WebServer
Relazione di progetto
Programmazione Di Reti

Mazzoli Alessandro

3 giugno 2021

Indice

Capitolo 1

Analisi

Si vuole sviluppare un WebServer per la gestione del sito web di un'azienda ospedaliera per esporre i propri servizi online in modo da ridurre il traffico di persone presso la sede fisica causa Covid-19.
Dalla piattaforma sarà possibile visualizzare gli orari dell'ospedale e le info quali i contatti telefonici per i vari reparti dell'ospedale.
Sarà data possibilità all'utente di prenotare una visita tramite il sito web in modo da non saturare le linee telefoniche che si vogliono lasciare libere in caso di emergenze. Questa prenotazione avverrà tramite la compilazione di un form che richiederà i dati anagrafici dell'utente e restituirà l'identificativo della sua visita.
Quest'ultimo codice andrà poi inserito nell'apposita pagina dove è possibile vedere i risultati degli esami una volta che sono pronti. Nel caso si vada ad inserire il codice di un esame il cui esito non è stato ancora pubblicato l'utente verrà avvisato che il risultato non è ancora pronto.
Tramite il sito web dovrà essere possibile anche l'accesso alla sezione di admin mediante l'inserimento delle credenziali che fornirà accesso alla sezione dedicata alla gestione delle visite e degli esiti, tramite il quale la segreteria potrà pubblicare l'esito di una determinata visita oppure aggiungere dei contatti telefonici.

1.1 Requisiti

- 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.
- Dovrà essere possibile gestire non solo richieste di tipo GET ma anche altri tipi quali le POST.
- Possibilità di prenotazione di una visita.
- Accesso riservato alla segreteria per la pubblicazione degli esiti.
- Accesso riservato alla segreteria per l'aggiunta/modifica dei contatti telefonici

Capitolo 2

Design

2.1 Architettura

La gestione architetturale dell'applicazine è ispirata al framework Django e al framework Flask.
Essa si basa su un thread principale che accetta richiesta HTTP e per ognuna di queste genera un nuovo thread per la sua gestione, così da avere la possibilità di gestire più richieste contemporaneamente.
Una volta generato il thread la richiesta verrà quindi passato all' Handler che si occuperà di analizzarla ed estrarre da essa i dati quali url, parametri, metodo della richiesta, etc...
Una volta stabilito l'url a cui è stata fatta la richiesta potrà controllare se esiste una View, ovvero una funzione il cui scopo è quello di rispondere ad una richiesta a quel determinato url, in caso di successo allora la richiesta verrà passata a quella View mentre in caso non sia presente verrà mandata una risposta al client con codice 404 - Not Found e la relativa pagina di errore per fargli sapere che non la pagina richiesta non esiste.
Sarà quindi compito della rispettiva View definire il comportamente di gestione per quel determinato url, scorporando quindi l'Handler dall'effettiva funzione da eseguire per una determinata richiesta.
Inoltre così facendo abbiamo creato un webserver dinamico che non si occupa solo di restituire delle pagine html dato uno specifico url ma ad ogni chiamata potranno essere effettuate operazioni quali la verifica di credenziali, la lettura di dati da un eventuale database, etc...

