

Madagascar Web Server

Samorì Andrea

18 maggio 2024

Indice

1	Descrizione	2
2	Funzionamento	3
2.1	Dettaglio	3
3	Come utilizzarlo	6
3.1	Requisiti	6
3.2	Avvio	6
4	Considerazioni Aggiuntive	8

Capitolo 1

Descrizione

Il progetto da me realizzato consiste in un semplice web server, che ti permette di visualizzare immagini di alcuni degli animali del Madagascar. Per realizzarlo ho utilizzato il linguaggio di programmazione *Python*, servandomi della programmazione tramite *socket* TCP e della programmazione concorrente tramite *thread*.

Capitolo 2

Funzionamento

2.1 Dettaglio

Il server da me implementato è un server che utilizza il protocollo *TCP* per mandare e ricevere richieste *HTTP*. Le richieste che può gestire sono quelle basilari, quindi pagine *HTML*, fogli di stile *CSS* e immagini statiche. Il server è in grado di gestire più richieste simultanee e inviare adeguatamente le risposte ai client che le richiedono. Ora analizziamo le varie parti del codice per comprendere al meglio il loro funzionamento.

```
HOST = input("Inserire l'indirizzo dove hostare il server:")
if HOST == "":
    HOST = 'localhost'

PORT = input("Inserire la porta:")
if PORT == "":
    PORT = 8080

BUFSIZ = 1024
ADDR = (HOST, int(PORT))

server = socket(AF_INET, SOCK_STREAM)
server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
server.bind(ADDR)
```

In questa parte viene chiesto all'utente di inserire l'indirizzo ip e la porta dove vuole hostare il server. Poi nella parte successiva viene creata la socket del server, che accetterà le richieste dei client. Per permettere di riutilizzare le stesse risorse, ho aggiunto la seguente linea di codice:

```
server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
```

Nella seguente parte, viene creato il thread che serve ad accettare i vari client che vogliono interfacciarsi con il server

```
accept_thread = Thread(target=accetta_richieste)
accept_thread.start()
accept_thread.join()
```

La funzione che permette di fare questo è accetta_richieste

```
def accetta_richieste():
    while True:
        client, client_address = server.accept()
        print("%s:%s - si e' collegato." % client_address)
        Thread(target=gestisce_client, args=(client,)).start()
```

Ogni volta che un client prova a collegarsi, viene creato un thread a parte che si occupa di gestirlo, tramite la funzione gestisce_client

```
def gestisce_client(client):
    message = client.recv(1024)
    try:
        filename = message.split()[1]

        if "/" == filename.decode():
            filename = b'/index.html'

            f = open(filename[1:], 'rb')
            outputdata = f.read()
            client.send("HTTP/1.1 200 OK\r\n\r\n".encode())
            client.send(outputdata)
            client.send("\r\n".encode())
            client.close()

    except IOError:
        client.send(bytes("HTTP/1.1 404 Not Found\r\n\r\n", "UTF-8"))
        client.send(bytes("<html><head></head><body><h1>\r\n-----404 Not Found</h1></body></html>\r\n", "UTF-8"))
        client.close()
```

Questa funzione si occupa di ricavare i messaggi del client. Da questi messaggi, la funzione estrae il nome della risorsa che il client vuole visualizzare e, se esiste, viene mandato il messaggio di OK e il messaggio con il contenuto della risorsa richiesta. Se invece la risorsa richiesta non esiste, il server invia un messaggio di errore e una pagina che visualizza che c'è stato un errore.

```
except IOError:
    client.send(bytes("HTTP/1.1 404 Not Found\r\n\r\n", "UTF-8"))
    client.send(bytes("<html><head></head><body><h1>\r\n-----404 Not Found</h1></body></html>\r\n", "UTF-8"))
    client.close()
```

Per uscire dal server in maniera corretta e senza eccezioni, ho definito la seguente funzione

```
def signal_handler(signal, frame):
    print('Sto -uscendo -dal -server - ( Ctrl+C - premuto )')
    try:
        if( server ):
            server .server_close()
    finally:
        sys .exit(0)

signal .signal(signal .SIGINT, signal_handler)
```

che permette di uscire dall'applicazione premendo la combinazione di tasti CTRL + C.

