

Relazione Tecnica - Programmazione di Reti

Progetto: Web Server in Python + Sito Web Statico

Studenti: Yue Shen (0001117228)

ChengZhou Hu (0001132306)

Corso di laurea: Ingegneria e Scienze Informatiche

Anno accademico: 2024/2025

1. Introduzione

Il progetto prevede la realizzazione di un Web Server minimale utilizzando il linguaggio Python e la libreria standard *socket*, con l'obiettivo di comprendere i meccanismi di base del protocollo HTTP e la gestione delle richieste client-server. Il server sviluppato è in grado di rispondere a richieste GET sulla porta localhost:8080 ed è stato progettato per servire contenuti statici scritti in HTML e CSS, simulando un semplice web-server locale.

2. Obiettivi e Requisiti

Requisiti Minimi:

- Rispondere su *localhost:8080*
- Gestione delle richieste HTTP GET
- Restituire risposta *200 OK* se il file richiesto esiste
- Gestione degli errori con *404 Not Found*, *403 Forbidden*, *405 Method Not Allowed*
- Servire almeno tre pagine HTML statiche

Estensioni Opzionali Implementate:

- Riconoscimento e gestione dei MIME types
- Logging delle richieste del file *log.txt*
- Layout responsive e animazioni nelle pagine web

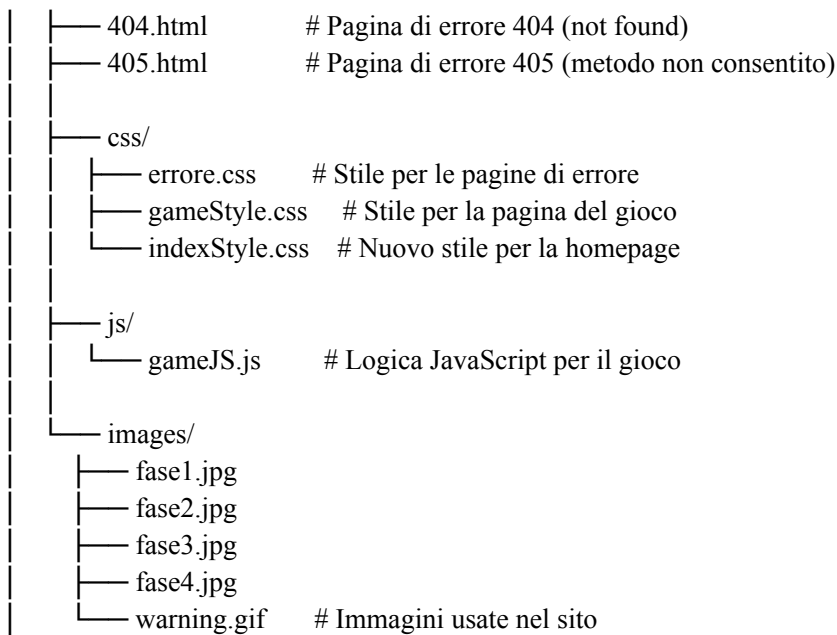
3. Struttura del Progetto

La struttura prevede una directory *www/* da cui vengono letti i file HTML, CSS o immagini richieste. Il server verifica anche che il path non punti a directory esterne alla root. Tutte le richieste vengono loggate a console.

- **server.py:** codice sorgente del Web Server Python.
- **www/:** cartella root del sito web statico.
- **log.txt:** registra tutte le richieste ricevute.

Progetto_di_Reti-main/

```
|
|— server.py          # Server HTTP minimale in Python
|— log.txt           # Log delle richieste ricevute
|— README.md        # Documentazione del progetto
|
|— www/             # Root directory dei contenuti serviti
|   |— index.html    # Homepage
|   |— game.html     # Pagina del gioco
|   |— secret.html   # Easter Egg
|   |— 403.html      # Pagina di errore 403 (forbidden)
```



4. Funzionamento del Web Server

Il server utilizza la libreria *socket* per creare un socket TCP, in ascolto su `localhost:8080`. Per ogni connessione:

1. Riceve la richiesta HTTP dal client
2. Analizza il metodo e il percorso richiesto.
3. Verifica l'esistenza del file all'interno della cartella *www/*
4. Restituisce una risposta *200 OK* con il contenuto del file se esiste, oppure *404 Not Found* con la pagina *404.html* se il file è assente
5. Determina il tipo di contenuto (MIME type) e registra i dettagli della richiesta nel file *log.txt*

5. Gestione dei MIME types

Tramite la funzione *get_mime_type()* che utilizza la libreria *mimetypes*, il server riconosce il tipo di contenuto del file richiesto (HTML, CSS, immagini, JS) e invia l'intestazione Content-Type corretta. Questo garantisce che il browser possa interpretare e rendere correttamente i file ricevuti.

