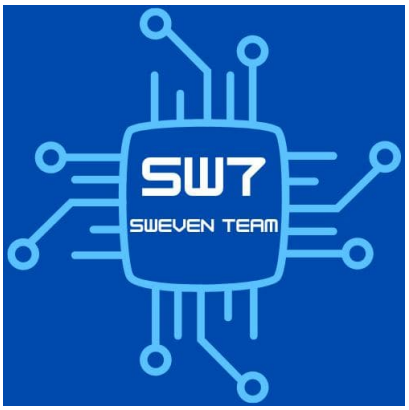
Specifica Architetturale



Sweven Team
swe7.team@gmail.com

Informazioni sul documento

Versione	1.0.1
Uso	Esterno
Destinatari	Gruppo Sweven Team
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Azienda Imola Informatica
Stato	In lavorazione
Redattori	Irene Benetazzo
	Samuele Rizzato
	Matteo Pillon
	Mattia Episcopo
Verificatori	Matteo Pillon
	Irene Benetazzo
	Tommaso Berlaffa
Approvatori	Mattia Episcopo

Sintesi

Specifica architetturale e delle tecnologie per la realizzazione del *Chatbot*$_G$.

Diario delle modifiche

Versione	Data	Descrizione	Ruolo	Autore	Verificatore
1.0.1	2022-09-24	Correzione simbolo §	Amministratore	Irene Benetazzo	Tommaso Berlaffa
1.0.0	2022-09-1	Approvazione documento	Responsabile	Mattia Episcopo	
0.1.0	2022-09-1	Verifica dell'intero documento	Verificatore	Matteo Pillon	
0.0.10	2022-08-31	Aggiornato §2.3 §2.4 §2.5	Progettista	Mattia Episcopo	Matteo Pillon
0.0.9	2022-08-30	Aggiornato §1.4.2 e §2.1	Progettista	Matteo Pillon	Irene Benetazzo
0.0.8	2022-08-30	Aggiornato §3	Progettista	Matteo Pillon	Irene Benetazzo
0.0.7	2022-08-30	Aggiornato §2.5	Progettista	Matteo Pillon	Irene Benetazzo
0.0.6	2022-08-30	Aggiornato §2.4 e §2.1	Progettista	Samuele Rizzato	Matteo Pillon
0.0.5	2022-08-29	Completamento §2 e §3	Progettista	Irene Benetazzo	Matteo Pillon
0.0.4	2022-08-27	Modifiche §2	Progettista	Irene Benetazzo	Matteo Pillon
0.0.3	2022-08-22	Scrittura §2.2 e §2.3	Progettista	Irene Benetazzo	Matteo Pillon
0.0.2	2022-08-09	Scrittura §3	Progettista	Irene Benetazzo	Matteo Pillon
0.0.1	2022-08-08	Scrittura §1	Amministratore	Irene Benetazzo	Matteo Pillon

	2022-07-21	Creazione documento	Amministratore	Irene Benetazzo	

Indice

1 Introduzione

1.1 Scopo del Documento

La Specifica Architetturale ha lo scopo di descrivere le scelte architetturali e tecnologiche attuate per la realizzazione del *Chatbot*$_G$.

1.2 Scopo del Capitolato

Lo scopo di tale progetto è quello di sviluppare un Chatbot che interfacciandosi con software aziendali spesso complessi e dispersivi, semplifichi i compiti che i dipendenti devono svolgere. In particolare vengono individuate le seguenti operazioni:

- Tracciamento della presenza in sede (**EMT**$_G$)
- Rendiconto attività svolte quotidianamente (**EMT**$_G$)
- Apertura del cancello aziendale (**MQTT**$_G$)
- Creazione di una riunione in un servizio esterno
- Servizio di ricerca documentale (**CMIS**$_G$)
- Creazione e tracciamento di bug (**Redmine**$_G$)

1.3 Glossario

Per assicurare la massima fruibilità e leggibilità del documento, il team SWEven ha deciso di creare un documento denominato *Glossario* il cui scopo sarà quello di contenere le definizioni dei termini ambigui o specifici del progetto. Sarà possibile riconoscere i termini presenti al suo interno in quanto terminanti con la lettera *G* posta come pedice della parola stessa.

