Relazione Tecnica - Giovanni Perreon

Web Server Statico in Python

Obiettivo del Progetto

Realizzare un web server minimale in Python, in grado di:

- Ricevere richieste HTTP (GET)
- Servire file statici da una cartella locale (www)
- Gestire correttamente richieste valide (200 OK) e errori (404 Not Found)
- Supportare diversi tipi MIME (html, css, immagini)

Tecnologie Utilizzate

- Linguaggio: Python 3.12.3
- Ambiente: Spyder 5.5.1
- Moduli principali:

  - **socket**: per la comunicazione TCP

  - **os**: per la gestione dei file e dei percorsi

  - **datetime**: per l'ottenimento delle date nel momento di risposta

Struttura del Progetto

- **server.py**: contiene tutto il codice del server
- **www/**: directory che contiene i file statici da servire:

  - **index.html**: pagina html statica, home del web server, carica delle immagini e reindirizza a tutte le altre pagine html inclusa una pagina non inclusa per mostrare l'errore 404.

  - **intro.html**: pagina di introduzione generale al progetto, spiega la struttura e il funzionamento del server HTTP e delle sue componenti.

server.html: pagina che spiega in dettaglio come il server legge le richieste e risponde con i file richiesti.

header.html: pagina che mostra un esempio di header HTTP in cui il client e il server comunicano.

Funzionamento del Server

1. Il server si mette in ascolto su localhost:8080

2. Quando riceve una connessione:

   - Legge la richiesta HTTP

```
request = connection.recv(1024).decode()
```

   - Se il path è /, serve il file index.html e controlla se il file richiesto esiste

```
if path == "/":
    path = "/index.html"
file_path = BASE_DIR + SEPARATOR + path.lstrip("/")

if os.path.isfile(file_path):
```

   - Se esiste: invia risposta 200 OK + contenuto

```
if os.path.isfile(file_path):
    f = open(file_path, "rb")
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    outputdata = f.read()
    mime_type = get_mime_type(file_path)
    headers = (
        f"HTTP/1.1 200 OK\r\n"
        f"Date: {timestamp}\r\n"
        f"Content-Type: {mime_type}\r\n"
        f"Content-Length: {len(outputdata)}\r\n"
        f"Connection: close\r\n"
        f"\r\n"
    )
```

```
    connection.send(headers.encode())
    print("Response to", ip, "at", timestamp)
    print(" 200 OK - File found:", file_path)
```

- Se non esiste: invia risposta 404 Not Found

```
else:
    outputdata = b"<h1>404 Not Found</h1>"
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    connection.send(bytes("HTTP/1.1 404 Not Found\r\n\r\n","UTF-8"))
    print("Response to", ip, "at", timestamp)
    print(" 404 Not Found - File missing:", file_path)
connection.send(outputdata)
```

3. Determina il tipo di contenuto (MIME) in base all'estensione del file

4. Chiude la connessione dopo la risposta

Gestione dei MIME Type

I tipi MIME supportati sono mappati in un dizionario Python:

MIME_TYPES = {

".html": "text/html",

".css": "text/css",

".jpg": "image/jpeg",

".png": "image/png",

".ico": "image/x-icon"

}

Questo permette di indicare al browser il tipo corretto di contenuto.

Estensioni Possibili

- Aggiunta di supporto per altri metodi HTTP oltre a GET
- Multithreading o async per gestire più client contemporaneamente
- Protezione da richieste pericolose