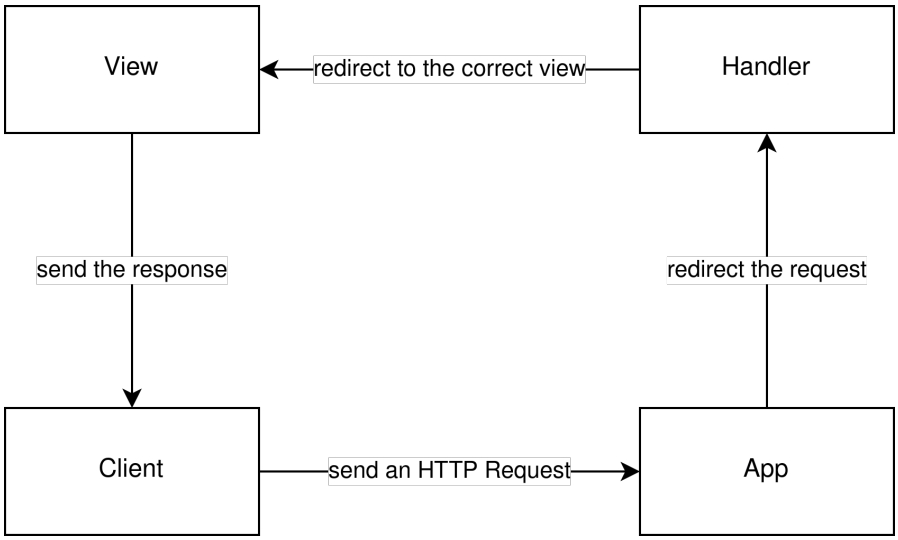

Figura 2.1: Gestione architetturale delle richieste

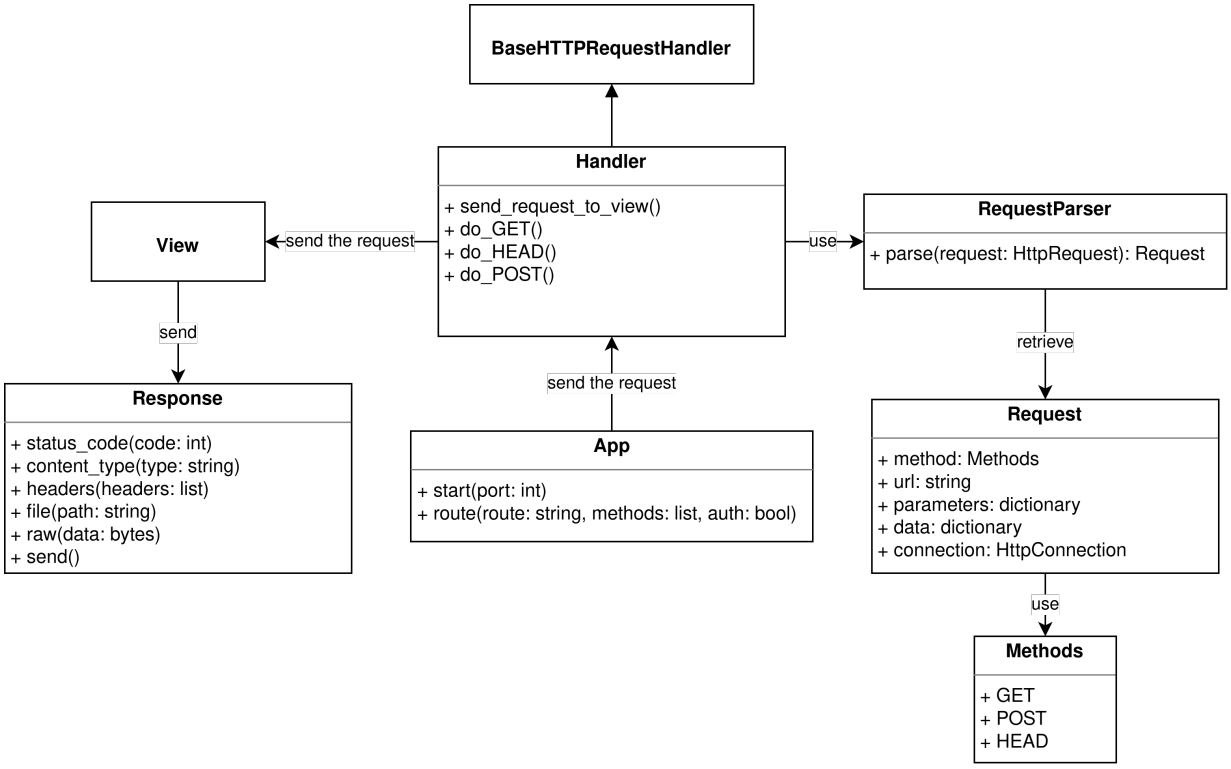

Figura 2.2: Schema UML delle entità dell'applicazione

2.2 Threads

Per quanto riguarda la gestione dei threads abbiamo dal lato client il browser tramite il quale farà le richieste al web server, che considereremo come un singolo thread X, dove X indica il numero della richiesta.
Il thread del client X quindi farà una richiesta HTTP al main thread del webserver, il quale si occuperà di generare un nuovo thread X il cui compito sarà quelli di gestire la richiesta del client X.
Lato server abbiamo quindi un thread principale sempre attivo, mentre in presenza di N client che stanno facendo una richiesta avremo ulteriori N thread dedicati a gestire le richieste dei clients.

