SPECIFICA ARCHITETTURALE

BOT4ME - IMOLA INFORMATICA

seven.solutions.unipd@gmail.com

INFORMAZIONI DOCUMENTO

Versione	1.0.0
Uso	esterno
Stato	approvato
Destinatari	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Imola Informatica
	Seven Solutions
Redattori	Cavaliere Alessandro
	Filippi Gabriele
	Ruffin Filippo
Verificatori	Bonato Manuele
	Galtarossa Marco
Approvazione	Marangon Marco

Descrizione
Scelte architetturali e tecnologie per la realizzazione del progetto.

Registro delle modifiche

Versione	Data	Autore (verificatore)	Ruolo	Descrizione
1.0.0	24-04-2022	Marangon Marco (-)	Responsabile	Approvazione
0.1.0	23-04-2022	Galtarossa Marco (-)	Verificatore	Verifica generale
0.0.13	22-04-2022	Ruffin Filippo (Bonato Manuele)	Programmatore	Definizione §2.4 e §2.5
0.0.12	20-04-2022	Bonato Manuele, Ruffin Filippo (Filippo Gabriele)	Programmatore	Integrazione §2.3.2 e §3.3
0.0.11	14-04-2022	Filippi Gabriele, Ruffin Filippo (Galtarossa Marco)	Progettista	Integrazione §2.3.3 e §2.3.4
0.0.10	06-04-2022	Galtarossa Marco, Gambirasio Leonardo (Filippi Gabriele)	Programmatore	Integrazione §2.3.2 e §3.3
0.0.9	05-04-2022	Bonato Manuele, Marangon Marco (Gambirasio Leonardo)	Programmatore	Definizione §2.2.2 e §3.2
0.0.8	29-03-2022	Bonato Manuele, Cavaliere Alessandro (Gambirasio Leonardo)	Progettista	Integrazione §2.3.3 e §2.3.4
0.0.7	21-03-2022	Galtarossa Marco, Gambirasio Leonardo (Filippi Gabriele)	Progettista	Definizione §2.2.3 e §2.2.4
0.0.6	16-03-2022	Marangon Marco, Ruffin Filippo (Gambirasio Leonardo)	Programmatore	Definizione §2.3.2 e §3.3
0.0.5	10-03-2022	Filippi Gabriele, Marangon Marco (Bonato Manuele)	Progettista	Definizione §2.3.3 e §2.3.4
0.0.4	04-03-2022	Bonato Manuele, Gambirasio Leonardo (Filippi Gabriele)	Programmatore	Definizione §2.1.2 e §3.1
0.0.3	24-02-2022	Cavaliere Alessandro, Galtarossa Marco (Ruffin Filippo)	Progettista	Definizione §2.1.3 e §2.1.4
0.0.2	22-02-2022	Bonato Manuele (Filippi Gabriele)	Progettista	Stesura §1
0.0.1	21-02-2022	Gambirasio Leonardo (-)	Responsabile	Creazione documento

Indice

1 Introduzione

1.1 Scopo del documento

Il seguente documento illustra le scelte architetturali fatte durante le varie fasi di progettazione del prodotto.
In particolare è possibile trovare:

- **Analisi architetturale**: insieme di diagrammi che illustrano la progettazione architetturale e di dettaglio del prodotto. Si articola in:
  - Diagrammi delle classi.
  - Diagrammi di sequenza.
- **Linguaggi**: linguaggi di programmazione ed eventuali FRAMEWORK$_G$ utilizzati.

1.2 Scopo del capitolato

Lo scopo del progetto è lo sviluppo di un CHATBOT$_G$ che sia in grado di aiutare i dipendenti delle aziende nelle mansioni che richiedono di interfacciarsi con applicativi spesso poco intuitivi.
In particolare il chatbot deve permettere agli utilizzatori di svolgerle tutte all'interno dello stesso applicativo.
Le principali operazioni sono:

- **TRACCIAMENTO DELLA PRESENZA$_G$ in sede in SISTEMA EMT$_G$.**
- **Inserimento nuova ATTIVITÀ$_G$ in SISTEMA EMT$_G$.**
- **Apertura del cancello.**
- **Inserimento di una nuova riunione su un APPLICATIVO ESTERNO$_G$.**
- **Servizio di ricerca documentale.**
- **Servizio di creazione TICKET$_G$.**

1.3 Organizzazione sezioni

Il documento si articola in 2 sezioni principali:

- **Architettura di sistema**: illustrazione delle scelte architetturali tramite diagrammi UML.
- **Tecnologie per sviluppo**: insieme di tecnologie e FRAMEWORK$_G$ utilizzati per la codifica dell'architettura.

1.4 Glossario e documentazione esterna

Per evitare possibili incomprensioni relative alle terminologie utilizzate nel documento, verranno utilizzate due simboli:

- *D* al pedice per indicare il nome di un particolare documento.
- *G* al pedice per indicare un termine che sarà presente nel GLOSSARIO v1.0.0$_D$.

1.5 Riferimenti

1.5.1 Normativi

- NORME DI PROGETTO v3.0.0$_D$.
- PIANO DI PROGETTO v3.0.0$_D$.