1.4 Riferimenti

1.4.1 Normativi

- Norme di Progetto *v2.0.0*

1.4.2 Informativi

- Capitolato di appalto C1 - BOT4ME
- Slide del corso - Diagrammi dei casi d'uso
- Slide del corso - Diagrammi di sequenza
- Slide del corso - I pattern architetturali
- API - AssemblyAI

- REACT - JS
- CORS
- Flask
- Chatterbot
- Rest

2 Architettura

L'architettura del prodotto si basa su un modello di comunicazione tra Client e Server, all'interno della quale possiamo trovare i seguenti componenti:

- *Client*$_G$: interfaccia web realizzata utilizzando *React*$_G$ che permette all'utente di interfacciarsi con i servizi offerti dal chatbot.
- *Server*$_G$: si occupa di gestire le richieste in arrivo dai client, contattando le API-Rest offerte da Imola per svolgere le operazioni
- API *Rest*$_G$ Imola Informatica: insieme dei servizi resi disponibili dall'azienda, vengono contattate dal server ed effettuano la richiesta a cui sono associate.

2.1 Client

2.1.1 Introduzione

Il *Client*$_G$ è rappresentato dall'interfaccia web, con la quale l'utente usufruisce dei servizi offerti dall'applicativo. Esso è *Stateless*$_G$ cioè non contiene lo stato attuale della conversione, questa informazione viene gestita e salvata solamente lato *Server*$_G$.
Il *Client*$_G$ ad ogni avvio manda una richiesta *POST*$_G$ all'indirizzo del server */getID* richiendo di ricevere un identificativo univoco. Una volta ricevuta la risposta dal server contentente l'ID, esso viene salvato all'interno del *LocalStorage*$_G$ del browser, in questo modo ogni messaggio che viene mandato dal *Client*$_G$ al *Server*$_G$ avrà all'interno dei parametri della *POST*$_G$ il *clientID*$_G$ che permetterà al chatbot di ricollegare la conversazione allo specifico client.
L'aggiornamento della pagina del browser comporta l'assegnazione di un nuovo *clientID*$_G$ il che implica lo sviluppo di uno dei seguenti scenari:

- utente precedentemente loggato: la sua *API-KEY*$_G$ viene reperita dal *LocalStorage*$_G$ del browser, dove era stata memorizzata. Verrà quindi chiesto all'utente se effetturae il login nuovamente con tale *API-KEY*$_G$ o eventualmente inserirne una diversa.
- utente precedentemente non loggato: non avendo effettuato il login in precedenza, l'applicazione ripartirà chiedendo all'utente di inserire un *API-KEY*$_G$ per poter utilizzare i servizi offerti.

2.1.2 Componenti

Il *Client*$_G$ è stato sviluppando utilizzando *React*$_G$ che per struttura invita ad utilizzare dei componenti, i quali aiutano a suddividere e riutilizzare il codice. I componenti utilizzati dalla nostra applicazione sono:

- **AudioRecorder**: componente dedicato alla registrazione e invio del file audio al servizio esterno. Quest'ultimo si occupa della conversione del file audio in un testo, che viene restituito come stringa pronta da inviare al server.
- **CustomButton**: per questioni di manutenibilità e futura espansione si è deciso di realizzare un componente dedicato ai bottoni presenti all'interno dell'applicativo.
- **CustomIcon**: per questioni di manutenibilità e futura espansione si è deciso di realizzare un componente dedicato alle icone presenti all'interno dell'applicativo.
- **Home**: componente principale dell'applicativo lato *Client*$_G$, racchiude al suo interno gli altri componenti e rappresenta l'UI con la quale l'utente si interfaccia per utilizzare i servizi.
- **LoadingSpinner**: componente dedicato che avvisa l'utente sullo stato di avanzamento del processo di conversione del file audio.

2.1.3 Dipendenze esterne

Lato client possiamo trovare le seguenti dipendenze esterne:

- **AssemblyAI**: servizio esterno che si occupa della conversione dei file audio registrati dall'utente, ritornando un testo sotto forma di stringa.

