Relazione Tecnica

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

Foschini Paolo

7 luglio 2025

1 Obiettivo

L'obiettivo del progetto è la realizzazione di un semplice server HTTP in Python, utilizzando il modulo `socket`, con lo scopo di servire un sito web statico composto da pagine HTML e file CSS.

1.1 Requisiti minimi

Il server doveva soddisfare i seguenti requisiti fondamentali:

- essere accessibile tramite `localhost` sulla porta `8080`;
- servire almeno tre pagine HTML statiche;
- supportare richieste `GET`, rispondendo con codice di stato `200 OK` in caso di successo;
- restituire un errore `404 Not Found` nel caso di richiesta di file non esistenti.

1.2 Requisiti opzionali

Oltre alle funzionalità di base, erano richiesti i seguenti requisiti:

- gestione dei principali MIME types (es. `.html`, `.css`, `.jpg`, ecc.);
- visualizzazione a terminale di un log delle richieste ricevute;
- supporto per layout responsive e animazioni CSS (funzionalità non implementata).

2 Funzionamento del server

Per la creazione del server HTTP ho utilizzato i moduli standard `http.server` e `socketserver` di Python.

Una volta avviato, il server si mette in ascolto sull'indirizzo `localhost` alla porta `8080`, in attesa di ricevere richieste da parte dei client. Alla ricezione di una richiesta HTTP di tipo `GET`, il server esegue i seguenti passaggi:

- Se l'URL richiesto è `/`, il server risponde con un codice `301 Moved Permanently` e reindirizza automaticamente il client verso `/www/index.html`;
- Per tutte le altre richieste, il server si comporta come un normale file server statico:
  - Se il file richiesto esiste, viene restituito con codice HTTP `200 OK` e il relativo contenuto;
  - Se il file non esiste, viene restituito un errore `404 Not Found`.

La gestione delle richieste è affidata a una sottoclasse (`CustomRequestHandler`), che estende il comportamento del modulo standard `SimpleHTTPRequestHandler` per aggiungere la redirezione dalla root a `index.html`.

Per consentire la gestione di più richieste contemporaneamente, il server utilizza `ThreadingTCPServer`, che crea un thread separato per ogni connessione. Questo permette di servire più client nello stesso momento in modo efficiente.

La gestione automatica dei MIME types (ad es. `.html`, `.css`, `.jpg`, ecc.) e la stampa a terminale del log delle richieste sono entrambe fornite da `SimpleHTTPRequestHandler`, integrato all'interno della classe personalizzata.

3 Funzionamento delle pagine HTML

Le pagine HTML servite dal server sono pagine statiche, strutturate in modo simile. In particolare:

- Ogni pagina include un foglio `style.css` esterno per l'aspetto grafico;
- È presente un'icona personalizzata (favicon) visualizzata nella scheda del browser;
- Ogni pagina contiene un'immagine in formato JPG;
- È incluso del testo con informazioni specifiche, diverse per ciascuna pagina;
- Sono presenti dei pulsanti per navigare facilmente tra le diverse pagine del sito;
- Tutti i file HTML, CSS e immagini vengono caricati tramite richieste HTTP `GET` gestite dal server.

4 Avvio e visualizzazione del server

Una volta avviato, il server rimane in ascolto su `localhost` alla porta `8080`. Ogni richiesta HTTP inviata a questo indirizzo viene gestita dal server.

4.1 Accesso tramite browser

Per visualizzare il sito web servito, è sufficiente aprire un browser e digitare nella barra degli indirizzi: http://localhost:8080.

Il server reindirizza automaticamente alla pagina iniziale `index.html` contenuta nella cartella `/www`. Da lì è possibile navigare tra le varie pagine del sito utilizzando i pulsanti.

4.2 Interruzione del server

Il server può essere arrestato in qualsiasi momento premendo la combinazione di tasti `Ctrl+C` nel terminale. Questo invia un segnale `SIGINT`, gestito nel codice dallo `signal handler`, che chiude correttamente il server e termina il programma.

Durante l'esecuzione, tutte le richieste ricevute dal server vengono stampate a terminale tramite il sistema di logging integrato in `SimpleHTTPRequestHandler`.

4.3 Esempi di richieste (vedi fig. 1)

- **200, OK**: la richiesta è stata elaborata con successo. Il server ha restituito la risorsa richiesta (file CSS, immagini JPG, pagine HTML);
- **301, Moved Permanently**: la risorsa richiesta è stata spostata permanentemente a un nuovo URL. Il browser dovrebbe automaticamente reindirizzare alla nuova posizione;
- **404, Not Found**: la risorsa richiesta non è stata trovata sul server. Questo errore compare, ad esempio, quando l'URL è sbagliato.

```
127.0.0.1 - - [02/Jul/2025 11:50:14] "GET / HTTP/1.1" 301 -
127.0.0.1 - - [02/Jul/2025 11:50:15] "GET /www/index.html HTTP/1.1" 200 -
127.0.0.1 - - [02/Jul/2025 11:50:15] "GET /www/style.css HTTP/1.1" 200 -
127.0.0.1 - - [02/Jul/2025 11:50:15] "GET /resources/travels.jpg HTTP/1.1" 200 -
127.0.0.1 - - [02/Jul/2025 11:50:20] "GET /www/iceland.html HTTP/1.1" 200 -
127.0.0.1 - - [02/Jul/2025 11:50:20] "GET /resources/iceland.jpg HTTP/1.1" 200 -
127.0.0.1 - - [02/Jul/2025 11:50:24] "GET /www/lofoten.html HTTP/1.1" 200 -
127.0.0.1 - - [02/Jul/2025 11:50:24] "GET /resources/lofoten.jpg HTTP/1.1" 200 -
127.0.0.1 - - [02/Jul/2025 11:50:32] code 404, message File not found
127.0.0.1 - - [02/Jul/2025 11:50:32] "GET /www/home.html HTTP/1.1" 404 -
```

Figura 1: Logging delle richieste