

Documentazione Sined Travels

Denis Caushaj

denis.caushaj@studio.unibo.it
0900069361

Indice

1	Introduzione	1
2	Analisi	1
3	Progettazione	2
3.1	Front-end (HTML e CSS)	2
3.2	Back-end (Python)	3

1 Introduzione

Il progetto si pone come obiettivo la gestione del web server di Sined Travels, un'agenzia di viaggi che si occupa della prenotazione di voli aerei e di alloggi in strutture convenzionate.

2 Analisi

Il web server deve essere in grado di:

- garantire il corretto funzionamento di tutte le pagine associate, dal punto di vista della visualizzazione e delle richieste fatte, ovvero deve consentire di visualizzare la lista dei servizi erogati tramite link di riferimento;
- gestire l'accesso in contemporanea di più utenti;
- gestire il download di file;
- permettere l'autenticazione degli utenti al sito;
- l'interruzione da tastiera (o da console) dell'esecuzione del web server deve essere opportunamente gestita in modo da liberare la risorsa socket.

3 Progettazione

Per la costruzione del web server e del sito sono stati utilizzati i linguaggi HTML e CSS per lo sviluppo del front-end e Python per lo sviluppo del back-end. In particolare è stata utilizzata la libreria di Python "Flask" che permette la gestione di un web server e fornisce features quali liberare la risorsa socket durante la chiusura del programma e la gestione di richieste in parallelo.

3.1 Front-end (HTML e CSS)

Essendo la parte di back-end la più importante ai fini di questo progetto, alla parte di front-end è stato dedicato il giusto tempo per renderne la User Experience più piacevole.

Esempi di tag generalmente utilizzati per la realizzazione delle pagine HTML sono:

- head: definisce i metadati della pagina;
- body: definisce il corpo della pagina;
- div: permette dividere le aree all'interno della pagina;
- table: permette la creazione di tabelle;
- menu: permette di generare menu (in particolare in questo progetto viene combinato con codice Python che carica i nomi della pagine e i relativi link);
- footer: contiene informazioni sulla sezione che le contiene.

Per caricare le pagine HTML sono stati utilizzati i template di Flask. Grazie a questi è stato possibile definire dei layout in grado di gestire i dati in base agli stati delle pagine, utili per evitare ripetizioni di codice.

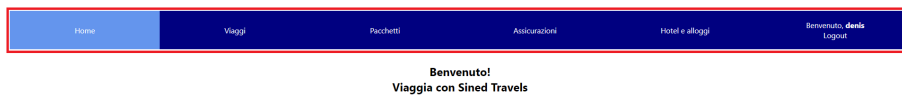


Figura 1: Immagine del menu con l'utilizzo dei template

```
{% for title, url in menu_entries %}
<li><a class="{% if url == request.path %}active{% endif %}" href="{{url}}"><span>{{title}}</span></a></li>
{% endfor %}
```

Figura 2: Esempio di implementazione dei template

3.2 Back-end (Python)

Per realizzare il back-end, come detto in precedenza è stato utilizzato Python, in particolare è stato rilevante l'utilizzo della libreria "Flask".

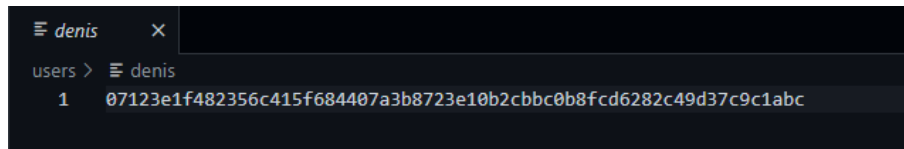
E' interessante notare la gestione delle pagine di registrazione e di accesso: l'implementazione permette di salvare i dati dell'utente senza l'utilizzo di un database (scelta progettuale) ma utilizzando solo dei file all'interno dell'apposita cartella (`./users`).

In `./users` ogni volta che un utente si registra, viene aggiunto un file di testo denominato come lo username, contenente l'hash (SHA256) della password, utile per non salvare la password in chiaro. Poiché le codifiche di alcune parole "hashate" sono note (basta scrivere sul proprio motore di ricerca una parola qualsiasi "hashata" per scoprire che molto probabilmente è messa in chiaro), alla password iniziale viene sommata una stringa segreta, generando così una nuova password priva di senso, poi "hashata", rendendo così più sicuri i dati salvati.

E' importante fare un piccolo appunto sul codice che permette di fare l'hashing della password, in cui sulla stringa viene richiamato il metodo "encode" e poi sull'hash "hexdigest". Grazie al metodo encode è possibile trasformare la stringa in semplici bytes, poiché il metodo sha256 richiede dei byte; mentre il metodo hexdigest permette di avere una versione in esadecimale del risultato della funzione precedente.

```
def hash(s):  
    return sha256(s.encode()).hexdigest()
```

Figura 3: Funzione hash



```
denis x  
users > denis  
1 07123e1f482356c415f684407a3b8723e10b2cbbc0b8fcd6282c49d37c9c1abc
```

Figura 4: Hash applicato ad una password

Le funzioni `signin` e `login` sono le più interessanti anche dal punto di vista delle reti, poiché gestiscono sia richieste GET che POST, a differenza delle altre che gestiscono solo richieste GET.