2.2 Diagramma delle classi lato Server

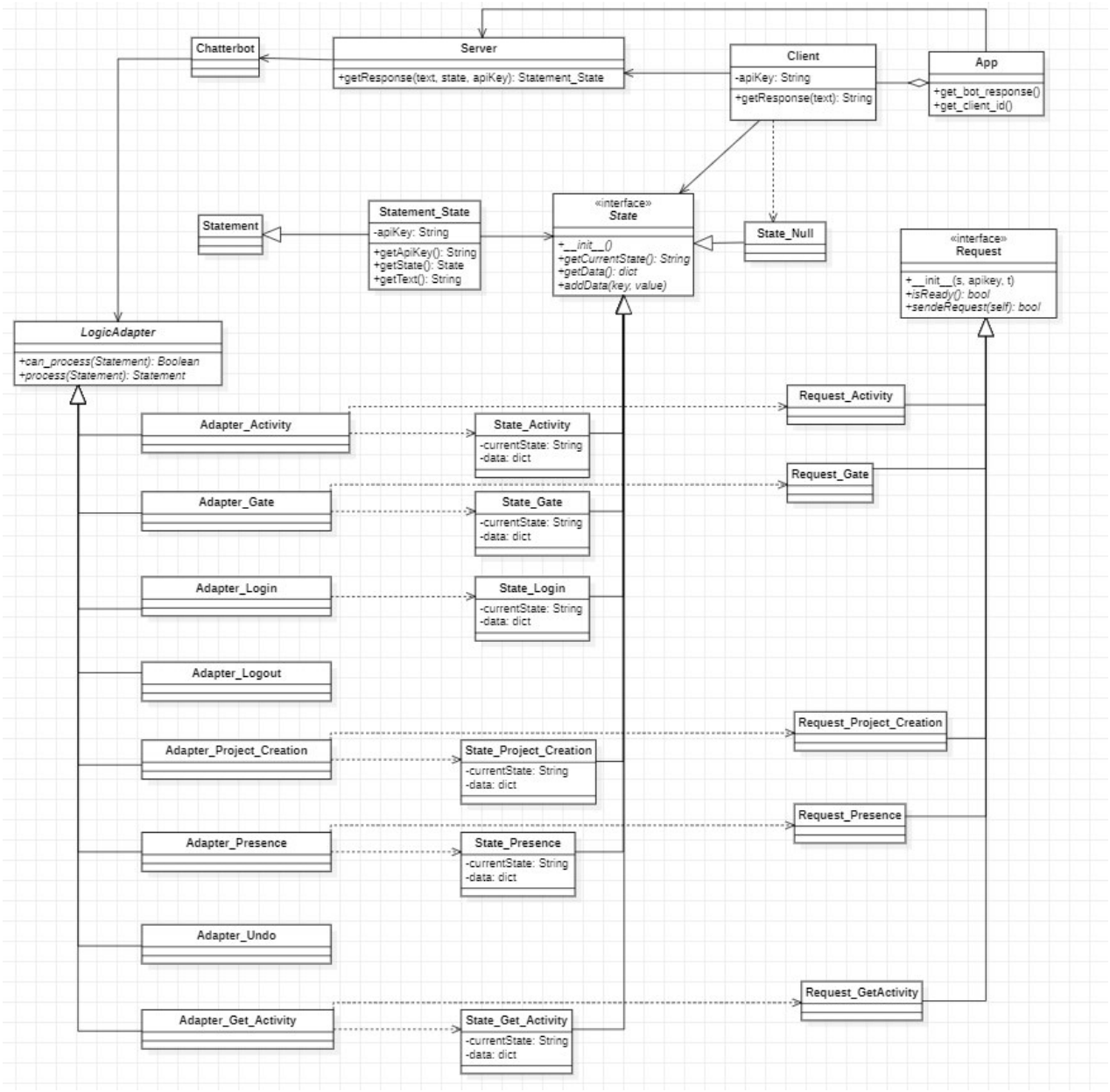


Figure 1: Diagramma UML delle classi lato Server

2.3 Server

2.3.1 App

Unico punto dell'applicativo dove viene interfacciato l'utente, qui viene istanziato il server e la lista dei client. Inoltre viene definito il routing delle richieste del client grazie al framework Flask.

2.3.2 Client

Classe che rappresenta ogni client connesso. Quando viene istanziata mantiene lo stato interno della richiesta, e con il metodo getResponse trasmette la richiesta al server.

2.3.3 Server

Classe che inizializza la libreria esterna *Chatterbot*. Presenta il metodo getResponse, il quale crea lo *Statement_State* con i parametri forniti dal Client e scorre tutti gli adapter cercando di rispondere in modo corretto alla richiesta.

2.3.4 Chatterbot

Classe della libreria esterna $Chatterbot_G$. Rappresenta un istanza del ChatBot e tutti gli adapter ad esso collegati.

2.3.5 Statement

Classe della libreria esterna $Chatterbot_G$. Rapprensenta un input che il ChatBot riceve o un output che il ChatBot ritorna in merito all'input ricevuto.

2.3.6 LogicAdapter

Classe della libreria esterna $Chatterbot_G$. Determina la logica di come il ChatterBot seleziona la risposta allo Statement di input. I suoi due metodi can_process e process sono il fulcro dell'applicativo lato server. can_process determina se una richiesta può essere soddisfatta dall'adapter, process definisce come elaborarla. Questa classe viene usata come classe base astratta per tutti gli adapter, questi infatti ereditano da LogicAdapter i due metodi e li sovvrascrivono per creare la propria logica di risposta.

