Web Server

Relazione di progetto di 'Programmazione di reti'

Tassinari Luca • Matr. 921373

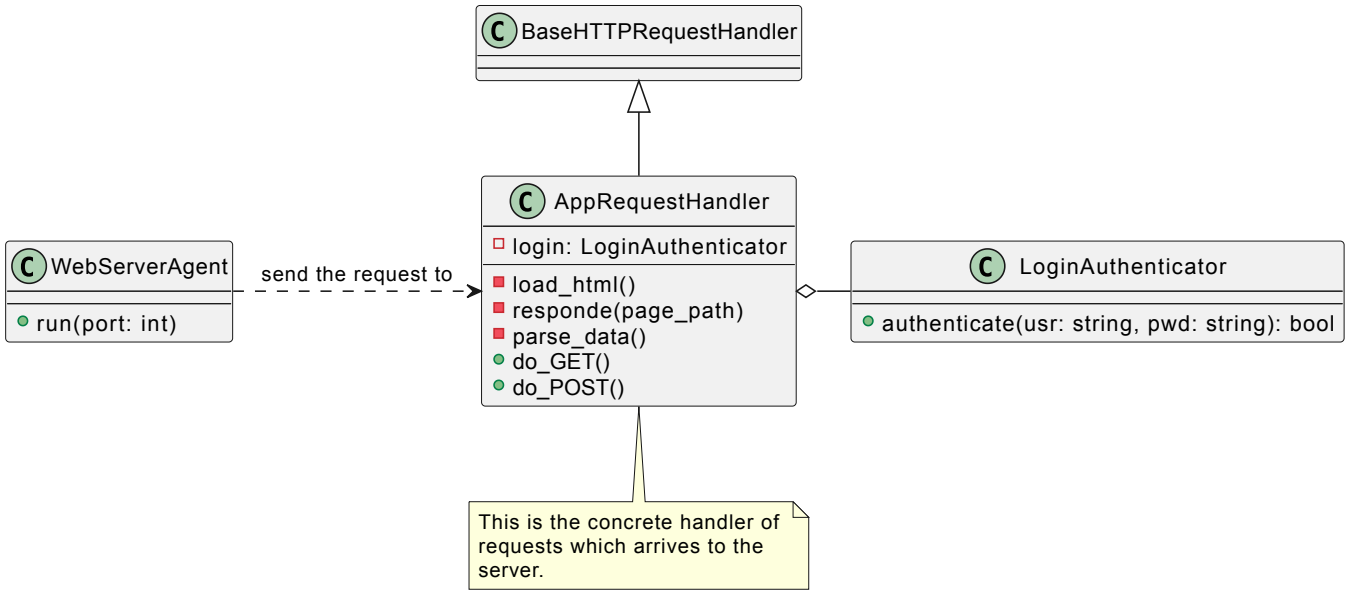
1. Analisi dei requisiti

Si vuole realizzare un *web server* per un'agenzia di viaggi. Di seguito sono elencati per punti i requisiti del sistema.

- il *web server* deve consentire l'accesso a più utenti in contemporanea;
- la *home page* del sito deve permettere la visualizzazione della lista di servizi erogati dall'agenzia viaggi (con relativo *link* ad una pagina dedicata);
- devono esserci la possibilità d'inserire *link* per il *download* di documenti pdf;
- si richiede la possibilità di autenticare gli utenti;
- l'interruzione da tastiera dell'esecuzione del *web server* deve essere opportunamente gestita in modo da liberare la risorsa *socket*.

2. Design

L'architettura del sistema è molto semplice ed è presenta qui di seguito.

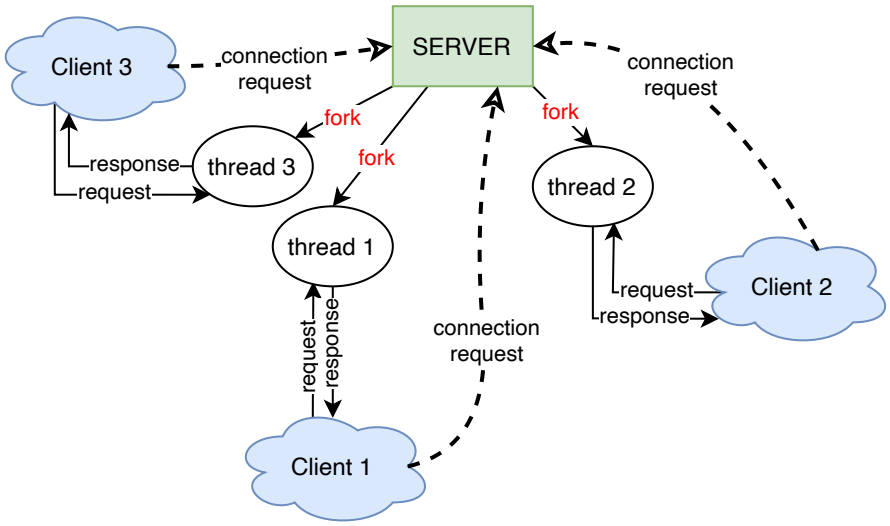


- `WebServerAgent` è il componente attivo che si occupa dell'istanziazione e configurazione del server web.

- `AppRequestHandler` è la classe che si occupa della gestione delle richieste HTTP che arrivano al server web dai vari *client*. Si noti che questa classe estende `BaseHTTPRequestHandler`, definita all'interno del modulo `http.server`, implementando la logica delle risposte all'interno dei due metodi `do_GET()` e `do_POST()`.
- `LoginAuthenticator`: classe che si occupa dell'autenticazione all'area riservata del sito web.

