Traccia 1 – Web Server + Sito Web Statico (livello base/intermedio)

ELABORATO PROGRAMMAZIONE DI RETI

Cristian Qorri matr. 0001129476
Nicholas Guerra matr. 0001125129
Matteo Botteghi matr. 0001129907

Realizzazione di un Web Server minimale in Python e pubblicazione di un sito statico

Questo progetto ha lo scopo di realizzare un semplice **web server HTTP** locale utilizzando Python e il modulo **socket**. Il server è in grado di rispondere a richieste HTTP di tipo **GET** per servire file HTML e altri contenuti statici (immagini, CSS, ecc.) presenti in una directory locale.

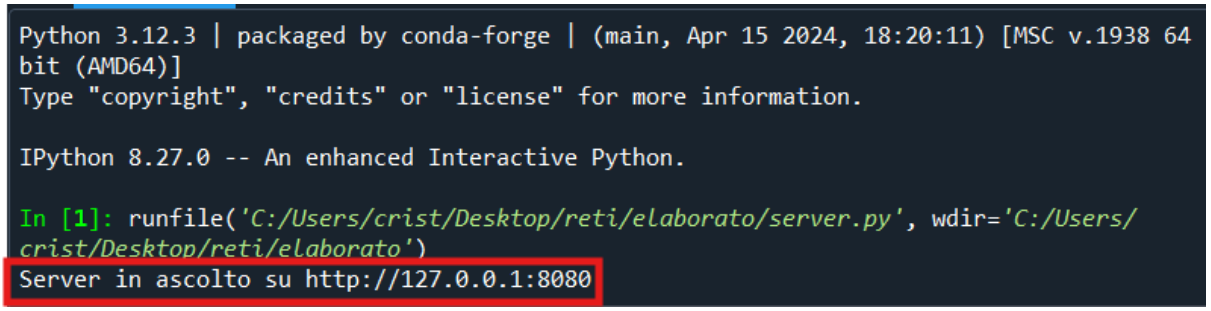
Nel file server.py, le funzionalità principali sono le seguenti:

- avvia un server TCP che ascolta sulla porta 8080 (HOST = 127.0.0.1);
- gestisce connessioni in entrata e analizza le richieste HTTP;
- restituisce file statici richiesti dai client (browser), come pagine HTML o immagini;
- mostra nel terminale un log con l'orario, il metodo HTTP, la risorsa richiesta e lo stato (200 o 404).

Se la richiesta è valida e il file esiste, invia il contenuto con intestazione **HTTP 200 OK**; nel caso in cui il file non esiste, viene invece inviata una pagina HTML con errore **404 NOT FOUND**.

Di seguito viene mostrato il funzionamento:

- avvio del server

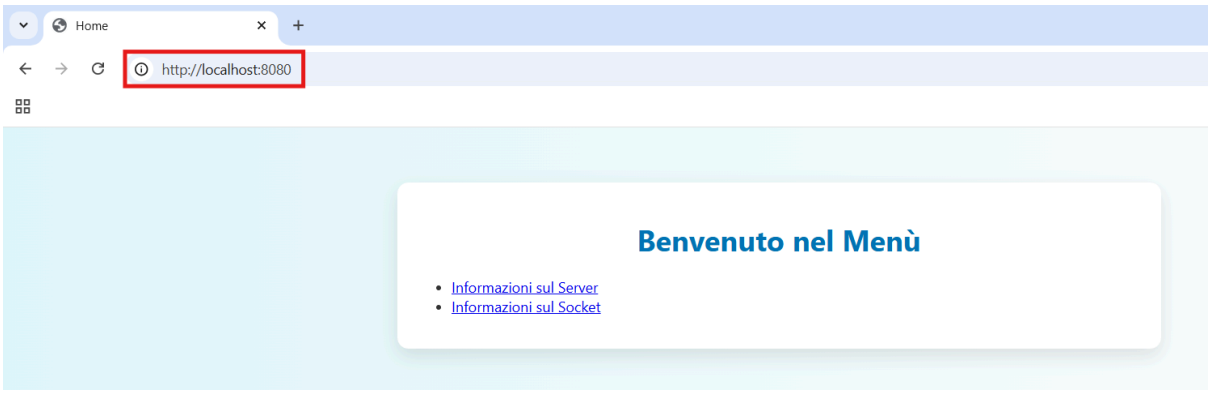


All'accensione del server viene creato un **socket TCP**, successivamente il socket viene associato a **localhost** sulla porta **8080**.
Il server inizia ad ascoltare connessioni in arrivo (listen(5)).
Successivamente viene stampato sul terminale:

