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

**Sommario**

Il progetto ha come obiettivo la progettazione e implementazione di un Web Server minimale in Python, capace di servire pagine statiche HTML/CSS su localhost:8080. Il server implementato è in grado di soddisfare i requisiti richiesti, ossia: essere capace di gestire almeno 3 pagine HTML statiche, gestire richieste GET, gestire una risposta 404 quando si richiede un file inesistente e gestire i MIME types.

**Obiettivi**

* Ascoltare richieste su 127.0.0.1:8080.
* Servire pagine statiche HTML e CSS.
* Gestire correttamente il codice di risposta **200** (OK) e **404** (Not Found).
* Registrare su console tutte le richieste ricevute.
* Gestire correttamente MIME types differenti.

**Descrizione codice**

1. **Importazioni dei moduli**

* **socket**,utilizzato per la creazione e gestione di un socket IPv4 TCP.
* **os**, utilizzato per gestire i classpath.
* **threading**, utilizzato per gestire più client in contemporanea.
* **logging**, utilizzato per fare logging nel terminale.
* **urllib.parse** (per lafunzione **unquote**), utilizzato per rendere il codice più robusto nella lettura degli URL (ad esempio se fossero presenti spazi).
* **mimetypes**, utilizzato per gestire i MIME types.

**Immagine che contiene testo, Carattere, schermata, design

Il contenuto generato dall'IA potrebbe non essere corretto.**

1. **Configurazione del Server**

Viene impostato il localhost (**127.0.0.1**) come host del server, si dichiara come porta da utilizzare la **8080** e si inizializza come percorso assoluto della cartella da utilizzare dove trovare le risorse web la ‘**www**’.

Infine, usando il modulo logging si configura la struttura base del log da stampare ad ogni richiesta (orario della richiesta e messaggio del log).

**Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.**

1. **Funzioni**

Ci sono due funzioni:

* **handle\_client()** si occupa di gestire un singolo client.

In particolare, prima legge i dati dell’utente, se non li trova vuol dire che la connessione è stata chiusa. Poi decodifica la richiesta ignorando eventuali errori di decodifica, in modo tale da evitare che eventuali dati errati o eccezioni rallentino/blocchino l’esecuzione. Infine, viene fatto un semplice parsing della richiesta, la prima parte sarà interpretata come metodo della richiesta mentre la seconda come il percorso da cercare.

Immagine che contiene testo, schermata, software, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

In seguito, viene corretto il percorso in modo tale da cercarlo direttamente nella cartella ‘www’.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Una volta pulito il percorso del file richiesto si entra effettua il controllo per vedere se il file esiste. In caso affermativo si procede a leggerlo in formato binario (per essere sicuri di leggere i file in maniera corretta); a questo punto si chiama la funzione *guess\_type* di *mimetypes* per cercare il MIME da usare per leggere correttamente il file, in caso il tipo non venisse riconosciuto si usa un tipo generico e si costruiscono le variabili utili per il log.

Immagine che contiene testo, schermata, Carattere, software

Il contenuto generato dall'IA potrebbe non essere corretto.

In caso negativo invece, ossia se il file non dovesse esistere, verrebbe generata la pagina di errore 404 ed impostato lo status di conseguenza.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Terminato questo blocco la funzione termina inviando la risposta prodotta al client, chiudendo il socket a lui dedicato e loggando il risultato della richiesta.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

* **run()** si occupa di inizializzare e mettere in esecuzione il server HTTP.

Inizialmente si occupa di creare il socket ed inizializzarlo, impostando l’indirizzo da usare all’host e porta precedentemente impostati e mettendolo in ascolto di richieste. In seguito, fa partire il loop nel quale vengono instaurate le connessioni e viene creato un thread per ogni client in modo da rendere più efficiente il server.

Immagine che contiene testo, schermata, Carattere, software

Il contenuto generato dall'IA potrebbe non essere corretto.

**Flusso di esecuzione**

Per quanto visto precedentemente, si capisce che eseguito il codice, viene avviata la configurazione del server ed in seguito chiamata la funzione *run()* in modo da creare il socket e mettersi in ascolto di richieste dal client. In seguito, quando arriveranno le richieste verranno gestite tramite la funzione *handle\_client().*

Il ciclo di vita di un thread è gestito impostando *daemon=true*, in questo modo il thread viene correttamente terminato quando viene chiuso il programma. Tuttavia, se il programma si chiudesse improvvisamente, la conseguente chiusura dei thread potrebbe lasciare eventuali task in esecuzione incomplete (essendo questo un server minimale che gestisce unicamente pagine statiche non comporta grossi problemi ed è accettabile come rischio).

Ciascun socket invece viene correttamente chiuso dopo aver gestito una richiesta con *conn.close().*

**Funzionalità**

Per quanto visto finora si possono riassumere le principali funzionalità del server:

* È in grado di gestire più client contemporaneamente senza perdere eccessivamente in quanto ad efficienza.
* È in grado di gestire correttamente richieste di vario tipo.
* È in grado di gestire tipi di file differenti (.html, .css, .png, …).
* Permette di visualizzare correttamente i log relativi alle richieste di ciascun client.

**Conclusioni**

Il progetto ha dimostrato come, attraverso l'uso esclusivo della libreria standard di Python, sia possibile realizzare un web server minimale ma funzionale. Il server sviluppato è in grado di gestire richieste HTTP GET, servire pagine statiche e rispondere con codici di stato appropriati (200 e 404). L'implementazione del supporto per i MIME types e il logging delle richieste ha arricchito le funzionalità del server, rendendolo più completo. La gestione delle connessioni tramite thread daemon ha permesso di affrontare la concorrenza in modo efficace, seppur con le limitazioni note di tale approccio. Complessivamente, il lavoro svolto rappresenta una solida base per eventuali estensioni future,