1.5.2 Informativi

- Slide del corso - Diagrammi delle classi
- Slide del corso - Diagrammi di sequenza
- Slide del corso - Design Pattern Architetturali
- Slide del corso - Model-View Patterns
- Documentazione libreria chatterbot
- Django - MVT design pattern
- Documentazione ANDROID$_G$

2 Architettura di sistema

L'architettura del prodotto è suddivisa nelle seguenti componenti:

- **App Client**: APP$_G$ ANDROID$_G$.
- **Web Client**: sito web con funzionalità equivalenti all'APP$_G$ ANDROID$_G$.
- **Server**: componente di business che gestisce le richieste degli utenti e le realizza.
- **API REST$_G$ CHATBOT$_G$**: API ENDPOINT$_G$ per le richieste dei client al server.
- **API REST$_G$ Imola Informatica**: vari API ENDPOINT$_G$ per effettuare le operazioni richieste dagli utenti aziendali.

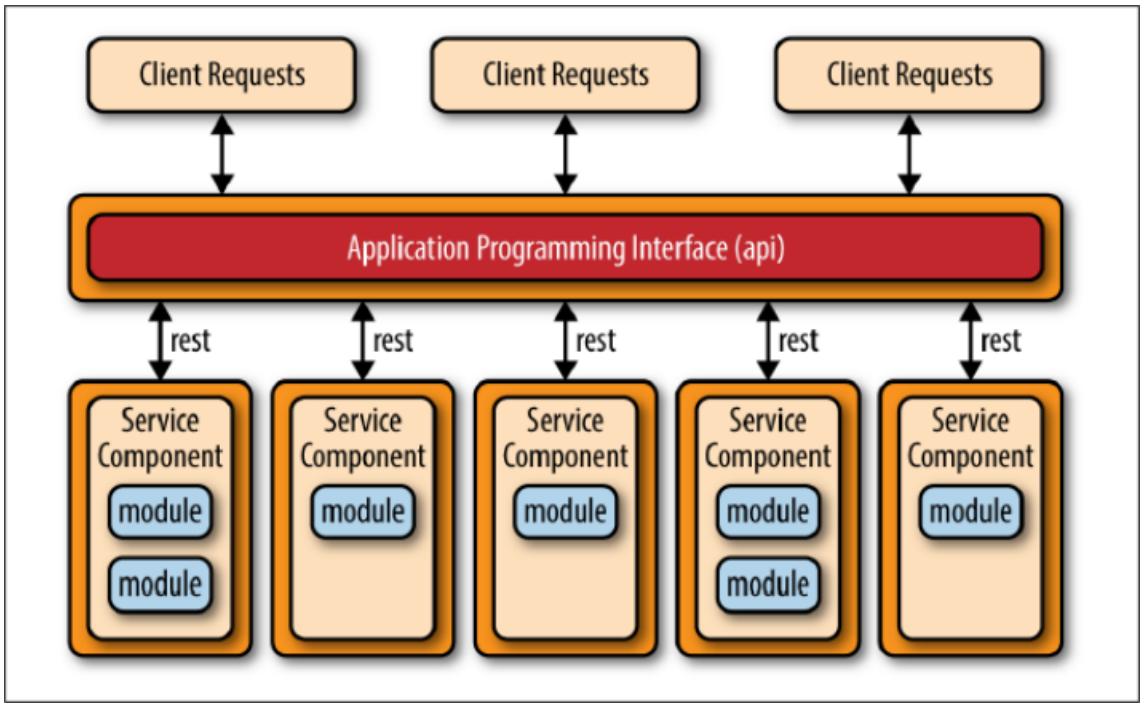


Figura 1: Schema sintetico del sistema, preso dalle slide del corso.

Le prime componenti saranno illustrate attraverso le seguenti sotto-sezioni:

- **Introduzione**: scopo della componente e suo utilizzo.
- **Dipendenze esterne**: dipendenza delle classi da componenti esterne all'architettura del prodotto.
- **Diagramma delle classi**: contenente le seguenti parti:
  - Definizione delle classi principali e loro interdipendenze interne tramite diagramma UML$_G$.
  - Breve descrizione della classe e del suo scopo.
- **Diagramma di sequenza**: mostrare come le componenti collaborano allo scopo di fornire la loro funzionalità.
- **Desgin pattern**: eventuali *desgin pattern* utilizzati nella componente.

Per i servizi REST$_G$ esposti dal CHATBOT$_G$ si indicheranno:

- **API ENDPOINT$_G$**: URL$_G$ dove risiede il servizio.
- **Chiamata HTTP$_G$**: tipo di verbo HTTP$_G$.
- **Parametri della richiesta**: *body* della richiesta HTTP$_G$.
- **HTTP$_G$ Headers**: *headers* della richiesta HTTP$_G$.
- **Risposte**: possibili risposte della chiamata HTTP$_G$, composte da:
  - HTTP$_G$ Status Code.
  - *Body* in formato JSON$_G$.
  - Descrizione.
- **Descrizione**: funzionalità che mette a disposizione.

2.1 App Client

2.1.1 Introduzione

L'APP$_G$ ANDROID$_G$ mira unicamente a fornire un'interfaccia utente equivalente a quella del sito web, mantenendo un più classico *chat layout* rispetto ad esso. L'applicazione permette di:

- Iniziare una conversazione con il CHATBOT$_G$ ogni volta che si apre l'applicazione.
- Mandare messaggi al CHATBOT$_G$ e ricevere risposte da esso.
- Impostare, modificare, eliminare l'*api_key* dalla sezione impostazioni.

