Relazione dell'elaborato di Programmazione di Reti

Francesco Meloni

2025

1 Introduction

1.1 Obiettivo

Progettare un semplice server HTTP in Python (usando socket) e servire un sito web statico con HTML/CSS.

1.2 Requisiti minimi

1.2.1 Il server deve rispondere su localhost:8080

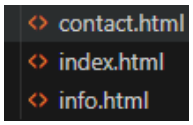
```
6   HOST = 'localhost'
7   PORT = 8080
8   BASE_DIR = './www'
9   s = socket(AF_INET, SOCK_STREAM)
10  s.bind((HOST, PORT))
11  s.listen(1)
12  print(f"Server in ascolto su http://{HOST}:{PORT}")
```

Creo una socket, eseguo il bind su localhost:8080, il server si mette in ascolto e definisco la coda di backlog pari ad 1 (dimensione della coda delle connessioni in entrata non ancora accettate)

```
64  while True:
65      connection, addr = s.accept()
66      HandleRequest(connection, addr)
```

Infine connetto un client creando un canale di comunicazione e mi occupo delle richieste.

1.2.2 Deve servire almeno 3 pagine HTML statiche



1.2.3 Gestione di richieste GET e risposta con codice 200

```python
def HandleRequest(connection, addr):
    request = connection.recv(1024).decode()
    if not request:
        connection.close()
        return

    lines = request.split('\r\n')
    method, path, _ = lines[0].split()

    if method != 'GET':
        connection.close()
        return
```

Ricevo la richiesta dal client, se e' nulla chiudo la connessione, se non lo e' guardo se e' una richiesta GET, se lo e' posso continuare.

```python
    if os.path.isfile(filePath):
        f = open(filePath, 'rb')
        content = f.read()
        mimeType, _ = mimetypes.guess_type(filePath)
        mimeType = mimeType or 'application/octet-stream'
        header = (
            "HTTP/1.1 200 OK\r\n"
            f"Content-Type: {mimeType}\r\n"
            f"Content-Length: {len(content)}\r\n"
            "Connection: close\r\n"
            "\r\n"
        )
        connection.sendall(header.encode() + content)
        LogRequest(addr, method, path, 200)
```

Se il percorso del file riporta ad un file esistente, allora, apro il file, leggo il contenuto, gestisco i MIME 1.3.1, scrivo l'header, spedisco la pagina a tutti i client ed eseguo il log della richiesta 1.3.2

1.2.4 Implementare risposta 404 per file inesistenti

```
52      else:
53          response = (
54              "HTTP/1.1 404 Not Found\r\n"
55              "Content-Type: text/html\r\n"
56              "\r\n"
57              "<h1>404 Not Found</h1>"
58          )
59          connection.sendall(response.encode())
60          LogRequest(addr, method, path, 404)
```

Nell'implementazione della risposta 404 si manda solo una risposta ai client per segnalare che non e' presente nessun file in quel percorso del file

1.3 Estensioni opzionali

1.3.1 Gestione dei MIME types (.html, .css, .jpg, ecc.)

La gestione dei MIME types avviene dopo aver letto il contenuto del file 1.2.3. Cerco di indovinare di che tipo e' il file (.html, .jpg, etc...), se non individuo nessun tipo imposto come tipo un file binario generico ("application/octet-stream"), infine inserisco il MIME nell'header della risposta.

1.3.2 Logging delle richieste

```
14  def LogRequest(addr, method, path, statusCode):
15      now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
16      logEntry = f"[{now}] {addr} {method} {path} -> {statusCode}"
17      print(logEntry)
18      with open("log.txt", "a") as logFile:
19          logFile.write(logEntry + "\n")
```

Il logging delle richieste viene effettuato gogni volta che si manda qualcosa ai client 1.2.3 1.2.4. La stringa di log viene scritta sia in console, sia in un file di testo log.txt, questa stringa comprende il datetime di quando si e' mandato qualcosa ai client, indirizzo ip e porta dei client, metodo (in questo caso sempre GET), il percorso del file richiesto e il codice di stato di HTTP (in questo caso 200 e 404)

1.3.3 Aggiunta di animazioni o layout responsive

Per le animazioni e il layout responsive e' stato usato un file .css