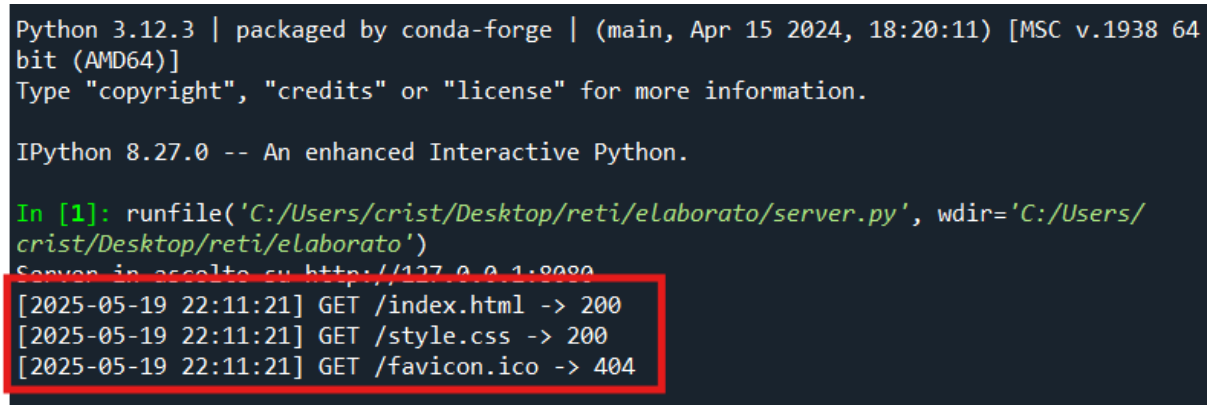
"*Server in ascolto su http://127.0.0.1:8080*"

N.b. → il file **index.html**, ossia nel nostro caso la *homepage*, è la pagina principale del sito, caricata quando si visita *http://127.0.0.1:8080/*.

- visita *http://localhost:8080* nel browser



Di seguito il risultato prodotto dal server python:



Dall'immagine possiamo notare che quando visito *http://localhost:8080* nel browser, il browser stesso invia istruzione:
GET /index.html e i relativi file che vengono richiesti.

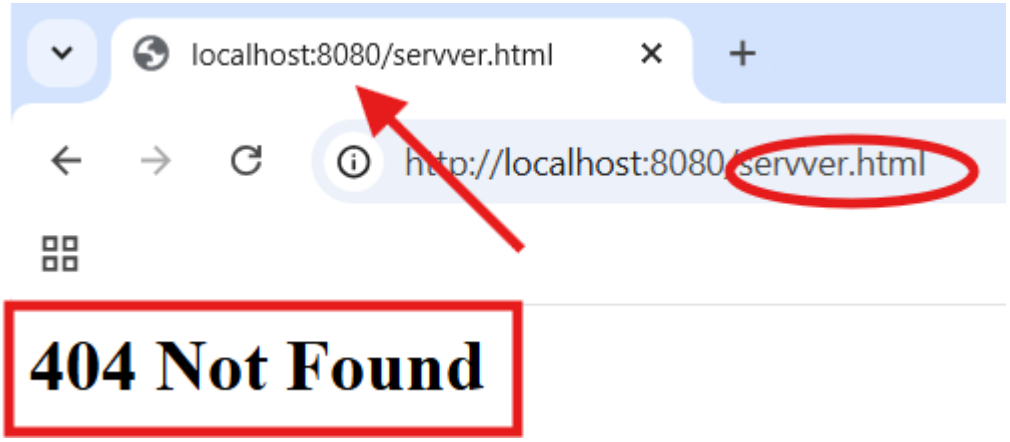
Se il file esiste:
- determina il tipo di MIME (es. text/html);
- costruisce una risposta **HTTP 200 OK**;
- invia il contenuto al browser.

Se il file non esiste invece invia una risposta **404 Not Found** con messaggio HTML.

Dopodiché il server ***chiude il socket client*** dopo aver risposto, per poi ritornare in ascolto per nuove connessioni.

Caso 404 Not Found

Mostriamo il caso di una pagina che non viene trovata, perché di fatto non esiste.
Per farlo modifichiamo il nome del file html, mettendolo appositamente sbagliato nell'url durante la ricerca nel browser.




```
In [1]: runfile('C:/Users/crist/Desktop/reti/elaborato/server.py', wdir='C:/Users/
crist/Desktop/reti/elaborato')
Server in ascolto su http://127.0.0.1:8080
[2025-05-19 22:11:21] GET /index.html -> 200
[2025-05-19 22:11:21] GET /style.css -> 200
[2025-05-19 22:11:21] GET /favicon.ico -> 404
[2025-05-19 22:21:23] GET /server.html -> 200
[2025-05-19 22:21:24] GET /style.css -> 200
[2025-05-19 22:21:24] GET /img/server.png -> 200
[2025-05-19 22:21:29] GET /servver.html -> 404
```

Ovviamente ci viene restituito un errore, in quanto la pagina "servver.html" non fa parte del nostro sito statico.