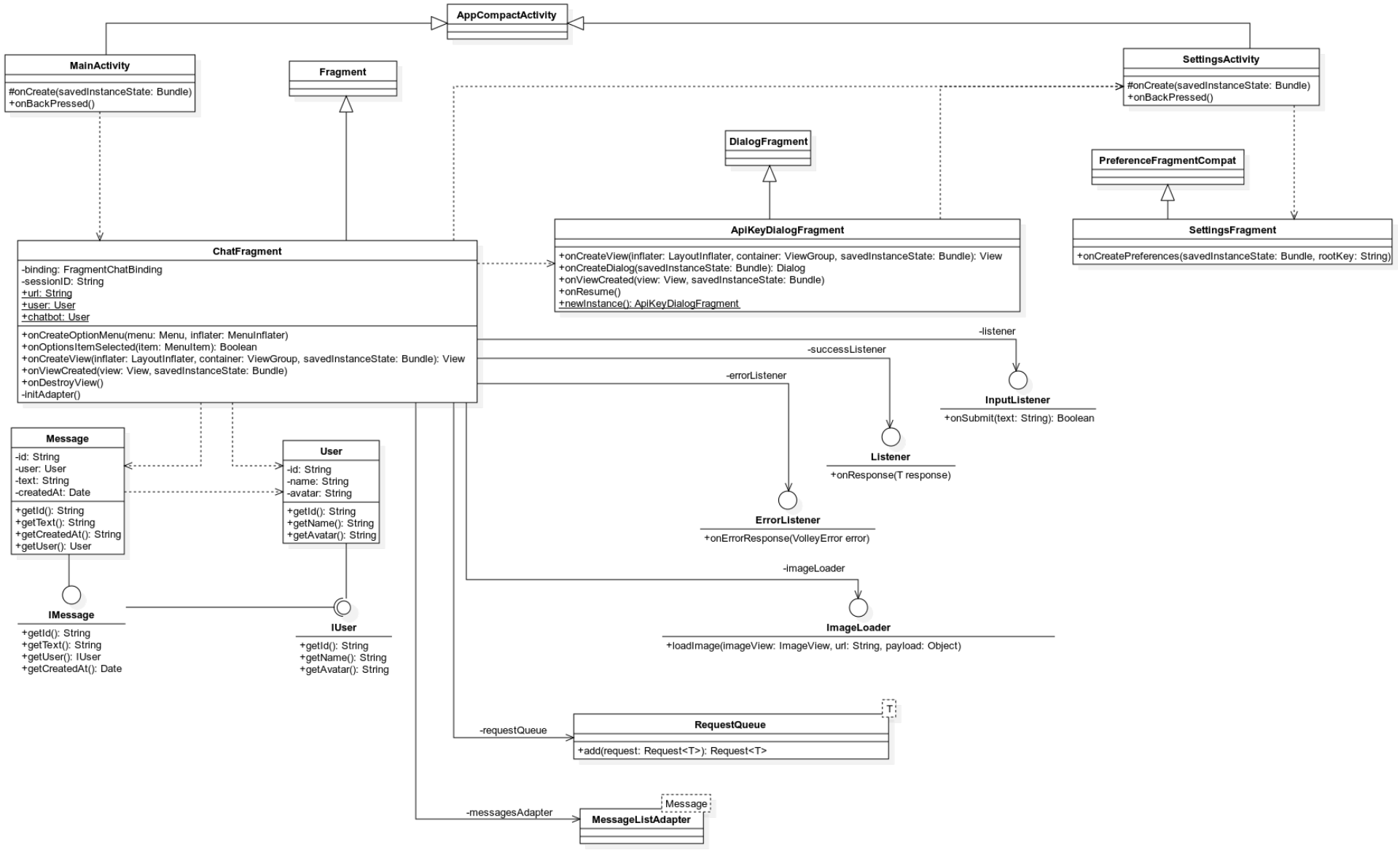
Tutte queste operazioni vengono svolte dall'utente senza la necessità che esso sia a conoscenza di come il SISTEMA EMT$_G$ aziendale funzioni.

2.1.2 Dipendenze esterne

La componente ha le seguenti dipendenze esterne:

- ***ChatKit***: libreria per implementare l'interfaccia grafica con *layout* simile ad una chat.
  - ***MessageListAdapter***: classe per la gestione dei messaggi nella chat.
  - ***MessageInput***: classe per l'implementazione del *widget* grafico per l'inserimento e l'invio del messaggio.
  - ***ImageLoader***: interfaccia per la gestione delle immagini e degli avatar nella chat.
  - ***MessageHolders***: classe per l'implementazione di vari media nella chat.
- ***Librerie AndroidX***: varie librerie per gestire il *lifecycle* dell'APP$_G$ e i *widget* grafici.
- ***Volley***: libreria per la gestione di chiamate HTTP$_G$.
  - ***RequestQueue***: classe per la gestione di una coda di chiamate HTTP$_G$.
  - ***Response***: classe che incapsula una richiesta HTTP$_G$ per effettuare la chiamata.
  - ***JsonObjectRequest***: classe per gestire chiamate HTTP$_G$ con *boby* in formato JSON.
  - ***NetworkResponse***: classe che contiene la risposta ad una chiamata HTTP$_G$.

