Relazione per il corso di

Programmazione di Reti

A.A. 2020-2021

Traccia 2: Python Web Server

Studente: Foschi Andrea (andrea.foschi13@studio.unibo.it)

Matricola: 0000933487

Università degli studi di Bologna

Campus di Cesena

Facoltà di ingegneria e scienze informatiche

INDICE

1 – Introduzione

Si immagini di dover realizzare un Web Server in Python per una azienda ospedaliera. I requisiti del Web Server sono i seguenti:

• Il web server deve consentire l'accesso a più utenti in contemporanea

• La pagina iniziale deve consentire di visualizzare la lista dei servizi erogati dall'azienda ospedaliera e per ogni servizio avere un link di riferimento ad una pagina dedicata.

• L'interruzione da tastiera (o da console) dell'esecuzione del web server deve essere opportunamente gestita in modo da liberare la risorsa socket.

• Nella pagina principale dovrà anche essere presente un link per il download di un file pdf da parte del browser

• Come requisito facoltativo si chiede di autenticare gli utenti nella fase iniziale della connessione.

2 – Avviare l'applicazione

Lanciare il comando *python webServer.py [numeroPorta]* all'interno della cartella **src**. Al posto di *[numeroPorta]* è possibile passare il numero di porta su cui il server si metterà in ascolto; se non viene passato nulla, assumerà di default il valore 8080.
Per lanciare il server di autenticazione basato su Flask, lanciare il comando *python authentication.py*. In questo caso non accetta nessun parametro, bensì utilizza univocamente la porta 8080. È possibile lanciare gli applicativi anche su Spyder. Per il web server classico, andare su *Run->Configuration per file..->Command line options:* se si vuole passare un valore per la porta da riga di comando.

3 – Descrizione dell'applicazione

Il web server standard definisce al suo interno un handler personalizzato definito dalla classe *CustomHandler*. Esso ridefinisce i metodi *do_HEAD() e do_GET()* della classe *http.server.SimpleHTTPRequestHandler*, per fare in modo di trattare correttamente i file contenuti all'interno del server che possono essere richiesti. Il metodo *find_file()* lavora con i percorsi dei file e gestisce la loro apertura. La struttura dati *files* contiene al suo interno un insieme di tuple composte da due elementi: il primo fa riferimento al nome del file, il secondo al rispettivo percorso relativo.

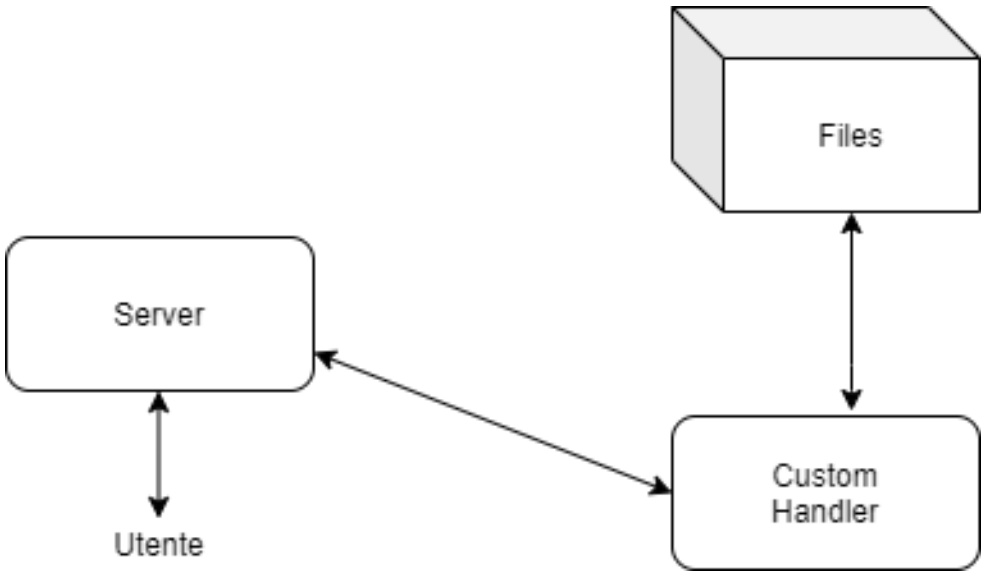
```
files = [("chisiamo.html","html/chisiamo.html"),
         ("contatti.html", "html/contatti.html"),
         ("", "html/index.html"),
         ("index.html", "html/index.html"),
         ("prenotazioni.html","html/prenotazioni.html"),
         ("servizi.html", "html/servizi.html")]
```