2.3.7 State

Interfaccia che definisce il contratto di tutti i vari stati. Presenta i tre metodi che gli stati implementano per assolvere ognuno alla propria funzione. All'interno di ogni State saranno definiti i dati, in forma di dizionario (chiave, valore), necessari per completare la richiesta e su questi implementati i vari metodi. In generale getCurrentState ritornerà lo stato corrente, getData ritornerà i dati definiti all'interno di ogni State e addData aggiungerà un dato al dizionario.

State_Null Implementa l'interfaccia *State*, di particolare importanza perchè non è associato a nessun adapter. Serve per simulare lo stato nullo, viene utilizzato quando l'utente non è in nessun'altro stato.

2.3.8 Statement_State

Sottoclasse di Statement (classe della libreria esterna *Chatterbot*$_G$). Estende lo Statement di Chatterbot con lo State e l'ApiKey della richiesta.

2.3.9 Request

Interfaccia che definisce tutte le request. Presenta due metodi isReady e sendRequest il quale funzionamento verrà implementato per ogni request. In generale isReady verifica se la request è pronta per essere inviata, mentre invece sendRequest invia la richiesta *HTTP*$_G$ alle *API Rest*$_G$ di Imola Informatica per interagire con i loro servizi, e ritorna una risposta per verificare se la richiesta è andata a buon fine o meno.

2.3.10 Login

Classi *Adapter_Login*, *State_Login* permettono di effettuare il login

2.3.11 Logout

Classe *Adapter_Logout* permette di effettuare il logout.

2.3.12 Activity

Classi *Adapter_Activity*, *State_Activity* e *Request_Activity*. Sono le classi necessarie per comporre ed inviare una richiesta di consuntivazione per un attività svolta su un progetto esistente. La richiesta viene composta con i campi codice progetto, data , ore fatturabili, ore viaggio, ore viaggio fatturabili, sede, fatturabile (campo che determina se è fatturabile o meno) e descrizione.

2.3.13 Gate

Classi *Adapter_Gate*, *State_Gate* e *Request_Gate*. Sono le classi necesssarie a comporre ed inviare una richiesta per l'apertura del cancello. La richiesta viene composta con il campo sede che determina la sede della quale aprire il cancello.

2.3.14 Project_Creation

Classi *Adapter_Project_Creation*, *State_Project_Creation* e *Request_Project_Creation*. Sono le classi necessarie per comporre ed inviare una richiesta per la creazione di un nuovo progetto. La richiesta è composta dai campi codice progetto, dettagli, cliente, manager, status (che viene di default inizializzato come iniziale), area, data inizio, data fine.

2.3.15 Presence

Classi *Adapter_Presence*, *State_Presence* e *Request_Presence*. Sono le classi necessarie per comporre ed inviare la richiesta di registrazione in presenza presso una sede. La richiesta è composta dall'unico campo sede che determina appunto la sede nella quale si vuole registrare la presenza.

2.3.16 Get Activity

Classi *Adapter_Get_Activity*, *State_Get_Activity* e *Request_Get_Activity*. Sono le classi necessarie a comporre la richiesta per visualizzare le consuntivazioni già effettuate per un progetto esistente. L'unico campo che compone la richiesta è il codice progetto per il quale si vuole visualizzare tutto quello che fin'ora è stato consuntivato.

2.3.17 Undo

Classe *Adapter_Undo*. Permette di annullare qualsiasi operazione in corso e inserire la richiesta per la stessa o un'altra funzionalità.

