

Nome: Luca
Cognome: Ponseggi
Mail: luca.ponseggi@studio.unibo.it
Matricola: 0001089462

Traccia 2: Web Server Semplice

Requisiti per il programma

Versione Python: 3.11.7
Ambiente Anaconda: Base
Editor di testo: Spyder oppure Visual Studio Code

Funzionamento

Per la creazione del server HTTP ho utilizzato i moduli standard `http.server` e `socketserver`. Queste semplificano abbastanza la gestione delle richieste HTTP e delle connessioni di rete.

Il server funziona in questo modo:

1. Quando riceve una richiesta HTTP GET, viene analizzato l'URL per determinare il percorso del file e qualsiasi parametro passato.
2. Se il file richiesto esiste, viene inviata una risposta HTTP 200 OK al client insieme al contenuto del file richiesto.
3. Se il file non esiste, viene inviata una risposta HTTP 404 Not Found al client.

È stata utilizzata la classe `socketserver.ThreadingTCPServer` per gestire le connessioni di rete per il server HTTP utilizzando la programmazione concorrente a livello di thread per controllare più richieste contemporaneamente.

Esecuzione

Aprire Visual Studio Code tramite Anaconda.

Server

Posizionarsi nella cartella dove è presente il file “server.py”;
eseguire il comando “python .\server.py”.
(oppure cliccare sul “triangolino” in alto a destra di VS Code)

Verrà scritto su console: “Server funzionante su: `http://{address}:{port} ...`”, quindi in questo momento il server è attivo e pronto a rispondere alle richieste di GET.

Client

Ho testato il server in 3 modi diversi:

1. Il primo è il modo più intuitivo, ossia aprire un browser qualunque e inserire l'indirizzo di loopback con relativa porta, quindi "127.0.0.1:8000" nella barra degli URL; si verrà indirizzati nella pagina base "index.html".
Nella pagina sarà possibile inviare richieste GET al server tramite un modulo HTML. Tuttavia, queste richieste verranno reindirizzate allo stesso sito senza un'interazione dinamica con il server. Ciò significa che i parametri inviati saranno visibili nell'URL, ma non produrranno alcun effetto concreto sul sito poiché è gestito staticamente, verranno però stampati a schermo anche dalla parte del server.
2. Il secondo metodo è utilizzare un piccolo programma che funge da client tramite l'utilizzo della libreria socket eseguendo una richiesta HTTP GET.
3. Il terzo metodo fa utilizzo di Insomnia, un software mirato nel fare richieste fungendo da client verso un server con una GUI molto comoda per fare testing.

Immagini di test server

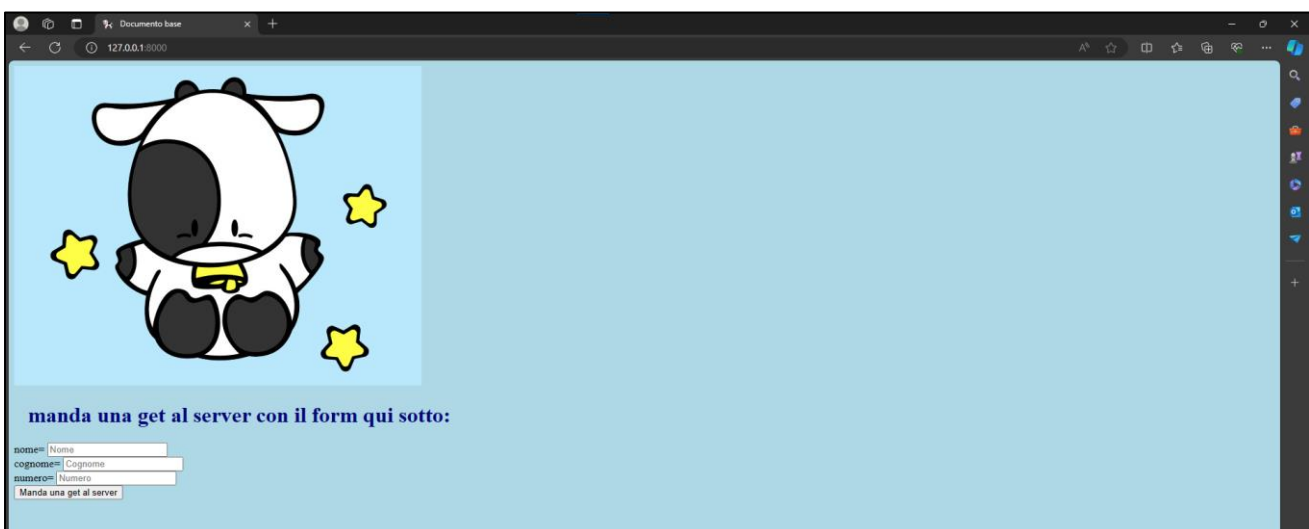
Avviato tramite il "triangolino" di VS Code