2.1.3 Diagramma delle classi



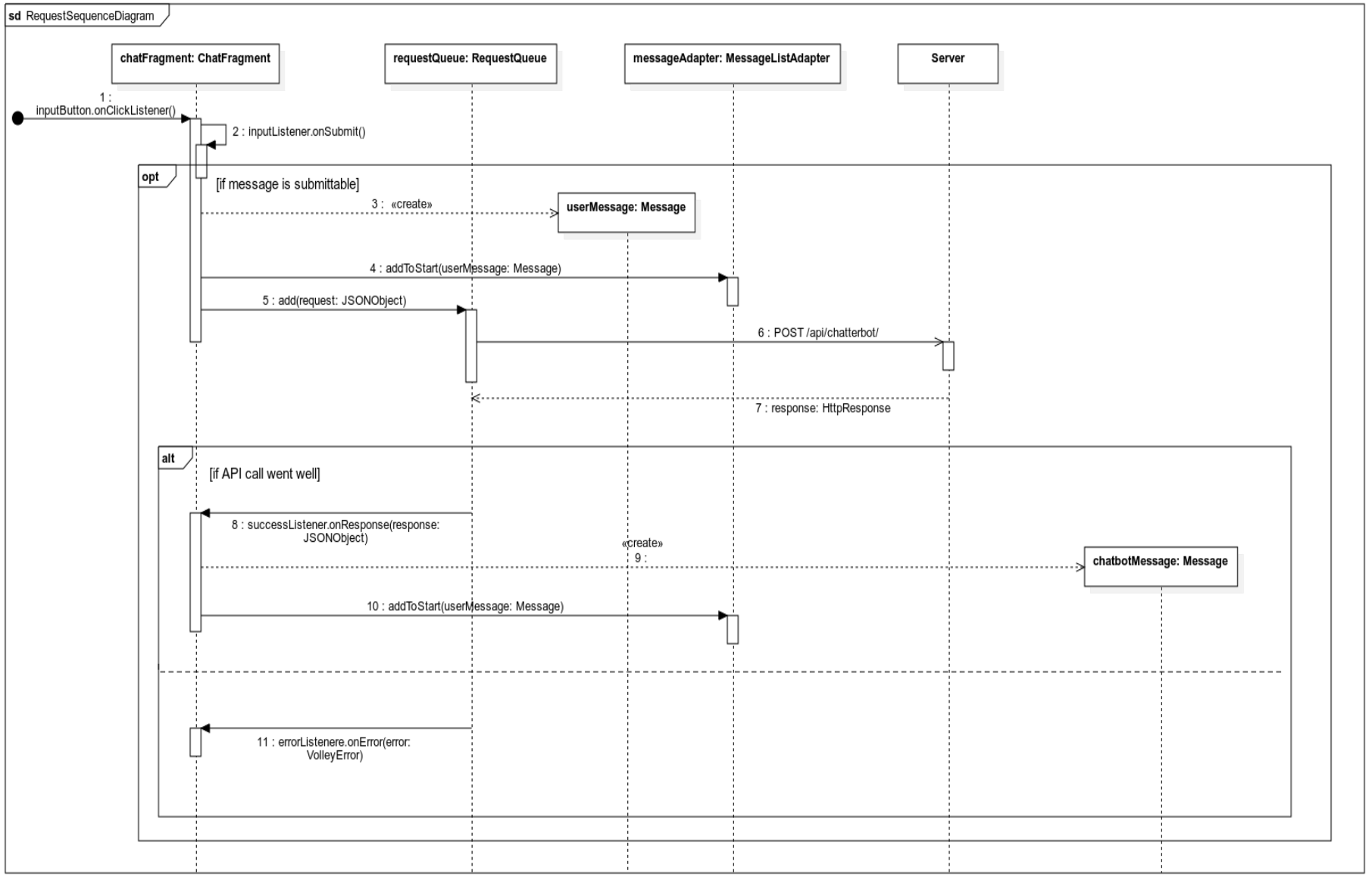
- ***AppCompactActivity***: classe base della libreria ANDROID$_G$ per tutte le attività dell'APP$_G$. Fornisce retrocompatibilità con *API level* più bassi.
- ***MainActivity***: classe che rappresenta l'attività base dell'APP$_G$. Deriva da *AppCompactActivity*. Necessita di saper costruire un *ChatFragment*, dato che lo va a creare e aggiungere alla vista ogni volta che l'attività viene creata. Fa l'*override* del metodo *onBackPressed* per gestire il caso in cui vi siano più *fragment* nel *backstack*.
- ***SettingsActivity***: classe che rappresenta l'attività di gestione delle impostazioni dell'APP$_G$. Deriva da *AppCompactActivity*. Necessita di saper costruire un *SettingsFragment*, dato che lo va a creare e aggiungere alla vista ogni volta che l'attività viene creata. Fa l'*override* del metodo *onBackPressed* per gestire il caso in cui vi siano più *fragment* nel *backstack*.

