Documentazione Progetto di Reti

Martina Malagoli
Davide Mancini

15 Luglio 2025

Indice

Capitolo 1

Introduzione

1.1 Obiettivi

Lo scopo del progetto è quello di realizzare un server HTTP in Python per gestire richieste da un client browser. Il server ha le seguenti funzionalità:

- Gestire richieste di pagine web HTML da parte di un client;
- Gestire richieste di file appartenenti allo standard MIME;
- Salvare ogni informazione ed errore in forma testuale sia su terminale sia in un file di testo dedicato al logging.

Ogni richiesta HTTP ricevuta otterrà una risposta con un codice che definisce lo stato di quest'ultima, tra cui i seguenti:

- **200 OK** indica che la risorsa richiesta esiste e verrà inviata al client;
- **400 Bad Request** indica che il messaggio di richiesta del client è vuoto;
- **404 Not Found** indica che la risorsa richiesta non esiste, in questo caso il server invierà in risposta al client una pagina HTML che informa quest'ultimo dell'inesistenza della risorsa;
- **500 Server Error** indica che il server ha ricevuto un errore inaspettato.

Il server contiene al suo interno 5 pagine HTML. Fra queste è presente la pagina che viene mostrata in caso d'inesistenza della risorsa richiesta al server. Le pagine HTML sono implementate in modo da essere responsive, ossia le dimensioni del contenuto delle pagine devono adattarsi alla dimensione dello schermo del client.

Capitolo 2

Implementazione

2.1 Server

2.1.1 Comunicazione socket

Il server è stato realizzato in modo che, una volta avviato, rimarrà in attesa di una connessione con un client all'indirizzo localhost (127.0.0.1) alla porta 8080. Il programma presenta una fase di inizializzazione, eseguita nei seguenti passaggi:

1. Creazione e formattazione del logger. Nel caso la cartella log nel server sia assente, viene automaticamente creata.

2. Apertura della connessione tramite una nuova socket, utilizzando la famiglia di socket *AF_INET* per connettersi agli host remoti e *SOCK_STREAM* per garantire una connessione orientata, affidabile e basata su un flusso di byte.

3. Associazione al server della socket formata dall'indirizzo 127.0.0.1 (localhost) con la porta 8080.

4. Inizializzazione della coda di backlog a 10, cioè accettazione fino a 10 richieste alla volta da parte del server, il quale scarterà quelle in eccesso.

Una volta conclusa questa fase, il server rimane in attesa di richieste fino alla chiusura manuale del programma.
Ogni volta che un client si connette al server, quest'ultimo legge la richiesta del client. La richiesta deve essere HTTP per essere riconosciuta dal server, altrimenti viene lanciata un'eccezione. Nel caso, invece, la richiesta sia vuota, il server restituisce l'errore *400 Bad Request*. Se invece la richiesta è ben

formattata, il server prende il file richiesto e controlla a quale tipologia di MIME appartiene. Qualora il file sia di tipo HTML si aggiunge al percorso locale del server la cartella */www*. Ogni file richiesto verrà letto e, se presente, verrà mandata una risposta con codice *200* contenente l'header HTTP e la risorsa richiesta. Qualora il file non sia presente, viene mandata una risposta con codice *404* contenente la pagina *notfound.html*. Eventuali errori inaspettati vengono segnalati dal logger. Una volta inviata la risposta, la connessione viene chiusa.

2.1.2 Gestione MIME

Ogni risorsa richiesta dal client è un file che può appartenere allo standard MIME. Nel codice del server viene utilizzata la funzione *guess_type* del package *mimetypes*, che restituisce a che tipologia appartiene un certo file. Qualora non venga riconosciuto, viene assegnato un tipo default chiamato *application/octet-stream*. Il tipo restituito verrà utilizzato nell'header della risposta. Invece il contenuto del file verrà letto utilizzando *open* di Python, sfruttando la modalità lettura file binario, e sarà salvato in *responseBody*, il quale verrà utilizzato per calcolare la lunghezza del body della risposta e verrà inviato come stream di byte al client.

2.1.3 Ulteriori cosiderazioni sull'implementazione

La funzione *sendResponse* crea un messaggio HTTP da mandare ad una certa socket (in questo caso il client) con codice di stato, tipologia di risorsa secondo lo standard MIME e il corpo del messaggio da mandare. Nella funzione viene utilizzato il metodo *sendall* anziché *send* in modo da garantire che il contenuto del messaggio venga mandato interamente. Il logger in questo caso notificherà eventuali errori, come ad esempio una connessione interrotta con il client. Altri dettagli sono elencati seguentemente:

- Il server è stato implementato con un solo thread, quindi può gestire una sola connessione alla volta.
- La coda di backlog viene impostata a 10 per ridurre il tempo di overhead nell'invio delle richieste del client dato dal caricamento dei file all'interno di una pagina HTML.
- Se il client richiede la root (/), esso viene reindirizzato automaticamente alla pagina principale */index.html*.

2.1.4 Librerie utilizzate

Per la creazione del server sono state utilizzate varie librerie rese disponibili da Python:

- **os**: modulo utilizzato per controllare l'eventuale esistenza della cartella di log e per crearla in caso negativo;
- **sys**: modulo utilizzato per la visualizzazione da terminale in standard output dei log;
- **mimetypes**: modulo utilizzato per gestire la tipologia di contenuti multimediali richiesti dal client;
- **logging**: modulo utilizzato per la realizzazione del logger e dei suoi handler;
- **socket**: modulo utilizzato per gestire la comunicazione fra server e client.

2.2 Pagine HTML

Le pagine HTML presenti all'interno della directory */www* compongono una piccola guida al gioco *The Legend of Zelda: A Link to the Past*. Tre di esse si occupano di approfondire un argomento specifico tra: dungeons, oggetti, boss nemici. Inoltre, presentano una serie di collegamenti che permettono di passare ad un punto d'interesse specifico all'interno di una delle altre pagine per approfondire un aspetto del gioco. La pagina *index.html*, invece, presenta una breve descrizione del gioco e dei suoi personaggi principali. L'unica pagina che non presenta contenuti informativi sul gioco è quella di reindirizzamento nel caso di richiesta di una risorsa inesistente (*404 Not Found*).
Infine, tutte le pagine HTML sono dotate di un menù a barra che facilita la navigazione fra esse e sono responsive.

2.3 Organizzazione dei file

I file all'interno del progetto sono stati organizzati in modo che:

- la **favicon** e il **file python del server** si trovino nella root;
- le **immagini** siano contenute ciascuna nella propria sottodirectory specifica per la loro tipologia all'interno della directory */images*;

- i **file HTML** e **CSS** siano contenuti nella directory */www*.

Capitolo 3

Informazioni aggiuntive

3.1 Esecuzione del server

Il server si può eseguire utilizzando un qualsiasi interprete python, ad esempio utilizzando il comando da terminale

```
python <path>/zelda_guide_server.py
```

dove *path* è il percorso contenente il file del server.
Una volta avviato, si può utilizzare un qualsiasi browser come client andando nella barra di ricerca e scrivendo l'URL

```
http://localhost:8080
```

aggiungendo un' eventuale risorsa cercata (per esempio *http://localhost:8080/favicon.ico*), altrimenti indirizzerà automaticamente alla pagina *index.html*.