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

Leonardo Meloni

Anno Accademico 2024/2025

Indice

1 Introduzione

Il progetto proposto ha come finalità la creazione di un web server minimale utilizzando il linguaggio Python. Tale server deve essere in grado di gestire richieste HTTP di tipo GET e restituire contenuti statici in formato HTML, CSS o altri formati supportati tramite l'identificazione del MIME type. Questo lavoro si propone come esercizio pratico di applicazione dei concetti di rete, protocolli e programmazione socket a basso livello.

2 Obiettivi

- Implementare un semplice server HTTP in Python usando la libreria socket;
- Servire un sito web statico con almeno tre pagine HTML;
- Gestire correttamente richieste di tipo GET con risposta 200;
- Rispondere con codice 404 per risorse non presenti;
- Gestire correttamente i MIME type;
- Implementare un sistema di logging delle richieste;
- Aggiungere animazioni o layout responsive per un'esperienza utente migliorata.

3 Struttura del Progetto

- `server.py`: file principale contenente la logica del server;
- `www/`: directory che ospita le pagine HTML, i file CSS e altre risorse statiche;
- `log.txt`: file di log che viene generato automaticamente per registrare le richieste gestite.

4 Implementazione del Server

Il codice è stato sviluppato in Python utilizzando le librerie standard `socket`, `os`, `logging` e `mimetypes`. Il server si mette in ascolto sull'indirizzo `localhost` alla porta `8080`, accettando connessioni TCP in ingresso. Una volta stabilita la connessione, la richiesta HTTP viene ricevuta tramite la funzione `recv()` del socket.

La prima riga della richiesta viene analizzata per estrarre il metodo HTTP, il percorso della risorsa richiesta e la versione del protocollo. Il server è progettato per supportare esclusivamente richieste di tipo `GET`; qualsiasi altro metodo non è gestito.

Per determinare la risorsa da fornire, viene costruito il percorso assoluto del file richiesto combinando la directory base `www` con il percorso relativo estratto dalla richiesta. Per garantire la sicurezza del server, viene eseguito un controllo di "path traversal" confrontando il percorso ottenuto con la directory di root autorizzata. Se il percorso risulta fuori dai limiti consentiti, la richiesta viene bloccata.

Una volta verificata la validità del percorso, il server controlla l'esistenza del file usando la funzione `os.path.isfile()`. Se il file esiste, viene aperto in modalità binaria tramite la funzione `open(..., 'rb')` per permettere la lettura anche di contenuti non testuali, come immagini.

Il tipo MIME del file viene determinato utilizzando `mimetypes.guess_type()`, che consente al server di specificare correttamente il tipo di contenuto nel campo `Content-Type` della risposta HTTP. Viene quindi costruita una risposta conforme al protocollo HTTP/1.1, includendo intestazioni come `Content-Length`, `Connection` e altre, seguita dal corpo del messaggio contenente il file richiesto.

Nel caso in cui il file non venga trovato o venga rilevato un tentativo di accesso non autorizzato, il server genera una risposta con codice di stato `404` e fornisce una semplice pagina HTML che comunica l'errore al client.

Tutte le richieste gestite vengono registrate nel file `log.txt` tramite il modulo `logging`. Ogni voce di log contiene informazioni dettagliate quali l'indirizzo IP del client, la risorsa richiesta, il codice di stato restituito, il tipo MIME e la durata della risposta, fornendo un tracciamento utile per analisi e debugging.

5 Gestione degli Errori e Sicurezza

Il server è progettato per rispondere con codice HTTP 404 nel caso in cui il file richiesto non esista oppure si tenti l'accesso a risorse fuori dalla directory `www`, evitando così vulnerabilità comuni come il path traversal. Questa protezione è implementata controllando che il percorso del file richiesto inizi effettivamente con la directory di root.

6 Logging delle Richieste

Il file `log.txt` registra ogni connessione con le seguenti informazioni:

- Timestamp della richiesta;
- Indirizzo IP del client;
- Metodo HTTP;
- Risorsa richiesta;
- Codice di risposta HTTP;
- Tipo MIME restituito;

- Stato di chiusura della connessione.

7 Estensioni Implementate

- Determinazione automatica del MIME type dei file serviti (`.html`, `.css`, `.jpg`, ecc.);
- Layout responsive con grafica curata e accessibile;
- Logging dettagliato e continuo;
- Gestione robusta degli errori e sicurezza per file esterni;
- Supporto a file binari come immagini e risorse multimediali.

8 Conclusione

Lo sviluppo di questo server HTTP in Python mi ha permesso di mettere in pratica nozioni di programmazione di rete e gestione del protocollo HTTP. Ho acquisito familiarità nell'utilizzo dei socket per ricevere e rispondere a richieste, nell'analisi di una richiesta HTTP e nella costruzione di risposte corrette, inclusi header e contenuti di vario tipo. Inoltre, ho imparato a gestire la sicurezza tramite controlli sul percorso delle risorse (con `os.path.abspath()`) e a integrare un sistema di logging dettagliato usando il modulo `logging`. L'uso di `mimetypes` ha semplificato l'identificazione del tipo di file da restituire al client.

Questa esperienza ha migliorato la mia capacità di strutturare un progetto software, curando sia la parte back-end (server) sia l'interfaccia front-end (HTML/CSS).

Estensioni future

Nei prossimi sviluppi potrei aggiungere il supporto a richieste POST per gestire form, implementare HTTPS con certificato SSL per garantire una maggiore sicurezza.