Relazione – Web Server in Python

Marcello Spagnoli
Matricola: 0001117244
`marcello.spagnoli2@studio.unibo.it`

26 giugno 2025

1. Richieste della consegna

Progettare un semplice server HTTP in Python (usando `socket`) e servire un sito web statico con HTML/CSS.

Requisiti minimi

- Il server deve rispondere su `localhost:8080`.
- Deve servire almeno 3 pagine HTML statiche.
- Gestione di richieste `GET` e risposta con codice 200.
- Implementare risposta 404 per file inesistenti.

Estensioni opzionali

- Gestione dei MIME types (`.html`, `.css`, `.jpg`, ecc.).
- Logging delle richieste.
- Aggiunta di animazioni o layout responsive.

Output atteso

- Codice del server Python.
- Cartella `www/` con i file HTML/CSS.
- Relazione tecnica.

Funzionalità implementate

- Gestione delle richieste `GET` con risposta 200.
- Gestione file non trovati con risposta 404.
- Rifiuto dei metodi diversi da GET con errore 405.
- Gestione di richieste malformate con errore 400.
- Gestione eccezioni interne con errore 500.

- Logging su terminale e su file con timestamp.
- Riconoscimento automatico del MIME type.
- Animazioni e layout responsive

Contenuti statici

Nel progetto sono presenti 3 pagine HTML accessibili:

- `index.html`
- `about.html`
- `contact.html`

Tutti i file si trovano nella cartella `www/`. È presente anche il template `error.html`, utilizzato per mostrare i messaggi di errore personalizzati.
È presenta una cartella css dove si trova un general.css che viene applicato a tutte le pagine + 3 file css specifici per ogni pagina

Funzionamento

Il server HTTP è stato implementato in Python utilizzando il modulo `socket` per la comunicazione e `threading` per la gestione concorrente dei client. Di seguito si riassume il flusso di funzionamento:

- All'avvio, il server effettua il **bind** sulla porta 8080 e resta in ascolto su `localhost`.
- Ogni volta che un client si connette, viene avviato un nuovo thread che gestisce la connessione tramite la funzione `handle_client`.
- Il server legge la prima riga della richiesta HTTP e ne estrae metodo e path
- Se la richiesta non è ben formata (ad esempio mancano elementi), il server restituisce un errore **400 Bad Request**.
- Se il metodo non è `GET`, viene restituito un errore **405 Method Not Allowed**.
- Se il file richiesto esiste all'interno della cartella `www`, viene restituito con codice **200 OK** e relativo MIME type.
- Se il file non esiste, il server restituisce un errore **404 Not Found**, generando dinamicamente la pagina tramite un template HTML.
- Se durante il processo si verifica un errore imprevisto, viene inviato un messaggio **500 Internal Server Error**.

Ogni richiesta viene loggata sia sul terminale che in un file `server.log`, indicando timestamp, tipo di richiesta e codice di risposta. Le pagine di errore vengono generate dinamicamente usando il file `error.html` come template, sostituendo il messaggio nella variabile `{{ message }}`.
Tramite CSS sono state incluse animazioni e è stato implementato un layout responsive adatto al ridimensionamento

Test effettuati

Ho effettutato un test scrivendo un file `test_errori.py` che prova tutti i seguenti errori mandando richieste errate o non supportate per qualche motivo:

Test	**Metodo**	**Risultato Atteso**
File esistente	GET `/index.html`	200 OK con contenuto
File inesistente	GET `/nofile.html`	404 Not Found
Metodo non supportato	POST `/index.html`	405 Method Not Allowed
Richiesta malformata	GET (e basta)	400 Bad Request
Errore interno	GET /test500.html (non ha permessi di lettura)	500 Internal Server Error

Estensioni opzionali implementate

- Gestione dinamica del tipo MIME con il modulo `mimetypes`.
- Logging dettagliato (IP client, metodo, percorso, stato).
- Template per errori con messaggio dinamico (`{{ message }}`).
- Utilizzo di `threading.Thread` per supportare più client contemporaneamente.

Avvio del Server e Requisiti

Per avviare il server, è necessario avere Python 3 installato. Il server si avvia eseguendo il file `server.py` da terminale con il comando:

```
python3 server.py
```

Il server ascolterà sulla porta 8080 e servirà i file contenuti nella cartella `www/`.

Istruzioni per l'avvio e il test del server

1. Assicurarsi di avere installato Python 3.6 o versione superiore.
2. Posizionarsi nella cartella `elaboratoReti` dove si trova il file `server.py` e la cartella `www`.
3. Avviare il server eseguendo:

   ```
   python3 server.py
   ```

4. Una volta avviato, il server sarà in ascolto sulla porta 8080. Si può testare il funzionamento aprendo un browser e visitando:

   ```
   http://localhost:8080/index.html
   ```

5. Per provare i vari casi di errore, si possono usare strumenti come Thunder Client, provare a richiedere pagine che non esistono dal browser o provare il file `test_errori.py`.
6. Tutte le richieste e gli errori verranno registrati nel file `server.log` nella stessa cartella di `server.py`.
7. Per fermare il server premere `Ctrl+C` nel terminale dove è in esecuzione.

Struttura delle cartelle

```
elaboratoReti/
|-- server.py
|-- server.log
|-- www/
    |-- index.html
    |-- error.html
    |-- contact.html
    |-- test500.html (file vuoto solo per il test dell'errore 500, togliere i permessi)
    |-- gear.html
    |-- css/
    |-- img/
```

Dove:

- `server.py` è il file principale che avvia il server
- `server.log` contiene i log delle richieste e degli errori
- `www/` contiene le pagine web statiche servite dal server