- ***Fragment***: classe base della libreria ANDROID$_G$ per gestire una particolare schermata di un'attività. Fornisce retrocompatibilità con *API level* più bassi.
- ***DialogFragment***: classe della libreria ANDROID$_G$ per gestire una particolare schermata di un'attività, con un *layout* simile ad un *modal dialog*. Fornisce retrocompatibilità con *API level* più bassi.
- ***PreferenceFragmentCompat***: classe della libreria ANDROID$_G$ per gestire una particolare schermata di impostazioni dell'APP$_G$. Fornisce tutte le funzionalità per la gestione delle impostazioni.
- ***ChatFragment***: classe che rappresenta la schermata principale dell'APP$_G$, dove è possibile chattare con il CHATBOT$_G$. Si occupa di gestire la conversazione, inserendo i messaggi inviati dall'utente e anche quelli ricevuti dal CHATBOT$_G$.
  Quando un messaggio viene inviato, si occupa di accodare la chiamata HTTP$_G$ relativa tramite la libreria *Volley*. Gestisce anche l'esito della chiamata, aggiungendo il messaggio alla conversazione se tutto è andato bene o notificando l'utente se qualcosa è andato storto; per implementare queste funzionalità va ad utilizzare varie interfacce.
  Fa l'*override* di vari metodi per gestire la creazione della schermata e la reazioni agli eventi scatenati dall'utente. Deve saper costruire ed usare le classi *Message* e *User*, per inserire nuovi messaggi *widget* grafico della conversazione.
  Infine ha una dipendenza verso *SettingsActivity* e *ApiKeyDialogFragment*, dato che l'utente può accedere alle impostazioni da questa schermata e può essere notificato nel caso in cui non abbia ancora inserito un'*api_key*.
- ***ApiKeyDialogFragment***: classe che rappresenta la schermata di una *dialog* informativa per l'utente. Compare nel caso in cui l'utente entra nell'APP$_G$ ma non ha ancora impostato un'*api_key*. Ha una dipendenza verso *SettingsActivity*, dato che l'utente può accedere alle impostazioni da questa schermata. Fa l'*override* di vari metodi per gestire la creazione della schermata e la reazioni agli eventi scatenati dall'utente.
- ***SettingsFragment***: classe che rappresenta la schermata di impostazioni dell'APP$_G$. Mostra le impostazioni modificabili (solo *api_key*) ed altre informazioni utili per l'utente. Fa l'*override* del metodo *onCreatePreferences* per gestire le modifiche dell'*api_key*.

- ***InputListener***: interfaccia della libreria *ChatKit* per la gestione di eventi legati all'invio dei messaggi.
- ***Listener***: interfaccia della libreria *Volley* per la gestione di risposte a chiamate HTTP$_G$ avvenute con successo. È parametrizzata dal tipo di richiesta HTTP$_G$ effettuata (utilizzata solo *JsonObjectRequest*).
- ***ErrorListener***: interfaccia della libreria *Volley* per la gestione di risposte a chiamate HTTP$_G$ non andate a buon fine.
- ***ImageLoader***: interfaccia della libreria *ChatKit* per la gestione del caricamento di immagini nella chat, sia su messaggi che sugli avatar degli *User*.
- ***IMessage***: interfaccia della libreria *ChatKit* che modella il comportamento di un generico messaggio della chat.
- ***IUser***: interfaccia della libreria *ChatKit* che modella il comportamento di un generico *user* della chat.

- ***RequestQueue***: classe della libreria *Volley* per la gestione di una coda di chiamate HTTP$_G$.
- ***MessageListAdapter***: classe della libreria *ChatKit* per la gestione della conversazione. Permette di aggiungere, gestire, modificare la conversazione e i suoi messaggi.
- ***Message***: classe concreta che va ad implementare l'interfaccia *IMessage*. Contiene, come campi privati, le informazioni fondamentali che contraddistinguono un messaggio (testo, *user*, data di invio).
- ***User***: classe concreta che va ad implementare l'interfaccia *IUser*. Contiene, come campi privati, le informazioni fondamentali che contraddistinguono uno *user* di una chat (nome ed eventuale *avatar*).

2.1.4 Diagramma di sequenza



Il diagramma descrive il *flow* dell'interazione tra le componenti per la gestione dell'invio di un messaggio da parte dell'utente e la ricezione della risposta dal CHATBOT$_G$. È stato inserito questo diagramma di sequenza, in quanto va a descrivere la funzionalità chiave dell'APP ANDROID$_G$.

Il funzionamento di una qualsiasi funzionalità è il seguente:

1. L'utente scrive un messaggio sulla barra di input e preme il tasto per inviare il messaggio. Questo scatena la *callaback* del button associato (*onClickListener*).
2. La *callback* va a chiamare l'*InputListener*, il quale valida il messaggio (controlla che non sia vuoto, ...).
3. Se la validazione è andata a buon fine, viene generato un nuovo *Message* da aggiungere alla conversazione e viene aggiunto a quest'ultima.
4. Viene poi generata e accodata una chiamata HTTP$_G$ POST contenente, nel *body*, il messaggio inserito dall'utente.
5. Non appena sarà possibile, la richiesta verrà evasa, attendendo risposta dal *server* in modo asincrono.
6. Quando il CHATBOT$_G$ risponderà, sono presenti due diversi scenari a seconda dell'esito. In caso la richiesta HTTP$_G$ sia andata a buon fine:
   (a) Viene generato un nuovo *Message*, contenente la risposta del CHATBOT$_G$.
   (b) Il messaggio precedentemente creato viene aggiunto alla fine della conversazione.

   Nel caso in cui la richiesta non fosse andata a buon fine:
   (a) Cerca di capire la natura dell'errore.
   (b) Informa l'utente, mostrando un messaggio di errore.

2.1.5 Design Pattern

Non sono stati utilizzati *design pattern*.

2.2 Web Client

2.2.1 Introduzione

Il *web client* è un sito *web* che fornisce un'interfaccia utente per dialogare in modo comodo, semplice e rapido con il CHATBOT$_G$. Questa componente permette all'utente di:

- Iniziare una conversazione con il CHATBOT$_G$ ogni volta che si accede al sito.
- Mandare richieste al CHATBOT$_G$ e ricevere risposte da esso.
- Impostare, modificare, eliminare l'*api_key* (punto di autenticazione).

Tutte queste operazioni vengono svolte dall'utente senza la necessità che esso sia a conoscenza di come il SISTEMA EMT$_G$ aziendale funzioni.