Capitolo 3

Come utilizzarlo

3.1 Requisiti

Avere installato Python all'interno del proprio computer.

3.2 Avvio

Per iniziare ad utilizzare il web server, basta scaricare il progetto da Github a questo link. Una volta scaricati tutti i file necessari, aprire un terminale e spostarsi all'interno del file system nella cartella dove sono scaricati i file. Ora basta copiare la seguente linea di codice per avviare il web server:

```
python3 server.py
```

Una volta lanciato il comando, si vedrà sul terminale un messaggio che conferma la corretta partenza del server:

```
[> python3 server.py]  server.setsockopt(SOL_SOCKET,
Inserire l'indirizzo dove hostare il server: 127.0.0.1
Inserire la porta: 8080
Madagascar Web Server in esecuzione
In attesa di connessioni...
[>]
```

Ora si è pronti per connettersi al server. Aprire il browser di maggiore preferenza, e inserire nella barra di ricerca l'indirizzo scelto, seguito da due punti e la porta scelta, ad esempio:

127.0.0.1:8080

e il gioco è fatto! Siamo collegati al server!



Ora si può navigare nel sito, visualizzando alcuni degli animali del Madagascar, cliccando gli appositi link. Per uscire e terminare il server, basta premere la combinazione di tasti CTRL+C nel terminale e il programma terminerà senza errori.

Capitolo 4

Considerazioni Aggiuntive

Per evitare di incombere in errori perché l'utente non ha inserito alcun indirizzo o porta, ho deciso di inserire in modo arbitrario l'indirizzo *localhost (127.0.0.1)* e la porta *8080* (questo dopo aver appositamente controllato che le variabili *HOST* e *PORT* fossero vuote). Per testare che tutto funzionasse, mi sono collegato al server tramite i browser più utilizzati, cioè Chrome, Firefox e Safari.

