Progetto di Programmazione Reti

Traccia 2

Università di Bologna
Ingegneria e Scienze Informatiche

Autori

Matteo Violani
921109

24 luglio 2021

Indice

0.1 Introduzione

Si vuole creare un web server in linguaggio python che sia in grado di gestire contemporaneamente le richieste dei client connessi senza che vi siano interruzioni di servizio o attese.
Il server una volta interrotto tramite opportuno comando da tastiera o da console deve consentire la terminazione corretta di tutti i thread generati e permettere il riutilizzo della risorsa socket.
I client che si connettono devono visualizzare la homepage della azienda ospedaliera con l'elenco dei servizi forniti ed i loro relativi link alle pagine. Sempre nella pagina principale deve essere presente un link che faccia scaricare un file pdf.
Come funzionalità aggiuntiva si introduce una pagina da amministratore dove è richiesto l'accesso tramite password.

0.2 Descrizione

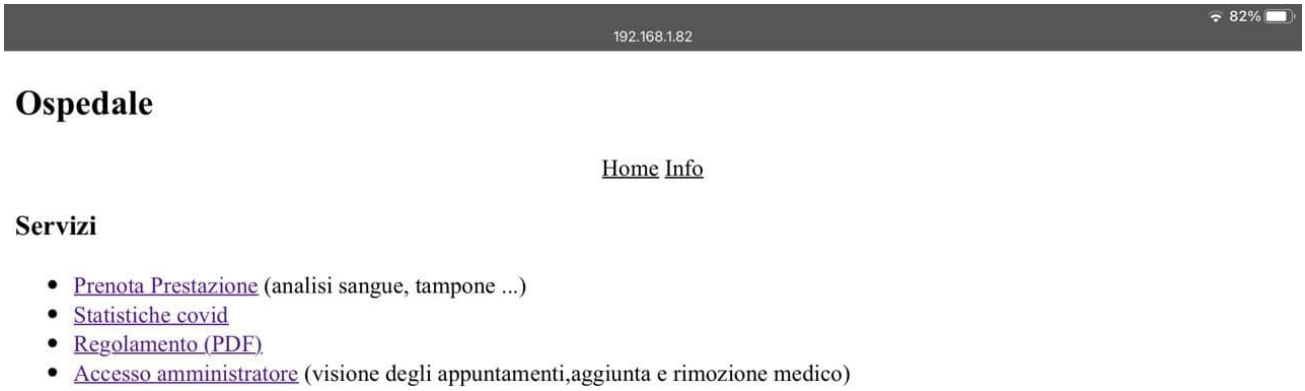

Figura 1: Pagina principale della azienda ospedaliera

Nella pagina principale (Fig.1) sono presenti diversi link oltre a Home e Info:

- ***Prenota Prestazione*** (Fig. 6), in cui è possibile effettuare una prenotazione per un prestazione sanitaria, lasciando la possibilità di scegliere il medico e la tipologia di servizio. È obbligatoria la compilazione dei campi Nome e Cognome. Una volta effettuata la prenotazione si viene reindirizzati ad una pagina di avvenuta registrazione della prenotazione.

- ***Statistiche Covid*** (Fig. 5), mostra una pagina in cui è presente la tabella aggiornata dei dati sulle vaccinazioni Covid-19. I dati vengono prelevati ad ogni caricamento della pagina, direttamente dal repository github della Presidenza del Consiglio dei Ministri, che giornalmente viene aggiornato.

- ***Regolamento(PDF)***, permette lo scaricamento di un file PDF (ipotizzando sia il regolamento dell'ospedale).

- ***Accesso amministratore*** (Fig. 7), in questa pagina è possibile aggiungere e togliere medici, indicando nome e codice medico (per la cancellazione occorre solo il codice). Conclusa la modifica si viene reindirizzati ad una pagina di conferma, in caso contrario si visualizzerà una pagina di errore con un link per ritornare indietro. Dopo ogni modifica occorre, per sicurezza, rieffettuare il login. È presente l'elenco dei medici attualmente presenti del database (file json). In questa pagina sono mostrate anche tutte le prenotazioni effettuate dai pazienti in una tabella. Sono indicati nome e congome del paziente, nome e codice del dottore e servizio richiesto.