2.4 Diagramma di sequenza Esecuzione Richiesta

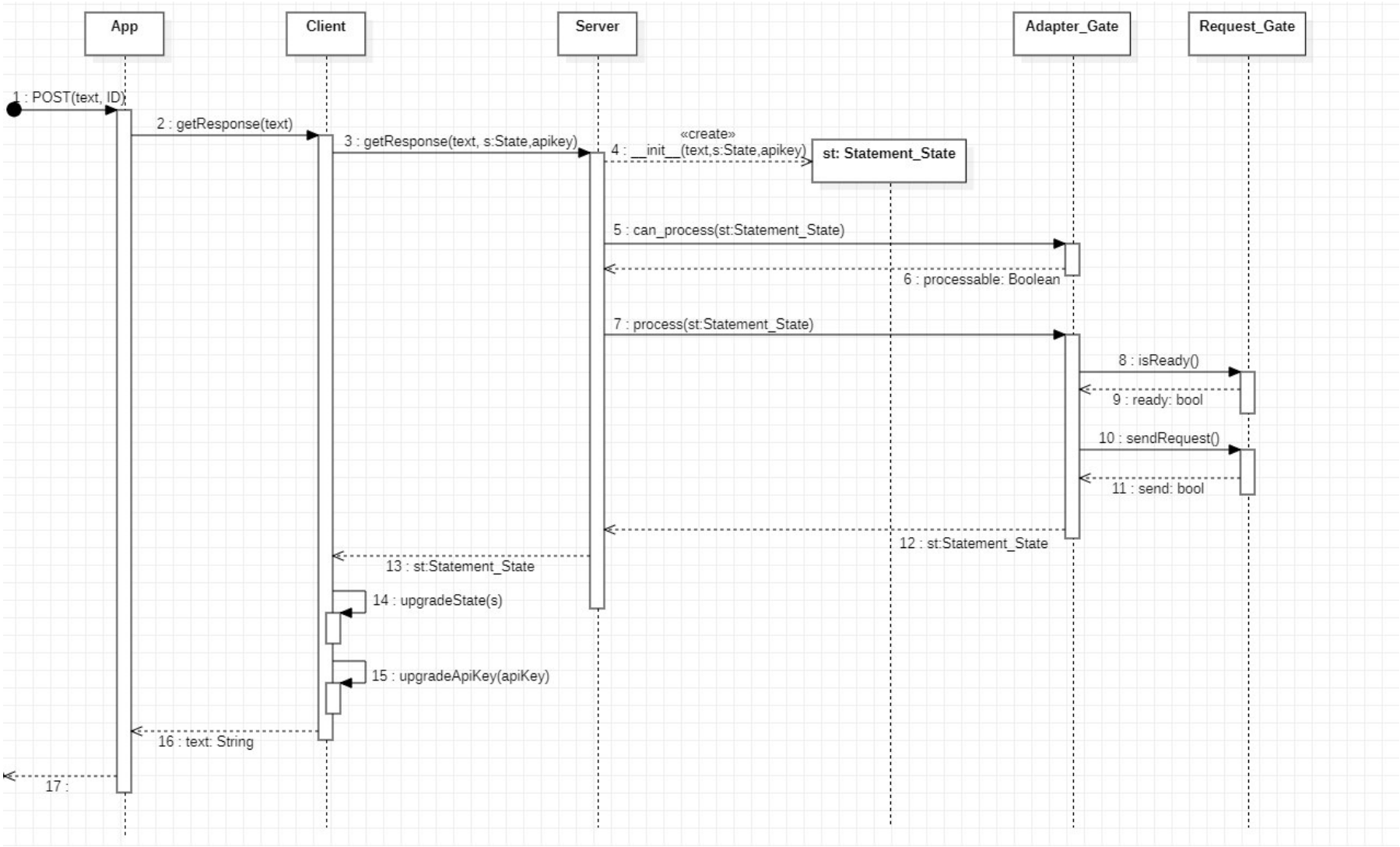


Figure 2: Diagramma di sequenza Esecuzione Richiesta

- Il *Client*$_G$ web effettua una richiesta POST all'*App*, che si compone di testo inserito dall'utente (text) e identificativo univoco del *Client* connesso (ID).
- *App* riceve la richiesta, seleziona il corretto *Client* che deve gestirla mediante l'ID ricevuto e inoltra a *Client* il testo della richiesta invocando il metodo *getResponse*.
- Il *Client* richiama il metodo del *Server getResponse* passandogli il testo della richiesta, lo stato attuale della richiesta e l'*ApiKey* con la quale la richiesta è stata effettuata.
- Il *Server* riceve la richiesta del *Client*, crea lo *Statement_State* con i tre dati che ha ricevuto, e invoca la libreria Chatterbot per cercare la risposta più pertinente al testo della richiesta (in questo caso "apertura cancello"). In questo caso il Server fa la richiesta ad *Adapter_Gate* con il metodo *can_process*, passando come paramentro lo *Statement_State* che ha creato.
- *Adapter_Gate* risponde con un boolean alla richiesta per determinare se può processarla o meno, e lo ritorna al *Server*.
- Il server richiama *Adapter_Gate* per processare la richiesta passando come parametro lo *Statement_State* che ha creato.
- *Adapter_Gate* prepara la richiesta, in base ai dati che ha ricevuto e verifica con il metodo *isReady* di *Request_Gate* se questa è pronta per essere effettuata. *Request_Gate* risponde con un boolean indicando se la richiesta è pronta o meno per essere eseguita.
- Se la richiesta è pronta per essere effettuata (come nell'esempio) *Adapter_Gate* invoca il metodo *sendRequest* di *Request_Gate* per effettuare realmente la richiesta verso le API di Imola Informatica. *Request_Gate* riponde con un boolean che indica se la richiesta è andata a buon fine.
- A questo punto *Adapter_Gate* ritorna lo *Statement_State* al server che aveva iniziato la richiesta con il metodo *process*.
- Il *Server* dopo aver ricevuto lo *Statement_State* in risposta dell'adapter lo trasmette al *Client* che aveva fatto la richiesta *getResponse*.
- Il *Client* riceve lo *Statement_State*, aggiorna il proprio stato con il metodo *upgradeState* e la propria *ApiKey* con il metodo *upgradeApiKey*. Infine ritorna ad *App* la risposta sotto forma di string (text) perchè venga indirizzata al *Client*$_G$ web.