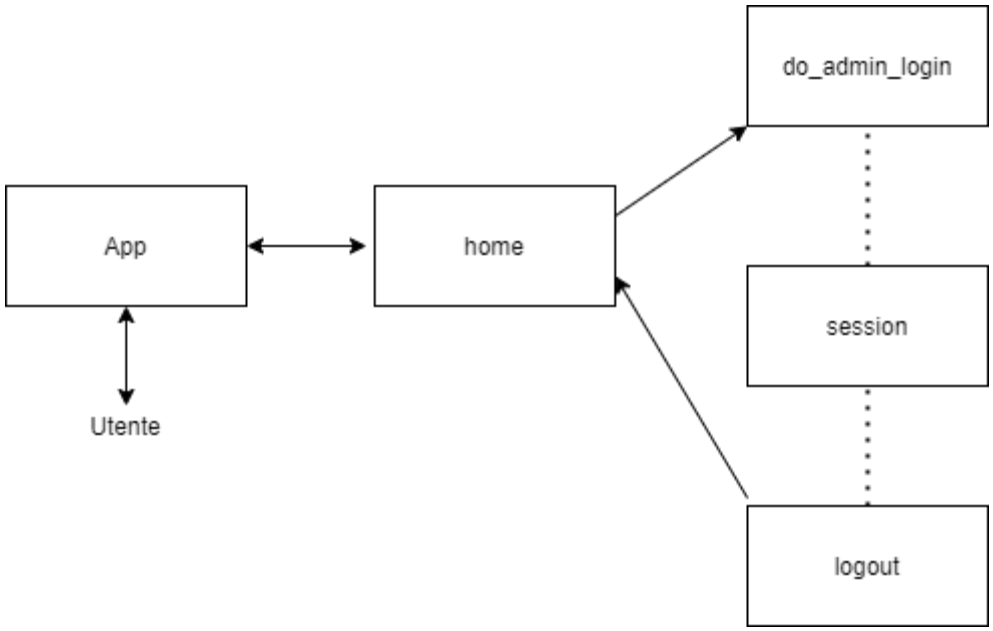
In questo server le operazioni di login e logout non funzionano, in quanto non sono riuscito ad integrare quest'applicativo con l'altro che utilizza Flask.

Il web server di autenticazione, invece, non è altro che il complementare di quello appena visto, infatti funzionano perfettamente le operazioni di login e logout ma le pagine non vengono caricate poiché viene già fatto dall'altro server. Per poter essere utilizzato Flask è necessario prima installarlo lanciando da terminale *pip install Flask*. L'app sa dove andare a ripescare le varie richieste tramite le annotazioni *@app.route(percorso, metodiAccettati)*. L'app per poter funzionare necessita di settare il valore *secret_key* di 12 caratteri, che io ho inizializzato mediante la funzione os.urandom per fornire così un valore casuale. Tramite la funzione *render_template* è possibile modificare il percorso a cui l'applicazione fa riferimento. Per far si che l'autenticazione sia coerente, Flask fa uso delle **sessioni**, in questo modo è possibile ricordare se un utente si è precedentemente loggato o meno; l'applicazione in questione accetta solo un utente, con username **admin** e password **admin**. Per poter verificare che i parametri siano corretti, la funzione *do_admin_login()* accetta una chiamata POST (così da non mostrare in chiaro username e password sulla url) che viene direttamente chiamata dal file *login.html* il quale contiene un riferimento per esso. Se le credenziali sono corrette la sessione viene aggiornata, in caso contrario viene mostrato un messaggio di errore. Una volta fatto il login ed essere così su *index.html*, si può solamente effettuare il logout cliccando sull'apposito riferimento; viene dunque mostrato che il logout è andato a buon fine.

4 – Diagrammi

Vengono ora mostrati i diagrammi relativi alle componenti dei server.





5 – Librerie usate

- http.server;
- socketserver;
- sys;
- signal;
- flask (Flask, flash, render_template, request, session);
- os.

6 – Crediti

Per visualizzare il codice sorgente e tutto il resto fare riferimento a: https://github.com/andreafoschi00/ProgettoReti