  Da notare che le modifiche effettuate sui medici (aggiunta/rimozione) si ripercuote sull'elenco dei medici selezionabile all'atto di prenotazione di una prestazione nella pagina "Prenota Prestazione".

0.3 Implementazione

0.3.1 Librerie usate

Nella implementazione sono stati utilizzati i seguenti moduli python:

- `sys.signal`
- `http.server`
- `socketserver`
- `cgi`
- `json`
- `w3lib.url`
- `wget`

Per l'installazione dei moduli mancanti si consiglia di utilizzare il package manager `pip` con il seguente comando: `pip install [nome modulo]`

0.3.2 Gestione di più client e riuso del socket

Per permettere al server di gestire più client in contemporanea si è dovuto fare uso di più thread, uno per ciascun client connesso. In questo modo il server alla ricezione di richiesta di una nuova connessione crea un nuovo thread che si occuperà unicamente di soddisfare le richieste di quel client. In questo modo il server è libero per accettare nuove connessioni (Fig.2).

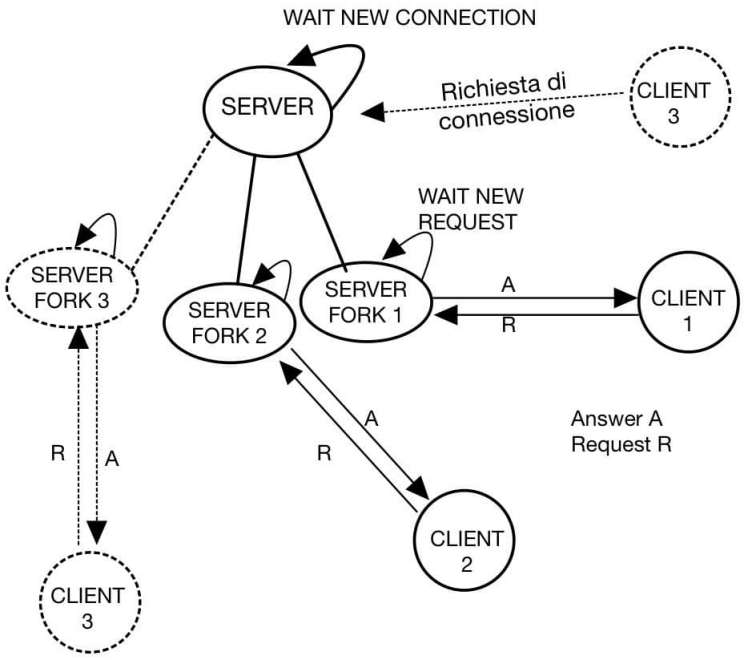

Figura 2: Schema esemplificativo della gestione multithread del server

Per fare ciò si è utilizzata la funzione `ThreadingTCPServer(indirizzo server, request handler)` del modulo `socketserver`. Il campo `indirizzo server` è una tupla `(IP,port)` che specifica l'indirizzo del socket del server che si va ad utilizzare. Se `IP` non viene specificato viene utilizzato di default l'indirizzo `127.0.0.1` oppure `localhost`.
In questo progetto non è stato specificato nessun indirizzo, ad ogni modo se il computer sulla quale è in esecuzione il web-server si trova in una rete LAN, è possibile collegarsi tramite indirizzo IP assegnato dalla rete a quella macchina specificando anche la porta del server.

La porta invece può essere specificata all'avvio del server oppure se omessa verrà usata la `8080`.
`Request handler` è la classe che si occupa di risolvere le richieste dei client. Per questo progetto ci si è appoggiati al `SimpleHTTPRequestHandler` del modulo http.server che spacchetta le informazioni dalla request del client e mette a disposizione diversi metodi per l'invio degli status http, di file (html e non) al client.
Questo handler lascia l'opportunità di personalizzare le azioni di risposta alle request sovrascrivendo i metodi `do_GET` e `do_POST`. Principalmente il codice del server in questo progetto si concentra in questi due metodi.