6. Logging delle Richieste

Ogni richiesta ricevuta viene registrata nel file *log.txt* nel formato seguente:

```
[YYYY-MM-DD HH:MM:SS] IP METODO PATH
```

[data ora] path richiesto e codice di risposta

Questo consente di tenere traccia delle operazioni svolte dal server durante la sua esecuzione.

7. Gestione degli Errori

- **404 Not Found:** per file inesistenti
- **403 Forbidden:** se si tenta di accedere fuori dalla directory *www/*
- **405 Method Not Allowed:** per metodi HTTP diversi da GET

Queste pagine sono personalizzate e includono messaggi visivi e animazioni (GIF)

8. Il Sito Web Statico

Il sito servito dal server è composto da tre pagine HTML principali:

- **index.html**: homepage del sito con un pulsante per accedere alla pagina di gioco
- **game.html**: pagina con layout dinamico, immagini e script JavaScript
- **secret.html**: pagina *easter egg*
- pagine di errore personalizzate

Le pagine HTML sono stilizzate tramite file CSS dedicati. Sono presenti immagini (**fase1.jpg**, **fase2.jpg**, etc.) e un file JS (**gameJS.js**) per effetti dinamici.

9. Esecuzione del Server

Per eseguire il server è sufficiente lanciare il file **server.py**. Il sito è accessibile all'indirizzo: **http://localhost:8080**.

Per l'analisi del traffico, è possibile utilizzare strumenti come Wireshark per osservare il 3-way handshake TCP, le richieste GET e le relative risposte HTTP.

10. Esecuzione in Wireshark

Per verificare il corretto funzionamento del Web Server, è stata dunque effettuata un'analisi del traffico locale tramite *Wireshark*, monitorando le richieste e risposte durante l'accesso alla risorsa.

3-way handshake TCP: si osservano le tre fasi del collegamento TCP tra client e server:

1. **185** - SYN dal client (porta 10764) al server (porta 8080)
2. **186** - SYN-ACK dal server al client
3. **187** - ACK dal client al server

185	25.617286	127.0.0.1	127.0.0.1	TCP	56	10764 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
186	25.617258	127.0.0.1	127.0.0.1	TCP	56	8080 → 10764 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=65495 WS=256 SACK_PERM
187	25.617284	127.0.0.1	127.0.0.1	TCP	44	10764 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0

La richiesta get e risposta 200: il client invia una richiesta HTTP GET (*riga 188*) e otteniamo una risposta HTTP/1.1 200 OK, seguito dal contenuto HTML della pagina (text/html) (*riga 191*).

188	25.617614	127.0.0.1	127.0.0.1	HTTP	810	GET /game.html HTTP/1.1
189	25.617633	127.0.0.1	127.0.0.1	TCP	44	8080 → 10764 [ACK] Seq=1 Ack=767 Win=262400 Len=0
190	25.618693	127.0.0.1	127.0.0.1	TCP	580	8080 → 10764 [ACK] Seq=1 Ack=767 Win=262400 Len=536 [TCP PDU reassembled in 191]
191	25.618697	127.0.0.1	127.0.0.1	HTTP	108	HTTP/1.1 200 OK (text/html)
192	25.618711	127.0.0.1	127.0.0.1	TCP	44	10764 → 8080 [ACK] Seq=767 Ack=601 Win=2619136 Len=0
193	25.618724	127.0.0.1	127.0.0.1	TCP	44	8080 → 10764 [FIN, ACK] Seq=601 Ack=767 Win=262400 Len=0
194	25.618730	127.0.0.1	127.0.0.1	TCP	44	10764 → 8080 [ACK] Seq=767 Ack=602 Win=2619136 Len=0
195	25.619369	127.0.0.1	127.0.0.1	TCP	44	10764 → 8080 [FIN, ACK] Seq=767 Ack=602 Win=2619136 Len=0
196	25.619398	127.0.0.1	127.0.0.1	TCP	44	8080 → 10764 [ACK] Seq=602 Ack=768 Win=2100992 Len=0

> Frame 188: 810 bytes on wire (6480 bits), 810 bytes captured (6480 bits) on interface \Device\NPF_{...}_Loopback, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 10764, Dst Port: 8080, Seq: 1, Ack: 1, Len: 766

Source Port: 10764

Destination Port: 8080

[Stream Index: 89]

[Stream Packet Number: 4]

> [Conversation completeness: Complete, WITH_DATA (31)]

> [TCP Segment Len: 766]

Sequence Number: 1 (relative sequence number)

Sequence Number (raw): 1949106142

[Next Sequence Number: 767 (relative sequence number)]

Acknowledgment Number: 1 (relative ack number)

Acknowledgment number (raw): 292994880

0101 ... = Header Length: 20 bytes (5)

> Flags: 0x010 (PSH, ACK)