2.2.2 Dipendenze esterne

La componente ha le seguenti dipendenze esterne:

- ***JavaScript***: linguaggio per le gestione del comportamento di pagine *web* lato *client*.
  - ***JavaScript Cookie***: libreria JAVASCRIPT$_G$ che fornisce API$_G$ per le gestione dei COOKIE$_G$.
- ***JQuery***: libreria JAVASCRIPT$_G$ designata per semplificare le operazioni con il DOM$_G$, la gestione di eventi e chiamate AJAX$_G$.
- ***Bootstrap***: CSS$_G$ FRAMEWORK$_G$ orientato allo sviluppo di APPLICAZIONI WEB$_G$ responsive, con il paradigma *mobile-first*.

2.2.3 Diagramma delle classi

Data la semplicità dell'interfaccia *web* da sviluppare, si è optato per non utilizzare nessun FRAMEWORK$_G$ per lo sviluppo del FRONT-END$_G$. Per questa ragione non sono presenti classi, ma una semplice pagina HTML$_G$ dinamica, che va ad implementare la componente *template* del *design pattern MVT* (2.3.5).

Il layout della pagina mira ad emulare l'interfaccia di una classica chat delle principali app di messaggistica. Gli elementi principali dell'interfaccia sono:

- Box dove vengono inseriti i messaggi man mano che la conversazione prosegue.
- Campo per l'inserimento di testo, dove l'utente può inserire il messaggio che vuole inviare al CHATBOT$_G$.
- Pulsante per l'invio del messaggio.
- Pulsante per accedere alle impostazioni, dove è possibile cambiare l'*api_key* con si viene autenticati durante le varie operazioni nel SISTEMA EMT$_G$ aziendale.

2.2.4 Diagramma di sequenza

Non essendoci un diagramma delle classi, non può esserci un corrispettivo diagramma di sequenza. Tuttavia, grazie all'utilizzo di *script* JAVASCRIPT$_G$, si può evidenziare una sequenzialità di azioni scatenate da un evento.

- **Accesso alla chat**

  1. Si accede all'indirizzo web della pagina.
  2. Non appena la pagina è stata caricata, uno *script* effettuerà una chiamata HTTP$_G$ GET (AJAX$_G$ *call*) all'ENDPOINT$_G$ "/api/chatterbot", per istanziare la sessione ed iniziare la conversazione.

- **Conversazione con il CHATBOT$_G$**

  1. L'utente scrive il messaggio nella casella di inserimento testo e preme il tasto per inviare il messaggio.
  2. Viene inserito nella chat il messaggio inviato dall'utente.
  3. Uno script, attraverso una AJAX$_G$ *call*, effettua una chiamata HTTP$_G$ POST all'ENDPOINT$_G$ "/api/chatterbot", per ottenere una risposta dal CHATBOT$_G$.
  4. Quando la chiamata ritorna, la risposta viene inserita nella chat.

- **Modifica della *api_key***

  1. L'utente entra per la prima volta nel sito o preme il pulsante per entrare nelle impostazioni.
  2. Viene mostrata una *modal dialog* contenente il link per effettuare l'autenticazione nei server di Imola Informatica.
  3. L'utente a questo punto può effettuare l'autenticazione se sprovvisto di *api_key* e successivamente inserirla nell'apposita casella di testo.
  4. L'utente preme il pulsante "*Salva*", salvando così l'*api_key* inserita nel *Local Storage* (memoria persistente a lato *client*).

  L'*api_key*, quando impostata, viene automaticamente inserita nell'*header* di qualsiasi chiamata HTTP$_G$ effettuata.

2.3 Server

2.3.1 Introduzione

Questa componente è il cuore del sistema nel suo complesso. Essa, attraverso vari **LogicAdapter**, esegue le seguenti operazioni:

- Riceve le richieste HTTP$_G$ provenienti dai vari client (browser, APP ANDROID$_G$).
- Interpreta le richieste, chiedendo all'utente (client) eventuali informazioni mancanti per eseguire l'operazione desiderata.
- Determina a quale microservizio di Imola Informatica deve interfacciarsi per eseguire l'operazione richiesta.
- Decodifica la risposta ottenuta al microservizio e ritorna al client una risposta *human-friendly*.

