Relazione Progetto di *Programmazione di Reti*

Autore
Gabriele Ranzari

Matricola
0001123397

Data

23 giugno 2025

1. Introduzione

Traccia scelta

Traccia 1 – Web Server + Sito Web Statico (livello base/intermedio)
Titolo: "Realizzazione di un Web Server minimale in Python e pubblicazione di un sito statico"

Descrizione Progetto

Il progetto prevede la creazione di un server HTTP minimale in Python, capace di gestire richieste GET e servire contenuti statici come pagine HTML, CSS e immagini. Il server deve rispondere correttamente con codice 200 per file esistenti e con codice 404 per risorse mancanti.

Parallelamente, viene sviluppato un sito statico con più pagine, organizzato per offrire una navigazione semplice e funzionale.

Sono previste estensioni opzionali come il supporto ai tipi MIME e il logging delle richieste.

Questo lavoro permette di approfondire la comprensione del protocollo HTTP e della comunicazione tramite socket, oltre a mettere in pratica la pubblicazione di un sito web statico.

Mia implementazione

Nella mia implementazione del progetto ho voluto creare un sito web statico simile a un portfolio, composto da diverse pagine accessibili pubblicamente, tra cui una sezione riservata all'amministratore accessibile solo tramite autenticazione con credenziali.

Per rendere il sito interattivo e responsive ho utilizzato Bootstrap 5, un framework CSS che facilita la gestione del layout, delle griglie e degli elementi grafici in modo adattabile a diverse dimensioni di schermo.

Il server Python realizzato gestisce le richieste HTTP principalmente di tipo GET e POST, servendo i file statici. Inoltre supporta correttamente la gestione dei MIME types per restituire il contenuto con il tipo appropriato, migliorando la compatibilità con i browser.

Per la parte di logging, il server registra su console ogni richiesta ricevuta, fornendo informazioni utili per il debug e il monitoraggio delle attività.

La gestione delle richieste POST è stata implementata per un form di consigli, con parsing dei dati inviati, salvataggio delle

informazioni in un file di testo e invio di una pagina di ringraziamento all'utente.

La sezione amministrativa è protetta da autenticazione HTTP Basic, richiedendo username (admin) e password (admin) per accedere a una pagina che visualizza tutti i messaggi raccolti tramite il form di consigli.

Infine, sono state implementate pagine di errore personalizzate per i casi di risorsa non trovata (404) e errori interni al server (500).

2. Struttura

Struttura del progetto

➔ /Progetto

: Cartella principale che contiene tutto il progetto.

- **server.py**
  : Script Python che implementa il server HTTP. Quando eseguito, ascolta le richieste su localhost:8080, gestisce le richieste GET e POST, serve i file statici, esegue il logging, gestisce l'autenticazione per l'area admin e restituisce le pagine di errore appropriate.
- **submissions.txt**
  : File di testo dove vengono salvati i messaggi inviati tramite il form del sito, usato poi per visualizzare i messaggi nell'area admin.

- **/www**
  : Cartella contenente i file HTML, CSS e le immagini da visualizzare sul sito.

  - **index.html**
    : File HTML che fa da homepage per il sito.
  - **about.html**
    : File HTML nella quale descrivo le mie passioni e le mie skill.
  - **gallery.html**
    : File HTML nel quale mostro i miei progetti fatti.
  - **contact.html**
    : File HTML con una form da compilare per mandare messaggi all'admin, con tre campi: Nome, Email e Message.
    Il messaggio viene salvato in *submissions.txt.*
  - **thankyou.html**
    : File HTML che viene mostrato quando viene mandato un messaggio tramite *contact.html*.
  - **admin.html**
    : File HTML area nascosta accessibile solo con le credenziali da admin, nella quale vengono letti e riportati i messaggi salvati su *submissions.txt*.
  - **404.html**
    : File HTML che viene mostrato quando avviene un errore di tipo 404 ovvero il server non trova la risorsa richiesta dal client.

- **500.html**
  : File HTML che viene mostrato quando avviene un errore di tipo 500 ovvero il server ha riscontrato un errore interno.
- **style.css**
  : File CSS per modificare l'aspetto grafico del sito attraverso bootstrap.
- **/images**
  : Cartella contenente tutte le immagini utilizzate nel sito.

