# Realizzazione di un Web Server minimale in Python e pubblicazione di un sito statico

[nicolas.montanari3@studio.unibo.it](mailto:nicolas.montanari3@studio.unibo.it) 0001128064

[mattia.mularoni@studio.unibo.it](mailto:mattia.mularoni@studio.unibo.it) 0001126065

[marko.cobo@studio.unibo.it](mailto:marko.cobo@studio.unibo.it) 0001127943

## 1.INTRODUZIONE

Nel contesto del corso di Programmazione di Reti, è stato sviluppato un web server minimale utilizzando il linguaggio Python e la libreria standard socket, con l’obiettivo di comprendere il funzionamento basilare del protocollo HTTP e la gestione delle richieste client-server. Il server è stato progettato per servire contenuti statici scritti in HTML e CSS, simulando un semplice web-server locale.

## 2. REQUISITI FUNZIONALI

* Sviluppare un server HTTP in grado di rispondere su localhost:8080.
* Gestire correttamente richieste GET con risposta 200 OK.
* Fornire una risposta 404 Not Found per file non esistenti.
* Servire almeno tre pagine HTML statiche.

## 3. REQUISITI NON FUNZIONALI

* Gestione dei MIME types
* Gestione dei logging
* Layout responsive

## 4. ARCHITETTURA DELL’APPLICAZIONE

La struttura dell’applicazione è composta come di seguito:

Immagine che contiene testo, Carattere, schermata, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

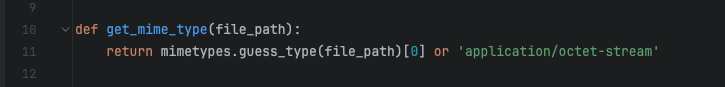
## 5. FUNZIONAMENTO DELL’APPLICAZIONE

Il server è implementato in Python utilizzando il package socket, e opera secondo il seguente flusso:

1. Crea un socket TCP e lo associa a 0.0.0.0:8080 (localhost).
2. Accetta connessioni client in un ciclo infinito.
3. Analizza la richiesta ricevuta (solo metodo GET supportato).
4. Risolve il path richiesto e verifica che non esca dalla directory www/.
5. Serve il file se esiste (200 OK), altrimenti restituisce un errore 404 o 403 (nel caso in cui si provi a richiedere altre root diverse da /www).
6. Determina il tipo MIME del file per la corretta visualizzazione.
7. Registra nel log le richieste e il tipo di risposte inviate.

### 5.1 Gestione MIME types

La funzione get\_mime\_type() utilizza la libreria mimetypes per restituire il Content-Type corretto (es. text/html, text/css, image/jpeg). Questo consente ai browser di interpretare correttamente le risorse caricate. Esempio in pyton:



### 5.2 GESTIONE DEGLI ERRORI

Il server gestisce tre principali errori HTTP:

* **404 Not Found**: il file richiesto non esiste.
* **403 Forbidden**: tentativo di accedere fuori dalla directory www/.
* **405 Method Not Allowed**: richieste diverse da GET.

### 5.3 LOGGING

Ogni richiesta viene registrata a console. Questo permette un tracciamento basilare delle operazioni effettuate dal server:

Immagine che contiene testo, Carattere, schermata, bianco

Il contenuto generato dall'IA potrebbe non essere corretto.

## 6. IL SITO WEB STATICO

Queste pagine presentano contenuti relativi a tutte le regioni italiane, con l’uso di:

* **Bootstrap:** per layout responsive e componenti grafici.
* Caroselli, tabelle, immagini.
* Navigazione tra pagine tramite frecce SVG e link.

Il sito è responsive, ottimizzato per dispositivi mobili grazie al meta viewport e alla libreria Bootstrap CDN.

È presente una homepage raffigurante la cartina dell’Italia, dalla quale è possibile navigare nelle varie regioni

## 7. ESECUZIONE IN WIRESHARK

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

La comunicazione segue il tipico schema client-server su TCP:

* I pacchetti iniziali (righe 874-879) mostrano il **3-way handshake TCP**: SYN, SYN-ACK e ACK, che stabilisce la connessione tra client (porta 62868) e server (porta 8080).
* Successivamente (riga 880), il client invia una richiesta **HTTP GET** (GET / HTTP/1.1).
* Il server risponde (riga 882) con un messaggio HTTP/1.1 200 OK, seguito dal contenuto HTML della pagina (text/html).
* La connessione viene poi chiusa correttamente con pacchetti FIN, ACK (righe 886–888).
* Altre richieste HTTP seguono, ad esempio per /script.js (riga 890) e /style.css (riga 894), con relative risposte HTTP/1.1 200 OK (text/javascript, text/css).

Questo traffico è tipico di una semplice interazione con un **server web locale**, che serve contenuti HTML, CSS e JavaScript.

## ESECUZIONE DEL SERVER

Per avviare il server:



Per aprire la pagina web:

[**http://localhost:8080**](http://localhost:8080)

## CONSIDERAZIONI FINALI

Il server sviluppato è uno strumento didattico efficace per esplorare i concetti chiave della programmazione di reti e del web server design. L’implementazione diretta del protocollo HTTP rafforza la comprensione delle richieste client, delle risposte del server e della gestione dei contenuti web. Il progetto ha permesso di comprendere il funzionamento interno di un server HTTP e la gestione diretta dei socket in Python. Pur essendo un server minimale, esso copre le funzionalità essenziali del protocollo HTTP/1.1.