Le richieste POST che vengono effettuate hanno come risposta da parte del server 302, ovvero il code status per il redirect. In questo caso utilizzato per cambiare pagina dopo l'accesso o la registrazione di un utente. Contestualmente al code 302, viene settato il response header "Location" per specificare la pagina

a cui fare il redirect. In questo caso specifico "/" (homepage) per l'accesso e "/login" per la registrazione.

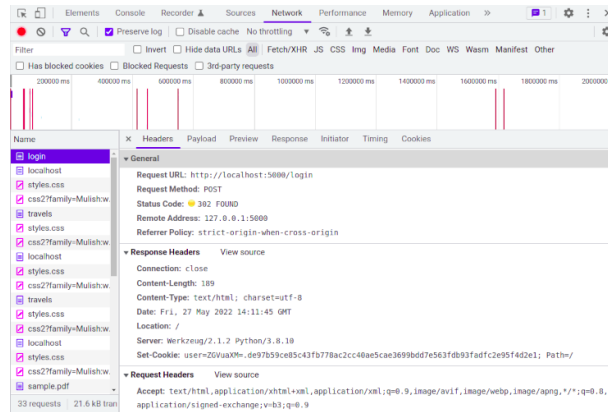


Figura 5: Esempio di richiesta POST e di reindirizzamento

Altri code status che vengono restituiti dal server possono essere:

- HTTP 200, code status per indicare che la richiesta è stata fatta con successo;
- HTTP 304, code status per indicare il download di un file già presente nella cache del sito, quindi senza il bisogno di dover ritrasmettere lo stesso file;
- HTTP 404, nel caso si stesse cercando di accedere ad una pagina inesistente.

Da codice, inoltre, ogni volta che viene effettuato un accesso vengono settati i cookie, come si può vedere sempre nei response headers.

```
resp.set_cookie([USER_COOKIE, encode_user_cookie(username)])
```

Figura 6: Esempio di set_cookie da codice

```
def encode_user_cookie(username: str) -> str:
    return f"{b64encode(username.encode()).decode()}.{hash_with_secret(username)}"

def decode_user_cookie(cookie: str) -> str:
    username_b64, secret_hash = cookie.split(".")
    return b64decode(username_b64).decode(), secret_hash
```

Figura 7: Codice che permette la generazione dei cookie

Per memorizzare il login di un utente anche dopo aver cambiato pagina o chiuso e riaperto il browser, il sito fa uso di cookie.

Il cookie viene utilizzato per memorizzare quale utente ha fatto l'accesso. Per evitare che un malintenzionato possa fingersi un utente diverso da quello che è, modificando il nome nel cookie, oltre allo username viene memorizzato anche un hash dello username con un segreto che solo il server è in grado di generare. In questo modo il server può sempre verificare l'integrità del cookie confrontando l'hash inviato dal client con quello generato.

Più tecnicamente, ogni volta che viene effettuato il login il server aggiunge ai suoi response headers "Set-Cookie" con il contenuto del cookie "user" da lui generato. Questo fa sì che a ogni richiesta successiva, il browser invii questo cookie nei suoi request headers e in questo modo il server è sempre in grado di conoscere il nome dell'utente loggato.

I cookie vengono codificati utilizzando la funzione "b64encode" che sfrutta l'alfabeto base64, che in questo specifico caso ovvia il problema dei punti (".") che potrebbero essere presenti nel cookie, poiché base64 non li contiene nel proprio alfabeto. Ciò è utile poiché lo username e l'hash del cookie sono divisi da un punto (come si può vedere nella Figura 6).

Il cookie viene salvato fino a quando l'utente con effettua il logout. In tal caso il cookie viene settato come stringa vuota, scadendo immediatamente, inoltre l'utente viene reindirizzato alla pagina di login.

```
@app.route("/logout", methods=["GET"])
def logout():
    resp = make_response(redirect("/login"))
    resp.set_cookie(USER_COOKIE, "", expires=0)
    return resp
```

Figura 8: Funzione di logout

I metodi di reindirizzamento sono probabilmente quelli meno interessanti dal punto di vista del codice, poiché effettivamente il loro compito è semplicemente quello di caricare le pagine HTML richieste (purché sia stato fatto l'accesso, altrimenti l'utente verrà reindirizzato alla pagina di login).

```
@app.route("/stay", methods=["GET"])
def stay():
    return check_login_and_render_template("stay.html")
```

Figura 9: Esempio di funzione di caricamento di una pagina

```
def check_login_and_render_template(template):
    username = login_username()
    if username is None:
        return redirect("/login")
    return render_template(template, menu_entries=MENU, username=username)
```

Figura 10: Funzione che controlla l'accesso dell'utente

Infine, l'ultima funzione implementata è quella di download di un file (in questo caso un file PDF contenente questa documentazione stessa), in cui oltre al percorso da cui prendere il file, indica inoltre al browser di scaricare il file piuttosto che aprirlo in un'altra pagina settando il parametro `as_attachment` a `True`.

```
@app.route("/static/RelazioneReti.pdf", methods=["GET"])
def download():
    return send_file("static/RelazioneReti.pdf", as_attachment=True)
```

Figura 11: Funzione per i download di file