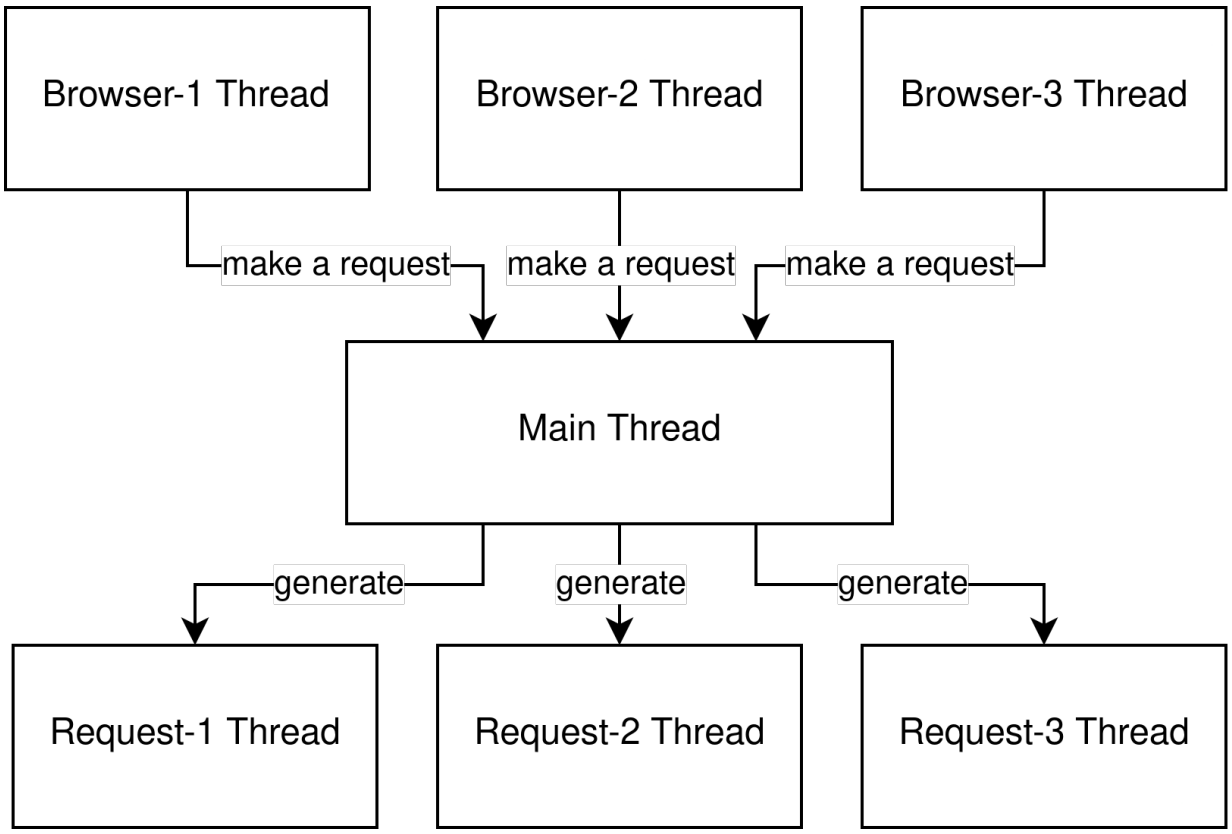

Figura 2.3: Schema dei threads dell'applicazione sia a livello client che a livello server