0.3.3 Metodo do_GET()

Questo metodo del modulo `SimpleHTTPRequestHandler` ha il compito di elaborare una risposta alla richiesta (`request`) del client. Nello specifico questo metodo considera la `request` come una path relativa (*relative path*) alla cartella attuale di lavoro del server di un file.
Nel caso il file non venga trovato o ci sia un qualsiasi errore di I/O viene inviato il codice di stato http 404 e la pagina indicante il codice errore con una breve descrizione (es. "404 File not found").
Ovviamente se il file è presente nella cartella viene restituita la pagina e/o il file con codice stato 200.

Il modo in cui questo metodo elabora le `request` è molto semplicistico per le finalità del progetto e quindi si è proceduto con la sovrascrittura del metodo. In questo modo è stato possibile personalizzare completamente il comportamento del server con determinate richieste.
Osservando il codice in Figura 3 è possibile notare 6 casi possibili di comportamento del server per l'elaborazione della risposta da inviare al client.
I casi in questione riguardano richieste di:
(fra parentesi la riga del codice in questione)

1. (`193`) accesso alla pagina da amministratore
2. (`199`) modifica (aggiunta/rimozione) di un dottore dal database
3. (`205`) pagina relativa alle statistiche covid sulle vaccinazioni
4. (`214`) pagina dedicata alle prenotazioni delle prestazioni sanitarie
5. (`224`) pagina delle informazioni (reindirizzamento alla path corretta)

Se la richiesta non ricade in nessuno dei casi sopra elencati (6° caso, riga `229`), la risoluzione della richiesta viene delegata al metodo standard `do_GET()` inizialmente introdotto.
Si noti che per l'effettiva computazione della richiesta ci si affida a metodi specifici in modo da rendere più leggibile e pulito il codice e per l'invio della pagina richiesta si fa uso dei metodi forniti dal modulo `SimpleHTTPRequestHandler`.

0.3.4 Metodo do_POST()

Come per il metodo `do_GET()` anche questo metodo si occupa di elaborare le richieste ricevute dal client. Nello specifico con la modalità di tipo `POST` si ha che i campi e/o le informazioni vengono inviate in modo "nascosto" all'utente e quindi non presenti all'interno della stringa della `URL`. Si può dedurre quindi che le richieste di questo tipo sono le preferite nel caso si debbano inviare dati sensibili o non si voglia mostrare il contenuto della richiesta all'utente.
Questo metodo come per il precedente è stato sovrascritto per permettere la ricezione e l'elaborazione adeguata dei campi dei diversi form del sito web, che nello specifico sono due:
(tra parentesi la riga del codice in questione, Figura 4)

1. (`277`)form della prenotazione di una prestazione sanitaria
2. (`282`) pagina di login per entrare nella schermata di amministratore

Per l'elaborazione dei campi sono stati scritti metodi apposta per elaborare la richiesta e restituire una risposta corretta.

0.3.5 Metodi ad-hoc per l'elaborazione delle richieste

Non verranno trattati in modo approfondito i metodi utilizzati per soddisfare le `request` ricevute ma più in generale si vuole illustrate l'approccio intrapreso.
Possiamo suddividere in due grandi categorie questi metodi che definiremo di *utility*:

- Per la generazione delle pagine HTML
- Per l'elaborazione campi dei moduli da compilare

La prima categoria (metodi presenti nel file `HTMLdataGenerate.py`) fa maggior uso di variabili di tipo stringhe per la memorizzazione di codice HTML/CSS e di vocabolari per la memorizzazione delle informazioni sui dottori, prenotazioni, dati covid ed orari opportunamente estrapolate dai file json con l'ausilio del modulo `json`. Per l'estrapolazione dei campi dei form inviati con metodo GET è stato utilizzato il metodo `url_query _parameter` del modulo *w3lib.url*.

La seconda invece (all'interno del file `server.py`) fanno uso principale di stringhe per la memorizzazione temporanea dei campi ricevuti e dei metodi del modulo `cgi` per l'estrapolazione degli stessi dalla `request`.