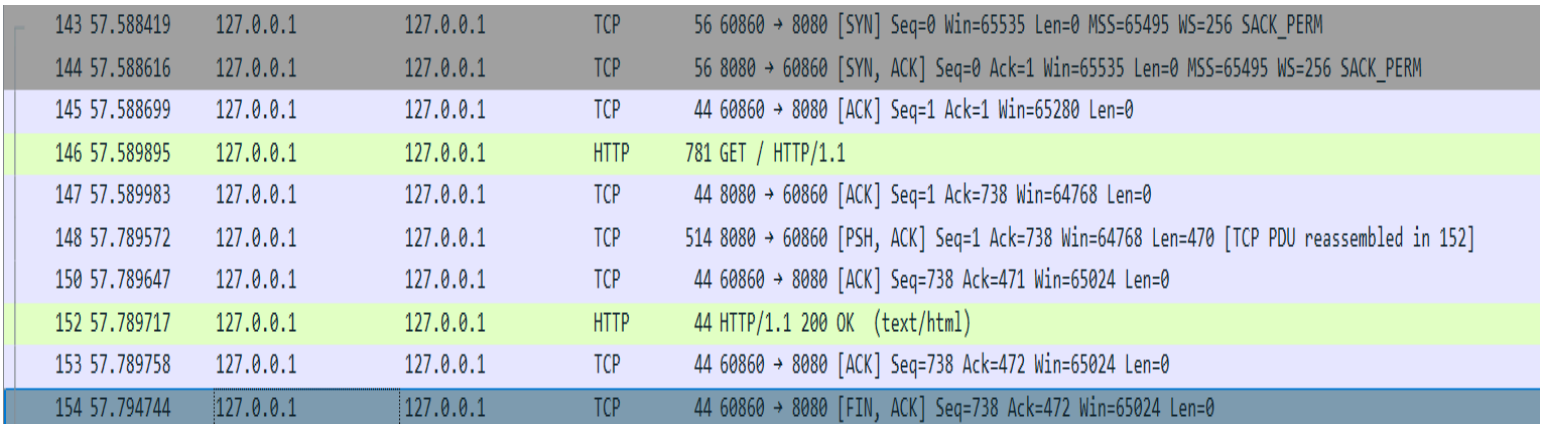
Monitoraggio del Traffico di Rete con Wireshark

Dopo aver avviato il web server (**server.py**), che si mette in ascolto sull'indirizzo **127.0.0.1** sulla porta **8080**, apro il browser e digito l'indirizzo *http://localhost:8080* per accedere alla home page servita localmente.

Analizzando il traffico con **Wireshark**, si osserva che il client (browser) avvia una connessione verso il server: 127.0.0.1:60860 si connette a 127.0.0.1:8080.
Il processo di **handshake TCP** si svolge correttamente, come evidenziato dai pacchetti alle righe **143-145**, che mostrano la sequenza classica: **SYN, SYN-ACK e ACK**.
Questo conferma che la connessione TCP è stata stabilita con successo.

Successivamente, nelle righe **146-153**, il browser invia una richiesta **HTTP GET**, che rappresenta la richiesta della pagina principale. Il **server risponde** correttamente alla riga 149 con un **HTTP/1.1 200 OK**, indicando che la risorsa è stata trovata e viene servita correttamente al client.
La sessione si completa con i pacchetti di tipo ACK e PSH/ACK per il trasferimento dei dati, seguiti dal **FIN, ACK alla riga 154**, che indica la **chiusura della connessione** da parte del **server**.

L'intero scambio dimostra il corretto funzionamento del server e la corretta gestione della connessione da parte del browser e del protocollo HTTP su TCP.



143	57.588419	127.0.0.1	127.0.0.1	TCP	56 60860 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
144	57.588616	127.0.0.1	127.0.0.1	TCP	56 8080 → 60860 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
145	57.588699	127.0.0.1	127.0.0.1	TCP	44 60860 → 8080 [ACK] Seq=1 Ack=1 Win=65280 Len=0
146	57.589895	127.0.0.1	127.0.0.1	HTTP	781 GET / HTTP/1.1
147	57.589983	127.0.0.1	127.0.0.1	TCP	44 8080 → 60860 [ACK] Seq=1 Ack=738 Win=64768 Len=0
148	57.789572	127.0.0.1	127.0.0.1	TCP	514 8080 → 60860 [PSH, ACK] Seq=1 Ack=738 Win=64768 Len=470 [TCP PDU reassembled in 152]
150	57.789647	127.0.0.1	127.0.0.1	TCP	44 60860 → 8080 [ACK] Seq=738 Ack=471 Win=65024 Len=0
152	57.789717	127.0.0.1	127.0.0.1	HTTP	44 HTTP/1.1 200 OK (text/html)
153	57.789758	127.0.0.1	127.0.0.1	TCP	44 60860 → 8080 [ACK] Seq=738 Ack=472 Win=65024 Len=0
154	57.794744	127.0.0.1	127.0.0.1	TCP	44 60860 → 8080 [FIN, ACK] Seq=738 Ack=472 Win=65024 Len=0

Tecnologie usate per la realizzazione dell'elaborato:

- Python (socket, mimetypes)
- HTML e CSS per il sito statico
- Spyder per realizzare server.py e visualizzare i risultati prodotti sul terminale
- Visual Studio Code per la realizzazione dei file HTML e CSS
- Wireshark per analisi del traffico di rete