2.3 Views & Routes

Le pagine sono gestite tramite i concetti di View e di Route.
Una View consiste in una funzione il cui compito è quello di gestire una richiesta un determinato url. Questa funzione prenderà come parametro la richiesta e poi dopo avere fatto ciò che implica la chiamata a quella funzione invierà una risposta a chi ha fatto la richiesta.
Le Routes invece consistono nel mapping di un url ad una determinata View. Questo è ottenuto tramite l'utilizzo di un dizionario dentro la classe App nel quale al caricamento dell'applicazione saranno aggiunte tutte le view presenti associandole al rispettivo url.
Per la creazione di una view ci basterà creare una funziona e marchiarla con il decoratore @app.route specificando:

- l'url a cui è associata
- i metodi permessi per quell'url (GET, POST, HEAD, etc...)
- se è richiesta o meno l'autenticazione di amministratore

Queste funzioni andranno dentro la cartella 'views' che all'avvio dell'applicazione verrà analizzata e per ogni file python verranno cercate le funzioni marchiate con il decorator @app.route e verrà costruito così il dizionario di mapping.

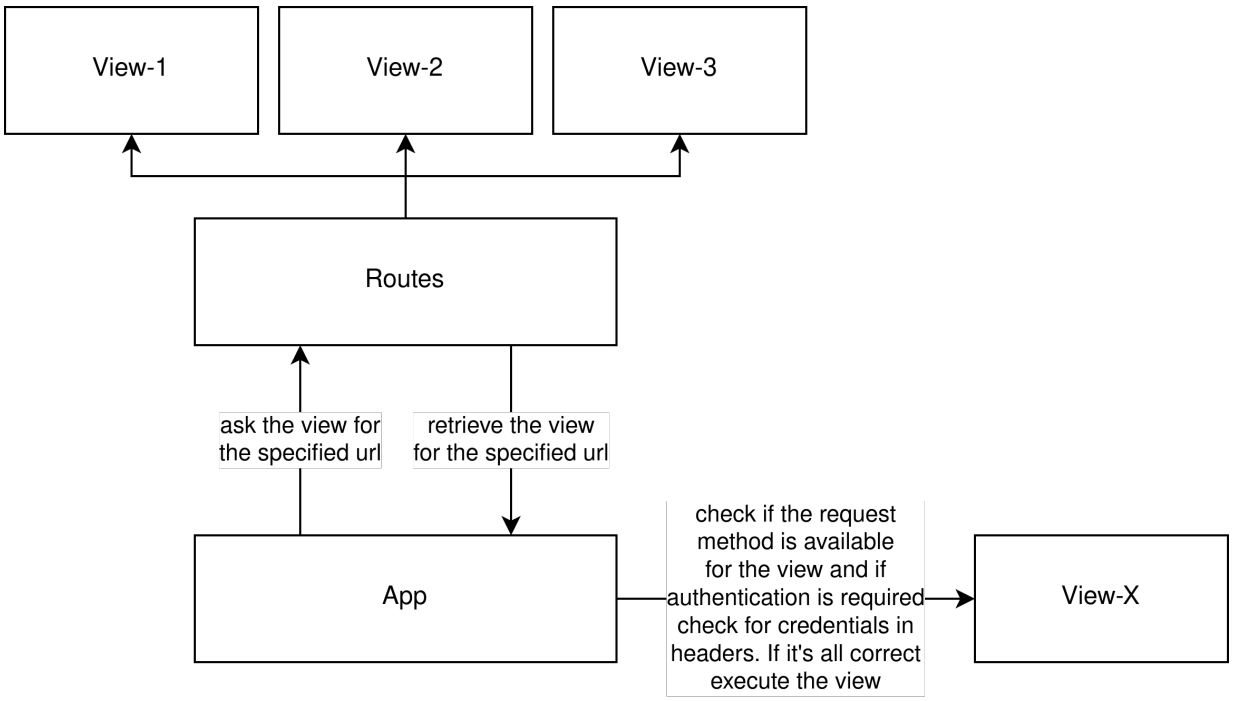

Figura 2.4: Gestione routes & views

2.4 Request

La gestione delle richieste inizia con la ricezione di una raw HTTP request da parte dell'Handler, tramite l'utilizzo di un RequestParser la richiesta verrà analizzata e verrà costruito un oggetto di tipo Request che conterrà le principali informazioni quali:

- method: il tipo di richiesta (GET, POST, HEAD, etc...)
- url: l'url richiesto
- parameters: i parametri della richiesta passati tramite url
- data: eventuali dati inviati tramite richiesta POST, necessariamente formattati in JSON
- connection: l'oggetto connessione con il client tramite il quale mandare una risposta

Avremo quindi ora un oggetto contenente il necessario per essere passato alla view che dovrà restituire il risultato al client in base alla richiesta.

