks

- Il existe beaucoup de frameworks pour simplifier le développement d'un serveur web
- Les plus connus sont :

Django	Le plus utilisé et très gros
Flask	Second plus populaire. Beaucoup plus petit, mais facile à étendre
Tornado	Focus sur l'asynchronisme. Les Jupyter notebooks l'utilisent
Falcon	Moins populaire que les autres, mais extrêmement rapide et petit

- Il y en a plusieurs autres, tous avec leurs forces et faiblesses

Frameworks (suite)

- Bien que Django soit le plus populaire, il convient moins pour de petits projets que les autres
- Le TD3 vous permettra d'explorer l'utilisation du framework que vous voudrez
- Voici des liens vers les tutoriels de Flask et Falcon
  http://flask.pocoo.org/docs/0.12/tutorial/
  http://falcon.readthedocs.io/en/stable/user/tutorial.html#first-steps