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\Luca\Desktop\progettoReti> & C:/Users/Luca/anaconda3/python.exe c:/Users/Luca/Desktop/progettoReti/server.py
Server funzionante su: http://127.0.0.1:8000 ...
```

Immagini di test Client

Utilizzo di un browser qualsiasi (edge in questo caso)

Visione client:



Visione server:

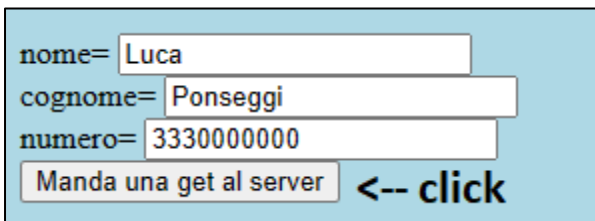
```
127.0.0.1 - - [16/May/2024 01:17:03] "GET / HTTP/1.1" 200 -  
Eseguita GET da ('127.0.0.1', 64276) su file "\index.html", utilizzato mime type "text/html" per header e inviato codice di risposta "200" (OK)  
Sono stati passati questi parametri: {}  
  
127.0.0.1 - - [16/May/2024 01:17:04] "GET /mystyle.css HTTP/1.1" 200 -  
127.0.0.1 - - [16/May/2024 01:17:04] "GET /immagine.jpg HTTP/1.1" 200 -  
Eseguita GET da ('127.0.0.1', 64277) su file "\mystyle.css", utilizzato mime type "text/css" per header e inviato codice di risposta "200" (OK)  
Sono stati passati questi parametri: {}  
  
Eseguita GET da ('127.0.0.1', 64278) su file "\immagine.jpg", utilizzato mime type "image/jpeg" per header e inviato codice di risposta "200" (OK)  
Sono stati passati questi parametri: {}  
  
127.0.0.1 - - [16/May/2024 01:17:04] "GET /favicon.ico HTTP/1.1" 200 -  
Eseguita GET da ('127.0.0.1', 64279) su file "\favicon.ico", utilizzato mime type "image/x-icon" per header e inviato codice di risposta "200" (OK)  
Sono stati passati questi parametri: {}
```

Il client fa la richiesta di 4 elementi:

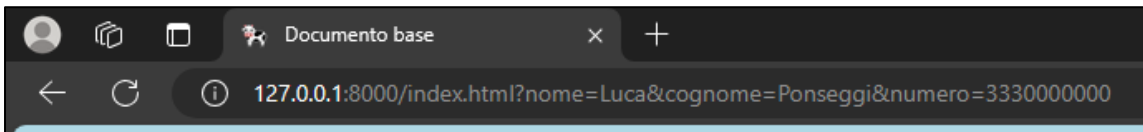
la pagina `index.html` e automaticamente il file `mystyle.css`, l'`immagine.jpg` e `favicon.ico`: la piccola immagine vicino al nome della pagina.

Test del form nella pagina:

Visione client:



nome=
cognome=
numero=
 <-- click



Visione server:

```
127.0.0.1 - - [16/May/2024 01:24:46] "GET /index.html?nome=Luca&cognome=Ponseggi&numero=3330000000 HTTP/1.1" 200 -  
Eseguita GET da ('127.0.0.1', 64536) su file "\index.html", utilizzato mime type "text/html" per header e inviato codice di risposta "200" (OK)  
Sono stati passati questi parametri: {'nome': ['Luca'], 'cognome': ['Ponseggi'], 'numero': ['3330000000']}  
  
127.0.0.1 - - [16/May/2024 01:24:46] "GET /mystyle.css HTTP/1.1" 200 -  
127.0.0.1 - - [16/May/2024 01:24:46] "GET /immagine.jpg HTTP/1.1" 200 -  
Eseguita GET da ('127.0.0.1', 64537) su file "\mystyle.css", utilizzato mime type "text/css" per header e inviato codice di risposta "200" (OK)  
Sono stati passati questi parametri: {}  
  
Eseguita GET da ('127.0.0.1', 64538) su file "\immagine.jpg", utilizzato mime type "image/jpeg" per header e inviato codice di risposta "200" (OK)  
Sono stati passati questi parametri: {}  
  