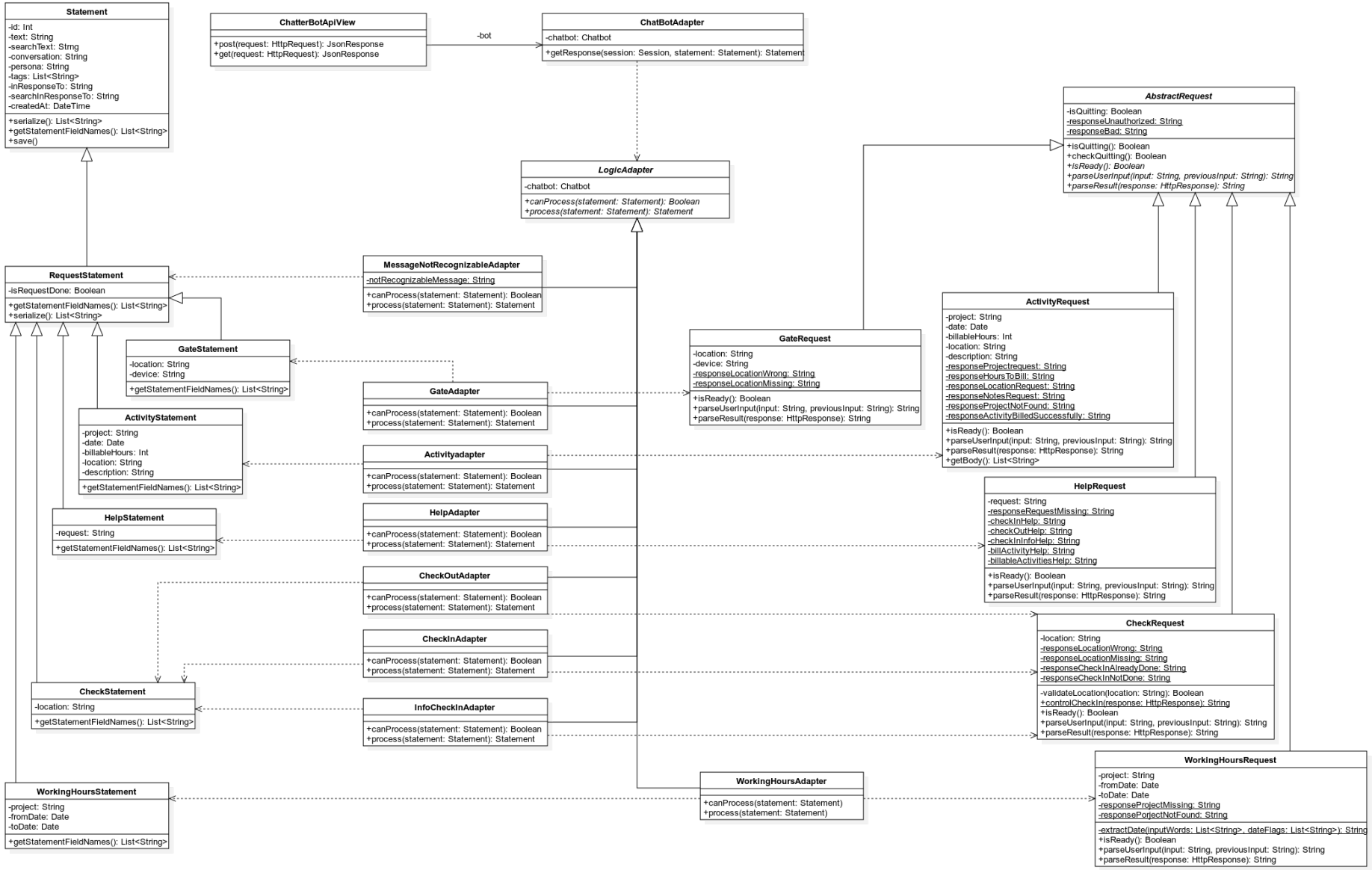
2.3.2 Dipendenze esterne

La componente ha le seguenti dipendenze esterne:

- ***Chatterbot***: libreria che fornisce le funzionalità principali di interpretazione e risposta ai messaggi.
  - ***LogicAdapter***: classe astratta che rappresenta l'interfaccia degli *adapter* che gestiscono e interpretano le varie richieste degli utenti.
  - ***Statement***: classe concreta che rappresenta una frase detta da qualcuno.
  - ***Chatbot***: classe concreta che racchiude tutte le funzionalità di un CHATBOT$_G$.
- ***Levenshtein***: libreria PYTHON$_G$ per il calcolo della distanza tra una parola/frase corretta e una parola/frase fornita. Viene utilizzata per implementare la tolleranza agli errori nei messaggi inviati dall'utente.
- ***Librerie*** **PYTHON$_G$**: varie librerie offerte dal linguaggio, quali:
  - ***Requests***: libreria per effettuare chiamate HTTP$_G$.
  - ***Datetime***: libreria per modellare la gestione del tempo.
  - ***RE***: libreria per effettuare operazioni utilizzando le espressioni regolari.
  - ***Optional***: libreria per modellare correttamente casi in cui il risultato di un'operazione non venga calcolato correttamente.

Un elenco completo delle dipendenze di questa componente può essere trovato nel file *requirements.txt*, all'interno della cartella *server*. La stragrande maggioranza delle librerie da cui si dipende sono necessarie per il corretto funzionamento della libreria *chatterbot*.

2.3.3 Diagramma delle classi



- ***ChatterBotApiView***: rappresenta l'ENDPOINT$_G$ per le richieste HTTP$_G$ ricevute dai client. È in grado di rispondere a chiamate POST e GET sull'ENDPOINT$_G$ "/api/chatterbot". Istanzia una sessione per ogni client che si collega. Si occupa principalmente di spacchettare la richiesta e prepararla per il *ChatBotAdapter*. Una volta che quest'ultimo ha elaborato la risposta, impacchetta la risposta per spedirla al client.
- ***ChatBotAdapter***: rappresenta il ponte di collegamento tra *ChatterBotApiView* e i *LogicAdapter* del *ChatBot*. Riceve una richiesta da *ChatterBotApiView* e, tramite la sessione da lui impostata, determina se c'è una richiesta pendente per quel client o se deve elaborarne una nuova. Nel primo caso fa proseguire la richiesta interpellando il *LogicAdapter* che gestisce quel tipo di richiesta; nel secondo caso cerca di capire quale *LogicAdapter* può gestire la richiesta e gliela fa processare. Al termine aggiorna la sessione del client con lo stato della richiesta.