2.5 Diagramma di sequenza Richiesta Identificativo

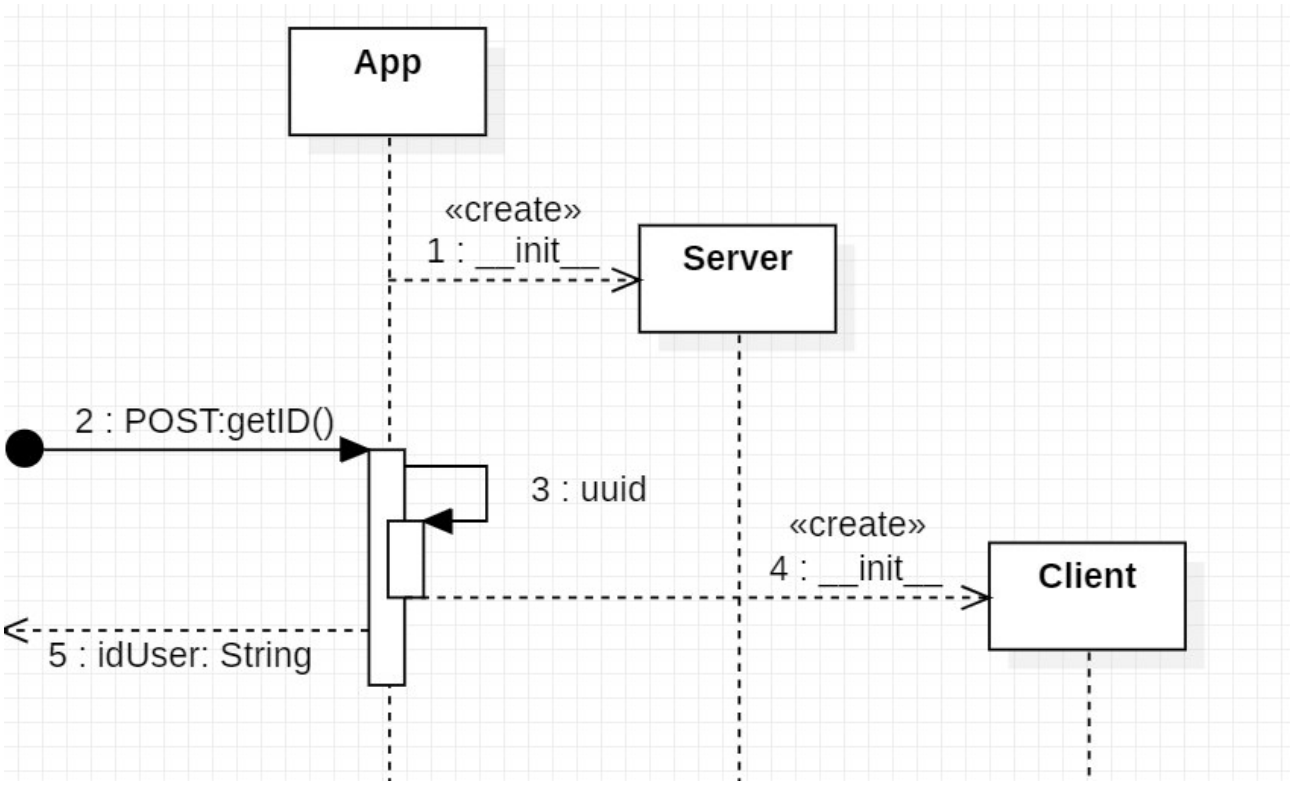


Figure 3: Diagramma di sequenza Richiesta Identificativo

- Il *Client*$_G$ web effettua una richiesta POST *getID* ad *App* richiedendo un ID.
- *App* riceve la richiesta del nuovo Client che vuole connettersi al *Server*, precedentemente creato e inizializzato, e crea un *UUID*$_G$, per identificarlo univocamente in tutte le comunicazioni.
- *App* in seguito crea ed inizializza un *Client* nuovo associato al ID appena creato.
- Infine *App* ritorna al *Client*$_G$ web l'identificativo univoco creato sotto forma di un campo stringa (idUser).