[]
```

Ovviamente dopo aver inviato il modulo la pagina si aggiorna con i parametri nell'URL, facendo le relative richieste al server delle risorse.

Utilizzo del programma in Python

Visione client:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Luca\Desktop\progettoReti> & C:/Users/Luca/anaconda3/python.exe c:/Users/Luca/Desktop/progettoReti/client.py
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.11.7
Date: Wed, 15 May 2024 23:41:17 GMT
Content-type: text/html

<!DOCTYPE html>
<html lang="it">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="mystyle.css">
    <title>Documento base</title>
  </head>
  <body>
    
    <h1>manda una get al server con il form qui sotto:</h1>
    <form method="get" action="index.html"> <!-- oppure action="index.html" (porta ugualmente lì) -->
      nome=
      <input type="text" name="nome" placeholder="Nome"><br>
      cognome=
      <input type="text" name="cognome" placeholder="Cognome"><br>
      numero=
      <input type="number" name="numero" placeholder="Numero"><br>
      <button type="submit">
        Manda una get al server
      </button>
    </form>
  </body>
</html>
PS C:\Users\Luca\Desktop\progettoReti>
```

Visione server:

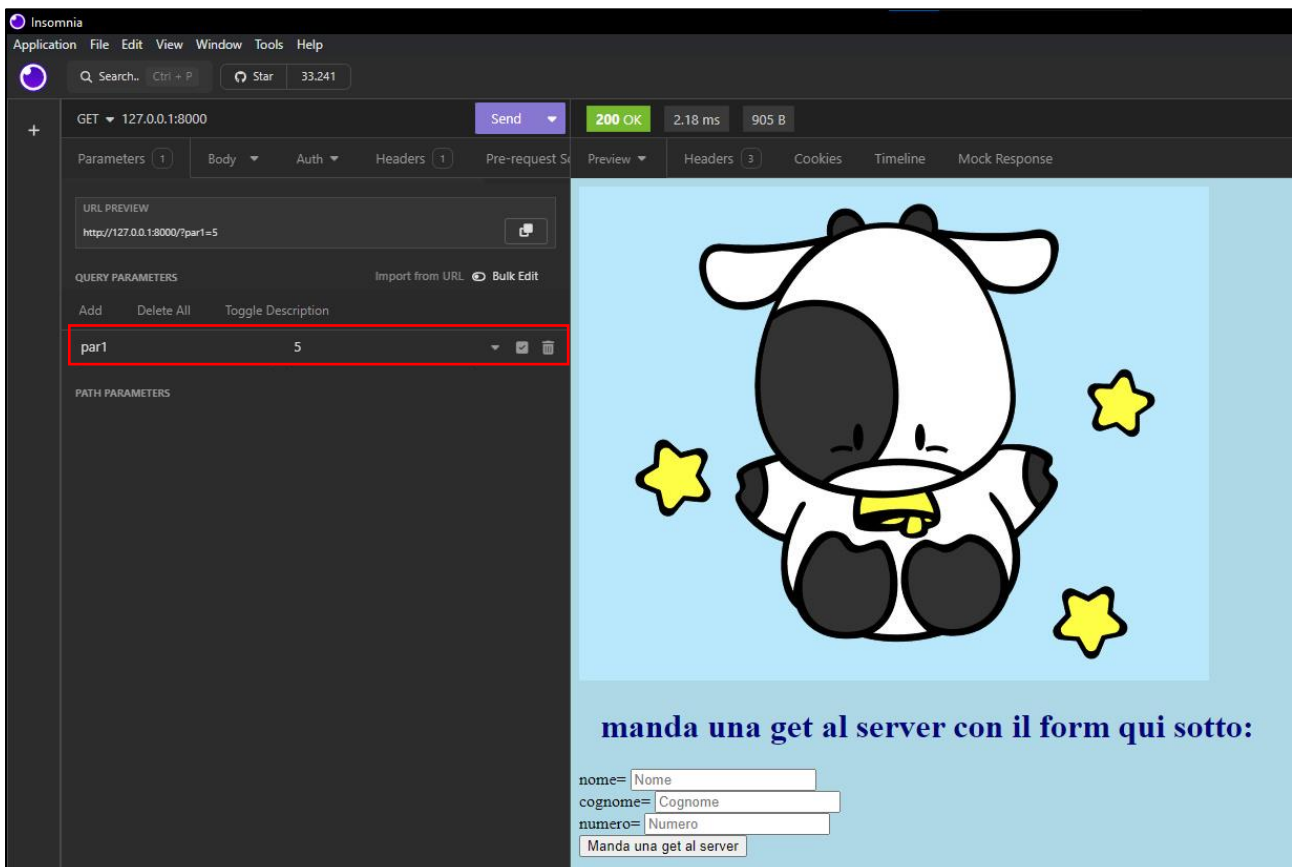
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Luca\Desktop\progettoReti> & C:/Users/Luca/anaconda3/python.exe c:/Users/Luca/Desktop/progettoReti/server.py
Server funzionante su: http://127.0.0.1:8000 ...