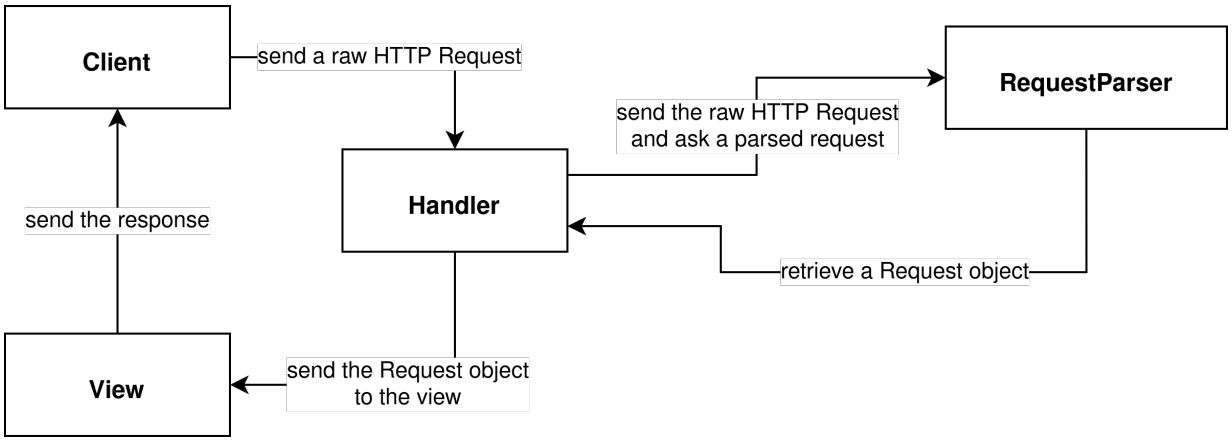

Figura 2.5: Schema gestione della fase di parsing delle request

2.5 Response

Una volta che la richiesta è stata analizzata ed è stato estratto il Request object questo viene passato alla view specifica di quell url, sarà quindi compito suo gestire la risposta da dare al client che ha fatto quella richiesta.
Per la costruzione di una risposta si utilizza un oggetto Response tramite il quale si potranno andranno a specificare i seguenti parametri:

- status code: il codice di risposta che sarà:
  - 200 (OK) in caso di successo.
  - 401 (Unauthorized) in caso di accesso senza credenziali / con credenziali errati a sezione dell'applicazione che richiedono livello di privilegi di admin.
  - 405 (Method Not Allowed) nel caso si stia facendo una richiesta ad un url tramite un metodo non supportato da quell url.
- content type: il tipo di contenuto che viene inviato tramite la response, text/html nel caso si stia restituendo una pagina HTML oppure application/pdf nel caso si stia inviando un documento PDF.
- headers: eventuali headers aggiuntivi da inviare
- file: il file da restituire che sarà letto e ne saranno estratti i bytes
- raw: nel caso non si voglia inviare un file si potranno inviare dei dati raw che dovranno essere necessariamente bytes.