2.6 Design Pattern

Front Controller pattern è un pattern architetturale per la progettazione di applicazioni web che forniscono un punto di ingresso centralizzato per la gestione delle richieste. Nel caso del nostro applicativo infatti la classe *App*, in cui è presente $Flask_G$ si occupa di ricevere tutte le richieste ricevute dai vari $Client_G$, le quali verranno poi gestite da un'istanza della classe *Client* lato $Server_G$ che andrà a contattare l'adapter corretto in base all'interpretazione del messaggio, effettuando in seguito una chiamata all'API-REST consona al servizio richiesto.

2.7 API Rest Imola Informatica

L'azienda ha fornito delle *API Rest*$_G$ che permettono al *chatbot*$_G$ di interagire con i loro sistemi aziendali. Sono facilmente consultabili a questo link.

2.7.1 Consuntivazione di un attività

2.7.1.1 API Url

/projects/{code}/activities/me

2.7.1.2 Metodo di richiesta *HTTP*$_G$

POST

2.7.1.3 Headers *HTTP*$_G$

- **accept**: application/json;
- **api_key**: api_key, per autorizzare le richieste;
- **Content-Type**: application/json.

2.7.1.4 Parametri URL

Nome	Tipo	Descrizione
code	string	rappresenta il codice del progetto di cui fa parte l'attività da rendicontare.

2.7.1.5 Parametri Body

Nel body della richiesta viene passato un array contenente un json con questi parametri

Nome	Tipo	Descrizione
date	string	data in cui si è svolta l'attività.
billableHours	int	ore fatturabili dell'attività.
travelHours	int	ore di viaggio spese per l'attività.
billableTravelHours	int	ore di viaggio fatturabili per l'attività.
location	string	nome della sede in cui si è svolta l'attività.
billable	bool	indica se l'attività è fatturabile.
note	string	descrizione dell'attività.

2.7.1.6 Risposte

Status code *HTTP*$_G$	Descrizione
204	indica che la richiesta di consuntivazione è andata a buon fine
404	indica che il codice specificato del progetto non è corretto
401	indica che l'api key non è stata specificata o quella utilizzata non è valida.

2.7.2 Apertura del cancello

2.7.2.1 API Url
/locations/{location_name}/devices/{device}/status

2.7.2.2 Metodo di richiesta *HTTP*$_G$
PUT

2.7.2.3 Headers *HTTP*$_G$

- **accept**: application/json;
- **api_key**: api_key, per autorizzare le richieste;
- **Content-Type**: application/json.

2.7.2.4 Parametri URL

Nome	Tipo	Descrizione
location_name	string	sede di cui aprire il cancello.
device	string	nome del dispositivo da utilizzare, in questo caso il cancello.

2.7.2.5 Parametri Body
Nel body della richiesta viene passato un json contenente questi parametri

Nome	Tipo	Descrizione
status	string	rappresenta il nuovo stato del dispositivo.

2.7.2.6 Risposte

Status code *HTTP*$_G$	Descrizione
204	indica che la richiesta di apertura del cancello è andata a buon fine.
404	indica che la sede di cui aprire il cancello non è valida.
401	indica che l'api key non è stata specificata o quella utilizzata non è valida.

2.7.3 Registrazione della presenza

2.7.3.1 API Url
/locations/{location_name}/presence

2.7.3.2 Metodo di richiesta *HTTP*$_G$
POST

2.7.3.3 Headers *HTTP*$_G$

- **accept**: application/json;
- **api_key**: api_key, per autorizzare le richieste;
- **Content-Type**: application/json.

2.7.3.4 Parametri URL

Nome	Tipo	Descrizione
location_name	string	sede in cui registrare la presenza.

2.7.3.5 Risposte

Status code *HTTP*$_G$	Descrizione
200	indica che la registrazione della presenza è stata effettuata con successo.
404	indica che la sede specificata non è valida.
401	indica che l'api key non è stata specificata o quella utilizzata non è valida.

2.7.4 Creazione di un nuovo progetto

2.7.4.1 API Url
/projects

2.7.4.2 Metodo di richiesta *HTTP*$_G$
POST

2.7.4.3 Headers *HTTP*$_G$

- **accept**: application/json;
- **api_key**: api_key, per autorizzare le richieste;

2.7.4.4 Parametri Body
Nel body della richiesta viene passato un json contenente questi parametri

Nome	**Tipo**	**Descrizione**
code	string	codice del progetto da creare.
detail	string	descrizione del progetto da creare.
customer	string	cliente per cui viene creato il progetto.
manager	string	manager del progetto da creare.
status	string	stato del progetto.
area	string	sede in cui svolgere il progetto.
startDate	string	data di inizio del progetto.
endDate	string	data di fine del progetto.