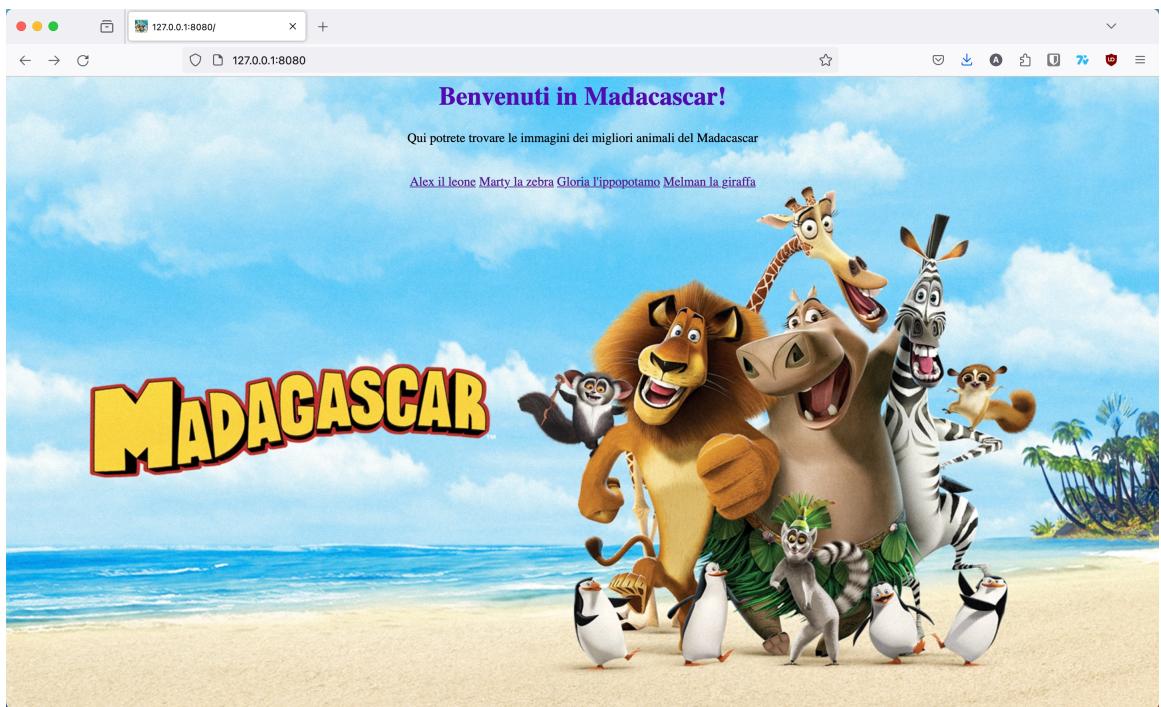


Figura 4.1: Sito visualizzato con Firefox



Figura 4.2: Sito visualizzato con Safari

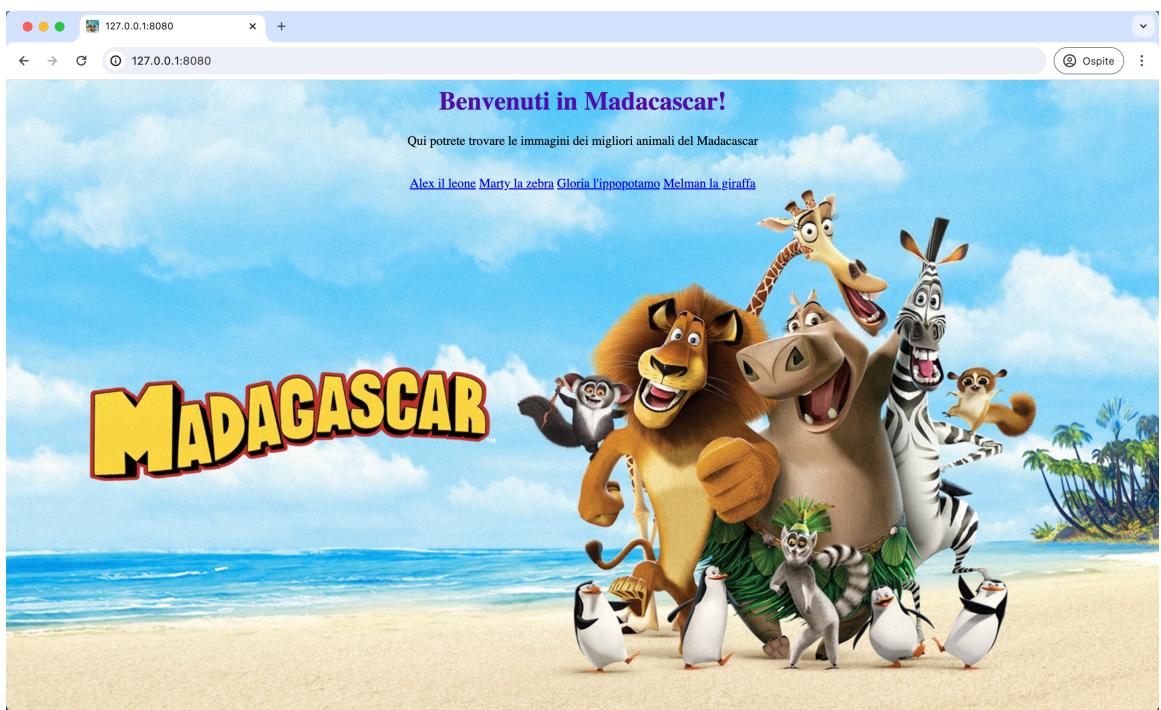


Figura 4.3: Sito visualizzato con Chrome