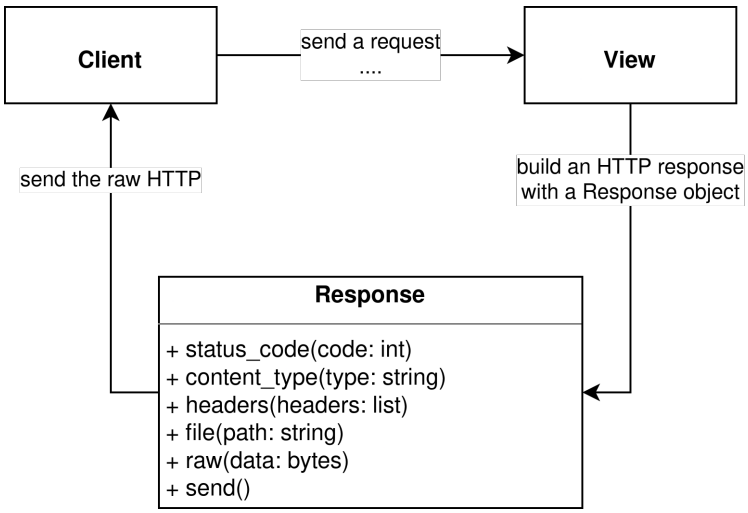

Figura 2.6: Schema gestione delle response da parte delle view

2.6 Autenticazione

Per quanto riguarda l'autenticazione si è deciso di utilizzare la Basic Access Authentication, avremo quindi una semplice autenticazione senza la necessità di usare cookies.
Questa autenticazione sarà richiesta solo in determinate pagine dell'applicazione e una volta inserità avrà valore fino alla fine della sessione, per sloggarci dovremo quindi o cancellare la cache oppure avviare un altra sessione del browser.
Le credenziali di accesso a queste pagine saranno quelle dell'amministratore e saranno specificate nel file 'credentials.json' in formato json in modo da rendere semplice la lettura.
Nel caso specifico dell'applicazione l'autenticazione servirà per accedere alla sezione di admin al quale solo la segreteria potrà accedere tramite l'ausilio delle credenziali fornitegli. In questo modo solamente loro saranno in grado di poter lavorare sulla pagina di pubblicazione degli esiti e dell'aggiunta dei contatti.