0.4 Istruzioni per l'uso

Per avviare il webserver occorre impartire sul terminale del proprio computer il comando `python3 server.py` dopo aver opportunamente installato i moduli necessari. **Molto importante** non cancellare i file all'interno della cartella `/json` o la cartella stessa.
Le credenziali di accesso alla pagina da amministratore sono:
User: `admin`
Password: `password`

User: `reti`
Password: `esame`
(Per modificare o aggiungere utenti admin occorre modificare il file `login.json` all'interno della cartella `/json`)

```
#in questo metodo si definisce come il server deve agirea richieste di tipo GET
def do_GET(self):
    request = self.path
    print('file REQUESTED: ' + request)
    #se si vuole accedere alla pagina da amministratore prima si viene reindirizzati
    #alla logInPage
    if request == ADMIN_PAGE:
        self.logInPage()
    else:
        #gestione della richiesta di aggiungere un nuovo medico;
        #verifico che la path ricevuta contenga i campi del form che rigurda
        #l'aggiunta del medico
        if (request.__contains__(ADMIN_PAGE) and
            request.__contains__("dottName") and
            request.__contains__("dottCode")):
            print("RICHIESTA Modifica DOTTORi")
            self.editDoctorRequest()
        else:
            if request == COVID_PAGE:
                #il client ha rischiesto la pagina con i dati covid
                self.send_response(200)
                self.end_headers()
                #scarico il file json con i dati delle vaccinazioni in locale
                covidHTML = HTMLgen.refreshCovidData()
                #restituisco la pagina con tab covid aggiornata
                self.wfile.write(bytes(covidHTML, 'utf-8'))
            else:
                if request == PREN_PAGE:
                    #il client ha richiesto la pagina delle prenotazioni
                    #genera la pagina delle prenotazioni a runtime
                    prenHTML = HTMLgen.genPrenotVisita()
                    self.send_response(200)
                    self.end_headers()
                    #invio la pagina della prenotazione al client in risposta
                    self.wfile.write(bytes(prenHTML, 'utf-8'))
                else:
                    #se pagina info richiesta, reindirizzati alla corretta path
                    if request == INFO_PAGE:
                        self.path = INFO_PATH
                    #se non si tratta di nessun caso sopra chiamo il metodo del
                    #modulo per risolvere la richiesta e restituire il file
                    #, se presente nella path, corrispondente all 'url digitato
                    return http.server.SimpleHTTPRequestHandler.do_GET(self)
```

Figura 3: Implementazione del metodo do_GET per soddisfare richieste azienda ospedaliera

```
273    def do_POST(self):
274        print("POST path: "+self.path)
275        #con l'if vado a distiguere i casi dei due form, quello delle prenotazini
276        #da quello utilizzato per il login dell'admin
277        if (self.path == PRENOT_PAGE):
278            self.doPrenotazione()
279
280        else :
281            #RICHIESTA (POST) LOGIN PAGINA ADMIN
282            if (self.path == ADMIN_PAGE):
283                print("\nlogin_POST")
284                self.checkLogin()
285
286            else :
287                self.send_error(404, 'Richiesta non gestita')
```

Figura 4: Implementazione del metodo do_POST

192.168.1.82

81%

Ospedale -> Statistiche Vaccinazioni Covid

Home Info

Ultimo aggiornamento:

2021-07-23T00:00:00.000Z

index	area	somministrate	consegnate	%	codice_NUTS1	codice_NUTS2	codice_regione_ISTAT	regione
0	ABR	1409861	1495024	94.3	ITF	ITF1	13	Abruzzo
1	BAS	565601	622165	90.9	ITF	ITF5	17	Basilicata
2	CAL	1856840	2100281	88.4	ITF	ITF6	18	Calabria
3	CAM	6207899	6718819	92.4	ITF	ITF3	15	Campania
4	EMR	4717886	5120153	92.1	ITH	ITH5	8	Emilia-Romagna
5	FVG	1274870	1387437	91.9	ITH	ITH4	6	Friuli-Venezia Giulia
6	LAZ	6476590	6868713	94.3	ITI	ITI4	12	Lazio
7	LIG	1593421	1789062	89.1	ITC	ITC3	7	Liguria
8	LOM	11361799	11792126	96.4	ITC	ITC4	3	Lombardia
9	MAR	1626866	1698946	95.8	ITI	ITI3	11	Marche
10	MOL	329559	353985	93.1	ITF	ITF2	14	Molise
11	PAB	519708	586118	88.7	ITH	ITH1	4	Provincia Autonoma Bolzano / Bozen
12	PAT	536115	592860	90.4	ITH	ITH2	4	Provincia Autonoma Trento
13	PIE	4549786	4820150	94.4	ITC	ITC1	1	Piemonte
14	PUG	4430023	4695393	94.3	ITF	ITF4	16	Puglia
15	SAR	1718825	1853768	92.7	ITG	ITG2	20	Sardegna
16	SIC	4766567	5154005	92.5	ITG	ITG1	19	Sicilia
17	TOS	3740091	4166925	89.8	ITI	ITI1	9	Toscana
18	UMB	929085	1018961	91.2	ITI	ITI2	10	Umbria
19	VDA	126296	137140	92.1	ITC	ITC2	2	Valle d'Aosta / Vallée d'Aoste
20	VEN	5141703	5534415	92.9	ITH	ITH3	5	Veneto