127.0.0.1 - - [16/May/2024 01:43:26] "GET /index.html HTTP/1.1" 200 -
Eseguita GET da ('127.0.0.1', 65488) su file "index.html", utilizzato mime type "text/html" per header e inviato codice di risposta "200" (OK)
Sono stati passati questi parametri: {}
```

Utilizzo di Insomnia

La comodità di questo programma è che si possono personalizzare le interrogazioni in modo estremamente facile data l'interfaccia utente molto basilare.

Visione client:



Visione server:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Luca\Desktop\progettoReti> & C:/Users/Luca/anaconda3/python.exe c:/Users/Luca/Desktop/progettoReti/server.py
Server funzionante su: http://127.0.0.1:8000 ...

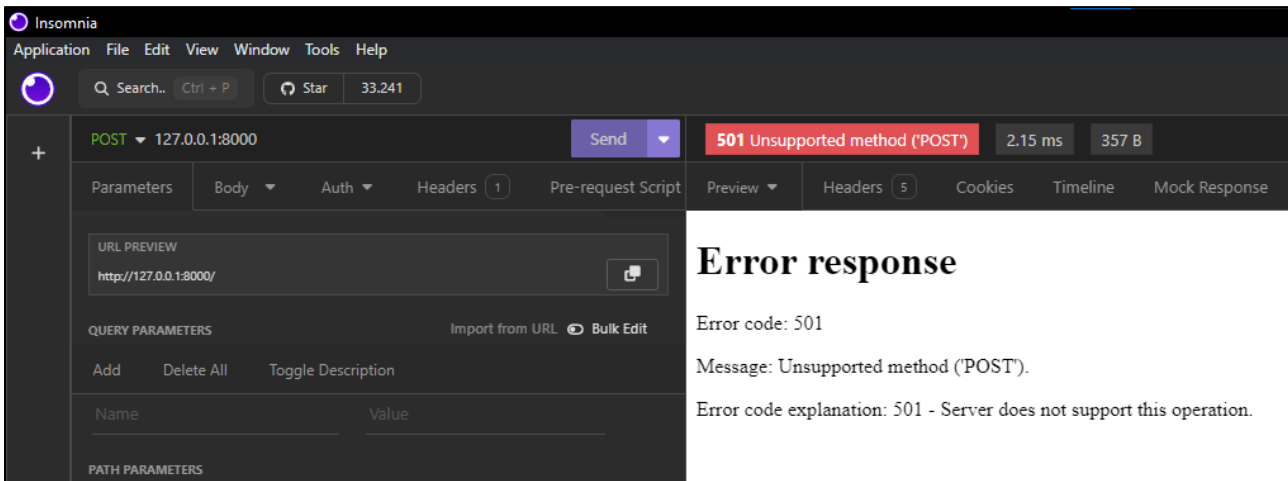
127.0.0.1 - - [16/May/2024 01:52:13] "GET /?par1=5 HTTP/1.1" 200 -
Eseguita GET da ('127.0.0.1', 49443) su file "\index.html", utilizzato mime type "text/html" per header e inviato codice di risposta "200" (OK)
Sono stati passati questi parametri: {'par1': ['5']}

127.0.0.1 - - [16/May/2024 01:52:13] "GET /mystyle.css HTTP/1.1" 200 -
Eseguita GET da ('127.0.0.1', 49444) su file "\mystyle.css", utilizzato mime type "text/css" per header e inviato codice di risposta "200" (OK)
Sono stati passati questi parametri: {}

127.0.0.1 - - [16/May/2024 01:52:13] "GET /immagine.jpg HTTP/1.1" 200 -
Eseguita GET da ('127.0.0.1', 49445) su file "\immagine.jpg", utilizzato mime type "image/jpeg" per header e inviato codice di risposta "200" (OK)
Sono stati passati questi parametri: {}
```

Chiaramente se al server viene fatta una richiesta di tipo POST non funziona in quanto non implementata:

Visione client:



Visione server:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Luca\Desktop\progettoReti> & C:/Users/Luca/anaconda3/python.exe c:/Users/Luca/Desktop/progettoReti/server.py
Server funzionante su: http://127.0.0.1:8000 ...

127.0.0.1 - - [16/May/2024 01:55:17] code 501, message Unsupported method ('POST')
127.0.0.1 - - [16/May/2024 01:55:17] "POST / HTTP/1.1" 501 -
```

Considerazioni aggiuntive

Ho deciso di creare la mia classe di handler ereditando dalla classe

“`http.server.BaseHTTPRequestHandler`” e non da “`http.server.SimpleHTTPRequestHandler`” in quanto quest’ultima, nonostante fosse quella vista in laboratorio, aveva già la funzione `do_GET()` implementata; quindi, non ci sarebbe stata sfida nel completamento del progetto.

Ho notato che in “visione server” di “Test del form nella pagina” non viene fatta richiesta al server di `favicon.ico`, questo perché il browser l’ha già salvata in cache e a meno che non venga fatto un refresh forzato (`CTRL+F5`) della pagina, edge continua ad usare quello delle scorse richieste.