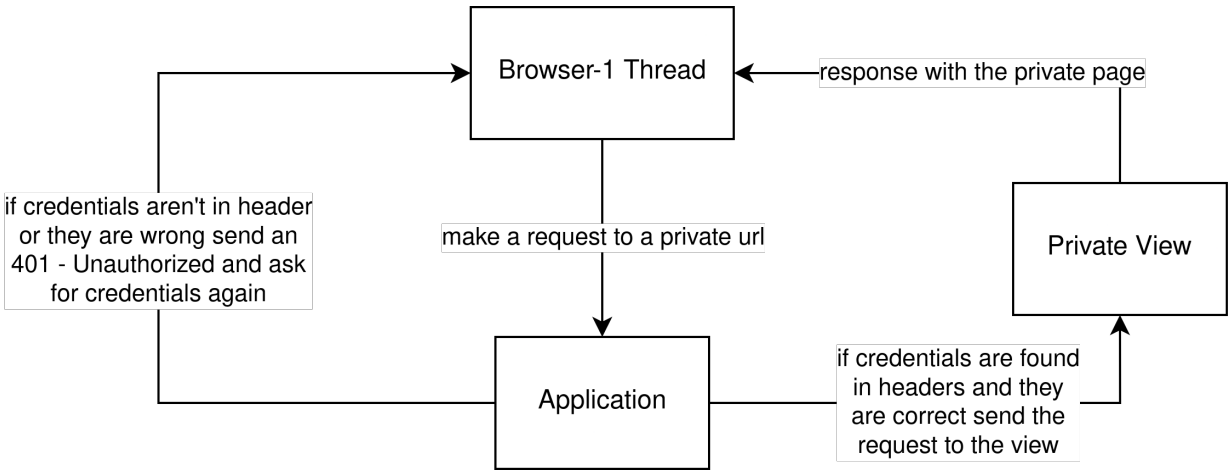

Figura 2.7: Schema gestione autenticazione per pagine private

2.7 API

Per la gestione della prenotazione delle visite e degli esiti di quest'ultime si è strutturato un sistema di API con degli endpoint che permetto la retrive e la pubblicazione di oggetti. Avremo quindi ad esempio l'endpoint '/api/visite' che se chiamato tramite una semplice richiesta GET restituirà tutte le visite presente nel database mentre se viene specificato un determinato ID ritornerà solamente la visita con quello specifico ID se presente, altrimenti un errore 404 verrà ritornato al client.
Se invece viene chiamato l'endpoint '/api/visite/post' tramite una richiesta POST significa che vogliamo pubblicare una nuova visita, dovremo quindi passare come body della richiesta i dati di questa visita che consistono per semplicità solamente nel nome e nel cognome della persona che vuole fare la visita.
La sicurezza è gestita richiedendo l'autenticazione agli endpoint che espongono dati sensibili quali la visualizzazione di una o tutte le visite e anche dell'accesso alla pubblicazione degli esiti e di nuovi contatti. In questo modo un utente che non si è autenticato con le credenziali della segreteria nel caso provasse a fare una POST request agli endpoint protetti riceverà un 401 - Unauthorized come risposta.
La gestione dei dati passati e forniti dalle API è completamente strutturata tramite l'utilizzo di JSON per avere una maggiore efficienza e flessibilità.
Per il testing delle API è stato utilizzato il software PostMan che ha permesso un veloce ed efficace testing nella creazione/lettura delle visite, degli esiti e dei contatti.

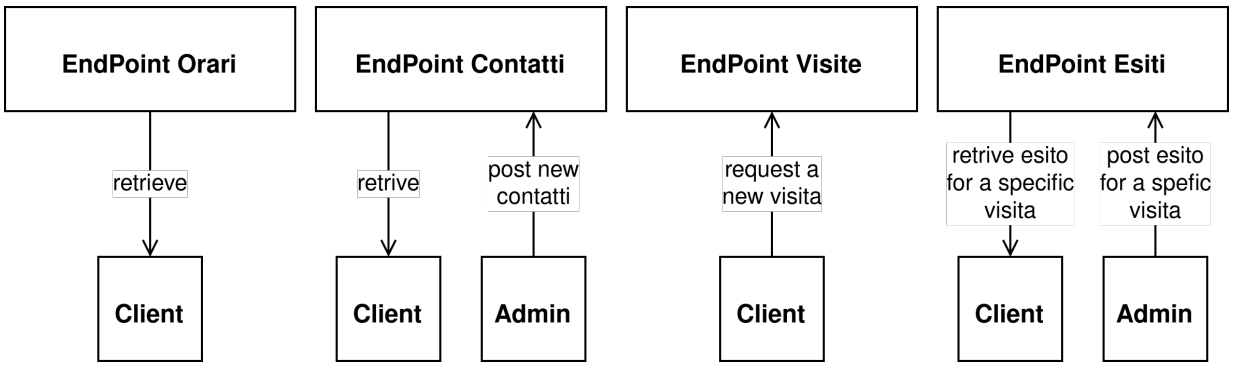

Figura 2.8: Endpoint delle API dell'applicazione

2.8 Database

JSON è stato inoltre utilizzato per quello che concerne il salvataggio dei dati, abbiamo infatti la cartella 'data' che costituisce il database dell'applicazione ed è gestito tramite l'ausilio di un file json per ognuna delle entità coinvolte, il quale conterrà tutte le istanze salvate di quest'ultima.
In questo modo per ottenere una determinata visita ci basterà caricare in memoria l'elenco delle visite salvate andando poi a filtrare in base all'id fornito, mentre per il salvataggio ci basterà andare ad aggiungere il nuovo record allo specifico file json.