2.7.4.5 Risposte

Status code *HTTP*$_G$	**Descrizione**
204	indica che è l'operazione di creazione del progetto è andata a buon fine.
400	indica che almeno uno dei parametri non è stato specificato.
401	indica che l'api key non è stata specificata o quella utilizzata non è valida.

2.7.5 Recupero delle sedi

2.7.5.1 API Url
/locations

2.7.5.2 Metodo di richiesta *HTTP*$_G$
GET

2.7.5.3 Headers *HTTP*$_G$

- **accept**: application/json;
- **api_key**: api_key, per autorizzare le richieste;

2.7.5.4 Risposte

Status code *HTTP*$_G$	**Descrizione**
200	ritorna una lista contenente le informazioni delle sedi.
401	indica che l'api key non è stata specificata o quella utilizzata non è valida.

2.7.6 Recupero della rendicontazione per il progetto corrente

2.7.6.1 API Url
/projects/{code}

2.7.6.2 Metodo di richiesta *HTTP*$_G$
GET

2.7.6.3 Headers *HTTP*$_G$

- **accept**: application/json;
- **api_key**: api_key, per autorizzare le richieste;

2.7.6.4 Parametri URL

Nome	**Tipo**	**Descrizione**
code	string	codice del progetto corrente.

2.7.6.5 Risposte

Status code *HTTP*$_G$	Descrizione
200	ritorna un json contenente le informazioni per il progetto corrente.
401	indica che l'api key non è stata specificata o quella utilizzata non è valida.

3 Tecnologie

3.1 API Rest

Un'API REST è un'interfaccia di programmazione delle applicazioni conforme ai vincoli dello stile architetturale REST, che consente l'interazione con servizi web RESTful.
Il termine REST è l'acronimo di REpresentational State Transfer. REST è un insieme di vincoli architetturali, non un protocollo né uno standard. Quando una richiesta client viene inviata tramite un'API RESTful, questa trasferisce al richiedente o all'endpoint uno stato rappresentativo della risorsa. L'informazione viene consegnata in HTTP in un formato JSON, HTML, Python o txt.

3.2 Server

3.2.1 Python

Linguaggio di programmazione ad alto livello, adatto alla programmazione orientata agli oggetti. E' stato utilizzato per sviluppare il back-end insieme alla libreria esterna Chatterbot.

3.2.1.1 Chatterbot Libreria esterna in *Python*$_G$ che utilizza algoritmi di intelligenza artificiale per trovare la migliore risposta per emulare il comportamento di un *chatbot*$_G$ nel server. Grazie alla sua flessibilità si sono implementati degli adapter che modellano e gestiscono le varie richieste dell'utente.
Durante l'esecuzione Chatterbot crea un adapter che gli consente di connettersi ad un database *SQLite*$_G$.

3.2.2 Flask

Framework Python per lo sviluppo di applicazioni web. Flask contiene tutte le classi e le funzioni necessarie per la costruzione di una web app, e ha agevolato l'organizzazione e la gestione del *chatbot*$_G$.

3.2.3 Cors

Cross-Origin Resource Sharing regola la cooperazione tra sito web e il caricamento dati da un server esterno garantendone la sicurezza attraverso un'intestazione *HTTP*$_G$.

3.2.4 UUID

Universally Unique IDentifier, cioè identificativo univoco universale, è usato nelle infrastrutture software per avere l'unicità pratica senza garanzia in un sistema distribuito semplificando il mantenimento dell'identità degli oggetti in ambienti disconnessi.

3.3 Client

3.3.1 React

React è una libreria JavaScript per costruire l'interfaccia utente caratterizzata dal fatto che è dichiarativa, efficiente e flessibile. E' stato utilizzato per creare l'applicazione lato client.

3.3.2 HTML

Linguaggio di markup, in standard W3C, per documenti visualizzabili attraverso un web browser.

3.3.3 CSS

Linguaggio di formattazione per i documenti HTML.

3.3.4 Regex

Libreria esterna che controlla le espressioni regolari per la validazione dell'*api-key*$_G$. Viene infatti definito uno schema secondo il quale un'*api-key*$_G$ per poter essere valida deve validare uno schema di cifre.

3.3.5 API AssemblyAI

Servizio esterno che permette di tradurre automaticamente file audio in testo richiamando l'API Speech-to-text.