RELAZIONE TECNICA - FILIPPO PATRIGNANI

WEB SERVER STATICO IN PYTHON

OBIETTIVO DEL PROGETTO

Realizzare un web server HTTP completo in Python, in grado di:
- Ricevere e gestire richieste HTTP multiple (GET, HEAD, OPTIONS)
- Servire file statici da una cartella locale con supporto MIME esteso
- Gestire correttamente richieste valide (200 OK) e diversi tipi di errore (404 Not Found, 405 Method Not Allowed)
- Fornire funzionalità avanzate come logging, statistiche e directory listing
- Supportare connessioni concorrenti tramite threading

TECNOLOGIE UTILIZZATE

- Linguaggio: Python 3.11
- Ambiente: Visual Studio Code
- Moduli principali:
  - socket: per la comunicazione TCP/IP
  - threading: per la gestione concorrente delle richieste
  - logging: per il sistema di log avanzato
  - pathlib: per la gestione moderna dei percorsi
  - datetime: per timestamp e statistiche temporali
  - mimetypes: per il riconoscimento automatico dei tipi MIME
  - urllib.parse: per il parsing e decodifica degli URL

STRUTTURA DEL PROGETTO

Il progetto e organizzato in una struttura modulare:

- src/server.py: contiene il codice completo del server HTTP
- src/www/: directory principale dei file statici da servire
  - index.html: home page principale del server con interfaccia moderna
  - intro.html: introduzione tecnica al protocollo HTTP
  - server.html: documentazione dettagliata del funzionamento del server
  - header.html: esempi pratici di header HTTP di richiesta e risposta
  - 404.html: pagina di errore personalizzata
  - style.css: foglio di stile responsive per tutte le pagine
  - script.js: JavaScript per funzionalità interattive
  - icons/: directory contenente icone SVG generate in python
- src/logs/: directory per i file di log del server
- README.md: documentazione del progetto

ARCHITETTURA DEL SERVER

Classe SimpleHTTPServer

Il server e implementato come classe Python che incapsula tutte le funzionalità:

```
class SimpleHTTPServer:
    def __init__(self, host='localhost', port=8080, document_root='www')
```

La classe gestisce:
- Configurazione di rete (host, porta, document root)
- Sistema di logging multi-livello
- Statistiche delle richieste in tempo reale
- Mappatura dei tipi MIME
- Gestione thread per connessioni concorrenti

Gestione delle Richieste HTTP

Il server implementa un parser HTTP completo che gestisce:

1. Parsing della richiesta: Analisi della request line e degli header HTTP
2. Validazione del metodo: Supporto per GET, HEAD, OPTIONS
3. Decodifica URL: Gestione di caratteri speciali e query parameters
4. Sicurezza: Protezione contro directory traversal attacks

Metodi HTTP Supportati

- GET: Retrieval completo delle risorse richieste
- HEAD: Ottenimento dei soli header senza body della risposta
- OPTIONS: Supporto per richieste CORS preflight

FUNZIONAMENTO DETTAGLIATO

Ciclo di Vita della Richiesta

1. Ascolto: Il server si mette in ascolto su localhost:8080
2. Accettazione: Ogni nuova connessione viene accettata e assegnata a un thread separato
3. Parsing: La richiesta HTTP viene analizzata e validata
4. Routing: Il path viene processato e mappato al file corrispondente
5. Servizio: Il file viene letto e servito con gli header appropriati
6. Logging: Ogni operazione viene registrata nel sistema di log
7. Chiusura: La connessione viene chiusa correttamente

Gestione dei File Statici

Il server implementa diverse strategie per il servizio dei file:

- File singoli: Servizio diretto con detection automatica del MIME type
- Directory: Generazione automatica di directory listing HTML
- File index: Ricerca automatica di index.html nelle directory
- Errori 404: Pagina di errore personalizzata o fallback generica

Sistema di Logging

Implementazione del sistema di logging:

```
def setup_logging(self):
    log_dir = Path(__file__).parent / 'logs'
    log_dir.mkdir(exist_ok=True)

    file_handler = logging.FileHandler(log_dir / f'server_{date}.log')
    console_handler = logging.StreamHandler()
```

Il sistema registra:
- Timestamp precisi di ogni operazione
- Indirizzo IP e porta del client
- Metodo HTTP e path richiesto
- Codice di stato della risposta
- Dimensione del contenuto servito
- Eventuali errori ed eccezioni

GESTIONE DEI MIME TYPE

Il server supporta un'ampia gamma di tipi MIME attraverso una mappatura estesa:

```
self.mime_types = {
    '.html': 'text/html; charset=utf-8',
    '.css': 'text/css',
    '.js': 'application/javascript',
    '.json': 'application/json',
    '.png': 'image/png',
    '.jpg': 'image/jpeg',
    '.svg': 'image/svg+xml',
    '.pdf': 'application/pdf',
    '.zip': 'application/zip'
}
```

Questa mappatura garantisce che i browser interpretino correttamente ogni tipo di contenuto servito.

FUNZIONALITA' AVANZATE

Statistiche del Server

Il server mantiene statistiche dettagliate accessibili via endpoint /stats:

- Numero totale di richieste ricevute
- Tempo di uptime del server
- Distribuzione dei codici di stato HTTP
- File più richiesti con contatori di accesso
- Informazioni di sistema e configurazione

Directory Listing

Generazione automatica di pagine HTML per la navigazione delle directory:

```
def generate_directory_listing(self, dir_path):
    # Genera HTML navigabile per il contenuto della directory
    # Include dimensioni file e link di navigazione
```

Gestione Errori Avanzata

- 404 Not Found: Pagina personalizzata con design moderno
- 405 Method Not Allowed: Risposta informativa sui metodi supportati
- 500 Internal Server Error: Gestione graceful degli errori interni

Threading e Concorrenza

Ogni richiesta viene gestita in un thread separato per permettere:
- Servizio simultaneo di multiple richieste
- Non-blocking del server principale
- Maggiore throughput e responsiveness

SICUREZZA

Il server implementa diverse misure di sicurezza:

Path Traversal Protection

```
file_path = file_path.resolve()
if not str(file_path).startswith(str(self.document_root.resolve())):
    return self.serve_404()
```

Input Validation

- Validazione della sintassi delle richieste HTTP
- Controllo dei metodi HTTP supportati
- Sanitizzazione dei path richiesti

TESTING E UTILIZZO

Avvio del Server

```
cd src
python server.py [--host HOST] [--port PORT] [--dir DIRECTORY]
```

Endpoint Disponibili

- /: Homepage principale
- /intro.html: Introduzione tecnica
- /server.html: Documentazione server
- /header.html: Esempi header HTTP
- /stats: Statistiche in tempo reale
- Qualsiasi file nella directory www/

Testing Funzionalità

Il progetto include script di testing per verificare:
- Corretta gestione dei metodi HTTP
- Performance sotto carico
- Gestione degli errori
- Compatibilità browser

ESTENSIONI IMPLEMENTATE

Oltre ai requisiti minimi, il progetto include:

1. MIME Types Estesi: Supporto per oltre 10 tipi di file diversi
2. Logging Professionale: Sistema di log su file con rotazione giornaliera
3. Design Responsive: Interfaccia web moderna che si adatta a tutti i dispositivi
4. Statistiche Live: Dashboard in tempo reale con metriche del server
5. Directory Listing: Navigazione automatica delle directory
6. Multi-threading: Gestione concorrente delle richieste
7. Sicurezza Avanzata: Protezione contro attacchi comuni
8. Pagine Errore Personalizzate: Design professionale per errori 404