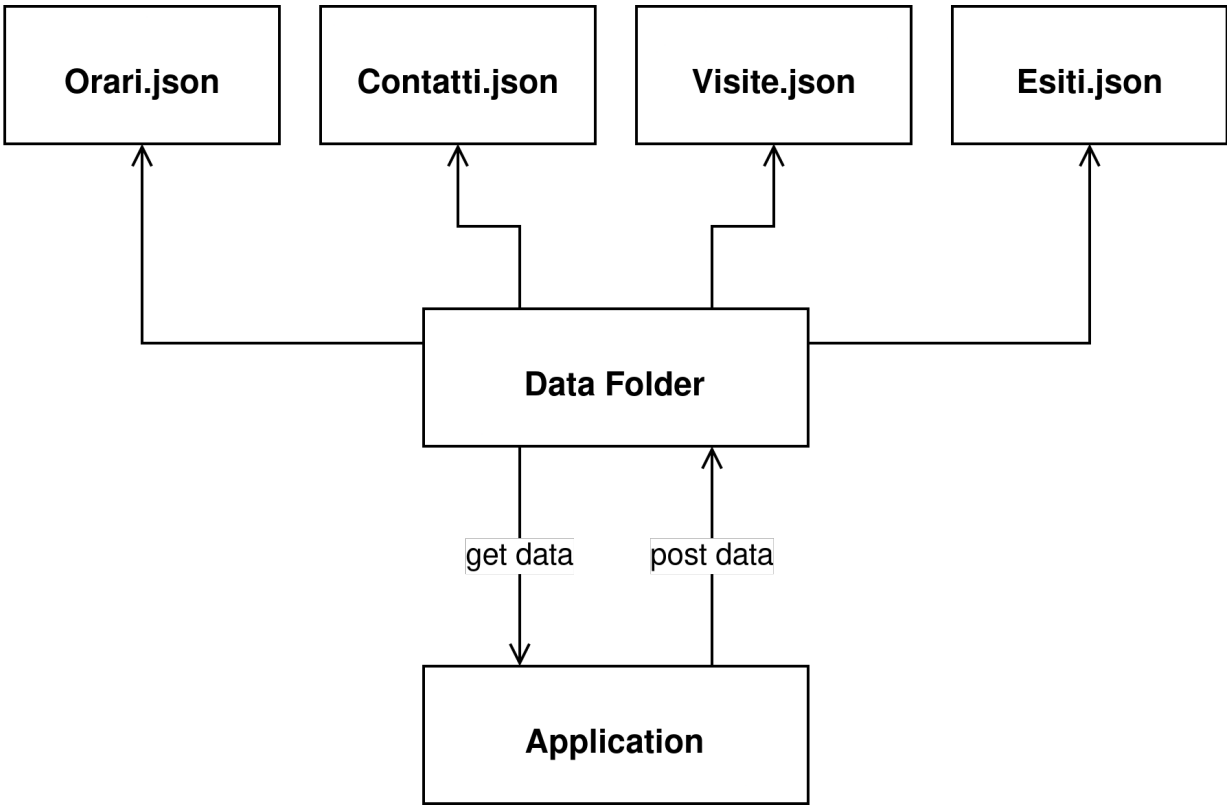

Figura 2.9: Gestione database dell'applicazione

2.9 Librerie

Le librerie utilizzate per costruire l'applicazione sono le seguenti:

- socketserver: per la gestione multi-threads del webserver
- http: per la costruizione di un Handler delle richieste
- sys: per ricevere parametri da riga di comando
- signal: per la gestione degli interrupt da tastiera al processo del webserver
- json: per la gestione delle credenziali salvate su file e per la lettura dei parametri passati tramite POST request che sono accettati solo se in formato JSON
- base64: per costruire la stringa di autenticazione con le credenziali di amministratore da utilizzare poi per confrontarla con l'autenticazione inviata dal client tramite specifico header.

Appendice A

Guida Utente

A.1 Avvio

Per l'avvio dell'applicazione sarà necessario digitare il comando:

```
$ python3 server.py <porta>
```

e il webserver si avvierà caricando tutte le Views presenti nei vari file della cartella 'views' ed avviando il server.
Nel caso non si specifichi una porta allora verra usata di default la porta 8888.

A.2 Aggiunta View

Nel caso si voglia aggiungere una o più view all'applicazione basterà modificare uno dei file .py presenti nella cartella views aggiungendo delle funzione imitando la sintassi di quelle già presenti, oppure creare un nuovo file python e crearle li dentro.
Esempio:

```
from modules.requests.methods import Methods
from modules.requests.request import Request
from modules.response.response import Response
from modules.app import app


@app.route(route='/normativa.pdf', methods=[Methods.GET])
def normativa(request: Request):
    response = Response(request.connection)
    response.status_code(200)
    response.content_type('application/pdf')
    response.file('files/normativa.pdf')
    response.send()
```