Figura 5: Pagina "Statistiche Vaccinazioni Covid" visualizzata dall'utente

Figura 6: Form visualizzato dall'utente per prenotare

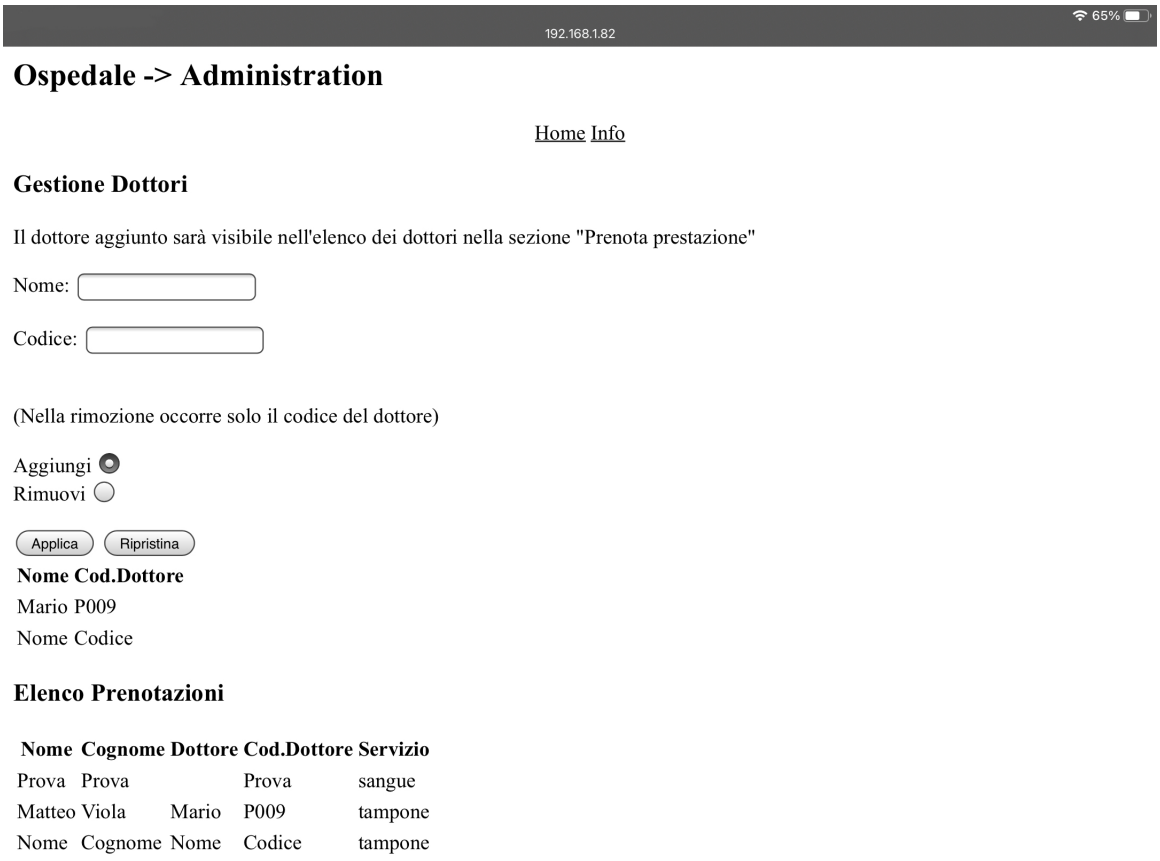

Figura 7: Pagina da amministratore