Funzionamento Server

Funzioni

- **startServer()**
  - Avvia il socket TCP su *localhost:8080*, mette il server in ascolto e accetta le connessioni entranti. Per ogni client, chiama *handleRequest()*.

- **handleRequest(client, address)**
  - Riceve la richiesta HTTP dal client, ne esegue il parsing, e chiama il gestore appropriato. Gestisce anche errori e autenticazione.

- **handleGet(path, client)**
    - Gestisce le richieste GET. Se il file esiste, lo restituisce con *200 OK*. Altrimenti risponde con una pagina di errore ovvero *404.html*.
- **handlePost(path, headersBlock, body, client)**
    - Gestisce l'invio di dati via POST. Salva i dati in *submissions.txt* e risponde con una pagina di ringraziamento ovvero *thankyou.html*.
- **serveFile(filePath, client)**
    - Apre un file in modalità binaria e lo invia al client insieme a intestazioni HTTP corrette.
- **getMimeType(path)**
    - Calcola il tipo *MIME* corretto del file in base all'estensione.
- **sendResponse(client, statusCode, statusText, headers, bodyBytes)**
    - Compone e invia una risposta HTTP completa con stato, intestazioni e corpo.
- **checkAuth(headersBlock)**
    - Verifica se nella richiesta ci sono credenziali di accesso valide per accedere nella zona di *Admin*.

- **promptAuth(client)**
    - Invia al client una risposta *401 Unauthorized* con richiesta di autenticazione *Basic*.
- **serveAdmin(client)**
    - Se l’autenticazione è corretta, carica la pagina *admin.html* e mostra i messaggi salvati su *submissions.txt*.

Funzionamento

Quando si avvia lo script server.py, viene eseguita la funzione *startServer()* che crea un socket TCP in ascolto sulla porta 8080, su localhost. Il server resta in attesa che un client si colleghi. Appena una richiesta arriva, viene gestita dalla funzione *handleRequest()*, che analizza l’header HTTP per capire se si tratta di una richiesta GET, POST o altro, e la smista alla funzione appropriata.

Se il client sta cercando di visualizzare una pagina del sito, il server verifica se si tratta di una richiesta GET, determina il file richiesto e prova a leggerlo dalla cartella *www/*. Se il file esiste, lo invia al client come risposta, aggiungendo l’header HTTP corretto con il tipo MIME associato al file. Il client riceve i dati e mostra la pagina. Se il file richiesto non viene trovato, viene restituito un errore *404 Not Found* e la pagina di *errore 404.html*.

Nel caso in cui il client compili il form di contatto presente nella pagina contact.html, il browser invierà una richiesta POST al server. Quest'ultima viene gestita dalla funzione handlePost, che legge i dati inviati e li salva nel file di testo *submissions.txt*. Al termine dell'operazione, viene restituita una pagina di ringraziamento *thankyou.html*.

Se invece un utente tenta di accedere alla zona da *Admin*, il server controlla se nella richiesta sono presenti credenziali corrette tramite autenticazione HTTP Basic. Se le credenziali non ci sono o sono sbagliate, viene restituito un errore *401 Unauthorized* e il browser mostra automaticamente una finestra di login. Inserendo username e password validi, il server mostra la pagina *admin.html*, che contiene tutti messaggi in *submissions.txt*.

Infine, se durante l'elaborazione della richiesta si verifica un errore interno imprevisto, il server invia lo stato *500 Internal Server Error* e una pagina *500.html*.

3. Conclusione

Considerazioni finali

Lo sviluppo del progetto mi ha permesso di approfondire il funzionamento del protocollo HTTP e della comunicazione tramite socket in Python. Avevo già affrontato questi temi in C e Java durante le scuole superiori, ma non avevo mai realizzato un autenticazione HTTP Basic.

Inoltre, è stata la mia prima vera esperienza con Bootstrap, e questo mi ha aiutato a migliorare anche nella parte grafica e responsive del sito.

Fonti e riferimenti usati

- W3Schools
  - Per documentazione su Python, HTML e CSS.
- Stack Overflow
  - Per esempi e chiarimenti durante il debugging, per l’autorizzazione e nell’uso di Bootstrap.
- GetBootstrap
  - Per capire come usare e implementare Bootstrap.
- ChatGPT
  - Per debugging del codice, chiarimenti su funzioni di Python e su strutture di Bootstrap e per miglioramento del codice e della relazione.