2.1. Design dettagliato

Per permettere al server di gestire più client in contemporanea è necessario fare uso di più *thread*, uno per ciascun *client* che si connette: in particolare, per ogni *client* che si connette al *web server*, il *server* crea un nuovo *thread* il cui compito è quello di rispondere al *client*; una volta esaurito il suo compito, termina.



A tale scopo si fa uso della funzione `ThreadingTCPServer` per gestire più richieste all'interno del modulo `http.server` e a cui viene passato il concreto *handler* che dovrà gestire le richieste di ciascun *client* (nel caso specifico `AppRequestHandler`).

I due metodi `do_GET()` e `do_POST()` definiscono la logica di gestione per le richieste, rispettivamente, GET e POST.

In generale, le richieste GET vengono gestite controllando che la risorsa richiesta esista, restituendola laddove presente oppure restituendo una pagina di *default* di errore (`not-found.html`).

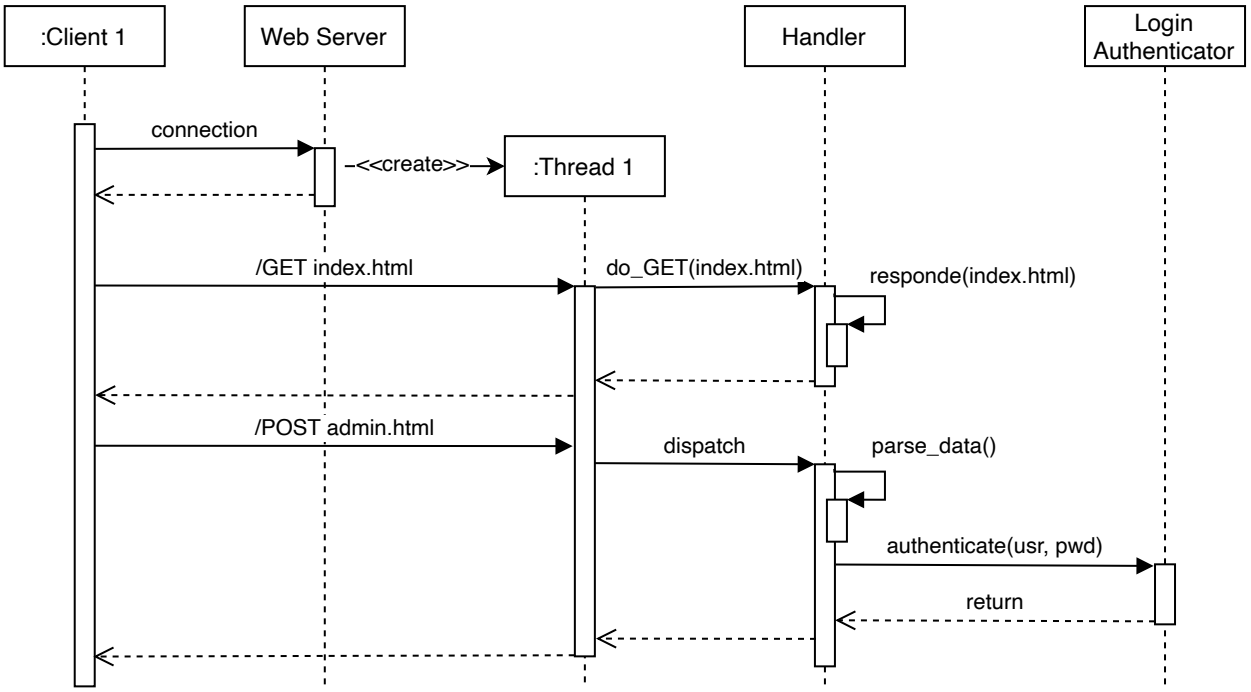
La richiesta POST è implementata per effettuare l'autenticazione al sito: quando l'utente compila ed effettua il *submit* del *form* viene inviata una richiesta POST. Si ricorda che nella richiesta POST le informazioni sono incapsulate all'interno del *body* della richiesta HTTP e non direttamente visibili nell'URL. Questo, tuttavia, **non** significa che la trasmissione dei dati è sicura in quanto è in chiaro (e potrebbe pertanto essere intercettata)!

La gestione della richiesta POST è, anch'essa, piuttosto semplice: viene fatto il parsing per ottenere i dati di autenticazione a seconda del tipo di *encoding* con cui i dati sono stati inviati al server (`application/x-www-form-urlencoded` (default) in cui tutti i caratteri vengono codificati prima dell'invio o `multipart/form-data` se vengono inviati *file*) e se la pagina *target* della richiesta è quella di amministratore, attraverso il `LoginAuthenticator`, viene effettuato un nuovo tentativo di accesso.

`LoginAuthenticator` effettua banalmente il controllo della corrispondenza tra le credenziali immesse dall'utente nel *form* e quelle presenti nel sistema (recuperandole dal file `login.json` - approccio

sicuramente non sicuro!).

Di seguito un diagramma di sequenza che mostra un esempio di possibile interazione con il *web server*:



Per quanto riguarda la richiesta di poter scaricare file PDF, è sufficiente aggiungere nell'HTML un link alla risorsa specificata (in locale).

3. Librerie utilizzate

Nell'implementazione sono state usati i seguenti moduli:

- `socketserver` per la gestione multi-thread del *web server*;
- `http` per l'*Handler* delle richieste;
- `sys` per gli argomenti da riga di comando;
- `cgi` per il parsing degli argomenti passati nel body della richiest POST HTTP;
- `json` per l'estrazione delle credenziali (scritte in file JSON).

4. Come avviare l'applicazione?

Semplicemente (dalla *route* del progetto):

```
python3 ./app.py [port_number]
```

dove `port_number` è il numero della porta su cui il server rimane in ascolto. Se non specificato il default è `8080`.