Window: 10233

[Calculated window size: 2619648]

[Window size scaling factor: 256]

Checksum: 0xd146 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

0010 7f 00 00 01 7f 00 00 01 2a 0c if 90 74 2c ff de *...GET
0020 11 76 bf 40 50 18 27 f9 d1 46 00 00 47 45 54 20 v. @P... F..GET
0030 2f 67 61 6d 65 2e 68 74 6d 6c 20 48 54 54 50 2f /game.ht ml HTTP/
0040 31 2e 31 0d 0a 48 6f 73 74 3a 20 6c 6f 63 61 6c 1.1- Hos t: local
0050 68 6f 73 74 3a 38 30 38 30 0d 0a 43 6f 6e 6e 65 host:808 0. Conne
0060 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 ction: k eep-aliv
0070 65 0d 0a 73 65 63 20 63 68 2d 75 61 3a 20 22 4e e- sec-c h-uas: "N
0080 6f 74 29 41 3b 42 72 61 6e 64 22 3b 76 3d 22 38 otAJ;Bra nd";v="8
0090 22 2c 20 22 43 68 72 6f 6d 69 75 6d 22 3b 76 3d ", "Chro nium";v=
00a0 22 31 33 38 22 2c 20 22 47 6f 6f 6f 6c 65 20 43 "138", " Google C
00b0 68 72 6f 6d 65 22 3b 76 3d 22 31 33 38 22 0d 0a hrome";v ="138"
00c0 73 65 63 2d 63 68 2d 75 61 2d 6d 6f 62 69 6c 65 sec-ch-u a-mobile
00d0 3a 20 3f 30 0d 0a 73 65 63 2d 63 68 2d 75 61 2d ; 70: se c-ch-ua-
00e0 70 6c 61 74 66 6f 72 6d 3a 20 22 57 69 6e 64 6f platform : "Windo
00f0 77 73 22 0d 0a 55 70 67 72 61 64 65 2d 49 6e 73 ws" Upg rade-Int
0100 63 63 72 65 2d 57 65 71 75 65 73 74 73 3a 20 Reque re-ques:1
0110 31 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d l;Jser- Agent: M
0120 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 57 69 6e 64 ozilla/5 .0 (Wind
0130 6f 77 73 20 4e 54 20 31 30 2e 30 30 20 57 69 6e ows NT 1.0.0; Win
0140 36 34 30 20 78 36 34 29 20 41 70 6c 65 57 65 d4; v64) Applewe
0150 62 4b 69 74 2f 35 33 37 2e 33 36 20 28 4b 48 54 bkit/537 .36 (KHT
0160 4d 4c 2c 20 6c 69 6b 65 20 47 65 63 6b 6f 29 20 ML, like Gecko)
0170 43 68 72 6f 6d 65 2f 31 33 38 2e 30 2e 30 2e 30 Chrome/1 38.0.0.0
0180 20 53 61 66 61 72 69 2f 35 33 37 2e 33 36 0d 0a Safari/ 537.36
0190 41 63 63 65 70 74 3a 20 74 65 78 74 2f 68 74 6d Accept: text/htm
01a0 6c 2c 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 68 l,applic ation/xh
01b0 74 6d 6c 2b 78 6d 6c 2c 61 70 70 6c 69 63 61 74 tml+xml, applicat
01c0 6d 6f 6a 7f 78 6d 6c 3b 73 5d 3b 3a 3d 3c 6d 6a tml+xml; m=0.0.1e

Chiusura della connessione: la connessione viene chiusa poi correttamente con pacchetti FIN, ACK (*righe 193-196*), rispettando il meccanismo TCP di terminazione della sessione.

193	25.618724	127.0.0.1	127.0.0.1	TCP	44	8080 → 10764 [FIN, ACK] Seq=601 Ack=767 Win=262400 Len=0
194	25.618730	127.0.0.1	127.0.0.1	TCP	44	10764 → 8080 [ACK] Seq=767 Ack=602 Win=2619136 Len=0
195	25.619369	127.0.0.1	127.0.0.1	TCP	44	10764 → 8080 [FIN, ACK] Seq=767 Ack=602 Win=2619136 Len=0
196	25.619398	127.0.0.1	127.0.0.1	TCP	44	8080 → 10764 [ACK] Seq=602 Ack=768 Win=2100992 Len=0

11. Considerazioni Finali

Il progetto ha permesso di comprendere i concetti fondamentali del protocollo HTTP, la gestione delle connessioni tramite socket in Python e la gestione dei contenuti web statici. Abbiamo implementato un server semplice ma completo, in grado di gestire le principali funzionalità richieste da un web server base e arricchito con funzionalità aggiuntive quali il logging e la gestione dei MIME types. Arricchendo così le nostre competenze pratiche sul funzionamento delle reti e della comunicazione client-server.