- ***Statement***: classe della libreria esterna *chatterbot*, rappresenta una singola entità parlata, una frase che qualcuno può dire. È stata riportata per completezza del diagramma. Contiene al suo interno il metodo *serialize()*, che permette un rapido impacchettamento della risposta elaborata dal CHATBOT$_G$.
- ***RequestStatement***: rappresenta la classe base per gli *statement* più elaborati utilizzati dal sistema. Deriva da *Statement*, a cui aggiunge lo stato della richiesta dell'utente.
- ***HelpStatement***: rappresenta uno *statement* in cui l'utente chiede al CHATBOT$_G$ le istruzioni da seguire per effettuare una determinata operazione. Aggiunge il campo *request*, che identifica il tipo di operazione per cui l'utente sta chiedendo le istruzioni.
- ***ActivityStatement***: rappresenta uno *statement* in cui l'utente chiede di consuntivare un'ATTIVITÀ$_G$ nel sistema EMT$_G$ aziendale. Aggiunge i campi necessari ad eseguire la richiesta: progetto, data, ore lavorate, località ed eventuali note aggiuntive.
- ***CheckStatement***: rappresenta uno *statement* in cui l'utente chiede di effettuare un'operazione di ingresso o uscita da una determinata sede. Ha come attributo aggiuntivo la sede in oggetto.
- ***WorkingHoursStatement***: rappresenta uno *statement* in cui l'utente chiede di recuperare le ore che ha consuntivato in un determinato progetto. I campi aggiuntivi sono il nome del progetto ed eventuale data di inizio e fine per la ricerca.
- ***GateStatement***: rappresenta uno *statement* in cui l'utente chiede di aprire il cancello principale di una delle sedi. I campi aggiuntivi sono la località della sede e il dispositivo con cui interagire (in questo caso sarà sempre il cancello, ma così facendo si è aperti ad estensioni future).

- ***LogicAdapter***: classe astratta della libreria esterna *chatterbot*, modella il modo in cui il CHATBOT$_G$ deve comportarsi in base allo *statement* ricevuto in input. In altre parole determina la logica con cui il CHATBOT$_G$ seleziona la risposta in relazione allo *statement* inserito dall'utente. I due principali metodi sono:

  - *can_process*: determina se quell'*adapter* è in grado di gestire la richiesta.
  - *process*: processa la richiesta ricevuta in input.

  È stata riportata per completezza del diagramma.

- ***MessageNotRecognizableAdapter***: rappresenta un *adapter* di *fallback*. Quando nessuno degli altri *adapter* è in grado di processare una determinata richiesta, si ricade su questo, il quale avvisa l'utente che il CHATBOT$_G$ non riesce a capire cosa stia chiedendo.
- ***InfoCheckInAdapter***: rappresenta un *adapter* per gestire la richiesta di informazioni sullo stato del check-in per un determinato utente. Per poter operare necessita di saper costruire un *CheckStatement* e conoscere l'interfaccia pubblica di una *CheckRequest*. Recupera l'informazione all'ENDPOINT$_G$ di Imola Informatica "/v1/locations/presence/me" tramite l'utilizzo di una richiesta HTTP$_G$ GET.
- ***CheckInAdapter***: rappresenta un *adapter* per gestire la richiesta di registrare il check-in in una determinata sede aziendale da parte dell'utente. Per poter operare necessita di saper costruire un *CheckStatement* e conoscere l'interfaccia pubblica di una *CheckRequest*. Tramite l'utilizzo di una chiamata HTTP$_G$ POST all'ENDPOINT$_G$ di Imola Informatica "/v1/locations/location/presence" inserisce l'informazione nel SISTEMA EMT$_G$ aziendale.
- ***CheckOutAdapter***: rappresenta un *adapter* per gestire la richiesta di registrare il check-out da una determinata sede aziendale da parte dell'utente. Per poter operare necessita di saper costruire un *CheckStatement*. Tramite l'utilizzo di una chiamata HTTP$_G$ DELETE all'ENDPOINT$_G$ di Imola Informatica "/v1/locations/location/presence" inserisce l'informazione nel SISTEMA EMT$_G$ aziendale.
- ***HelpAdapter***: rappresenta un *adapter* per gestire la richiesta dell'utente di ricevere istruzioni per dialogare con il chatbot. Per poter operare necessita di saper costruire un *HelpStatement* e conoscere parte delll'interfaccia pubblica di una *HelpRequest*.
- ***ActivityAdapter***: rappresenta un *adapter* per gestire la richiesta di inserimento attività nel SISTEMA EMT$_G$ aziendale. Per poter operare necessita di saper costruire un *ActivityStatement* e conoscere l'interfaccia pubblica di una *AcivityRequest*. Tramite l'utilizzo di una chiamata HTTP$_G$ POST all'ENDPOINT$_G$ di Imola Informatica "/v1/projects/project/activities/me" inserisce la consuntivazione nel SISTEMA EMT$_G$ aziendale.
- ***WorkingHoursAdapter***: rappresenta un *adapter* per gestire la richiesta di recupero delle attività consuntivate nel SISTEMA EMT$_G$ aziendale. Per poter operare necessita di saper costruire un *WorkingHoursStatement* e conoscere l'interfaccia pubblica di una *WorkingHoursRequest*. Tramite l'utilizzo di una chiamata HTTP$_G$ GET all'ENDPOINT$_G$ di Imola Informatica "/v1/projects/project/activities/me" recupera la consuntivazione per un determinato progetto dal SISTEMA EMT$_G$ aziendale.
- ***GateAdapter***: rappresenta un *adapter* per gestire la richiesta di apertura del cancello di una delle sedi aziendali. Per poter operare necessita di saper costruire un *GateStatement* e conoscere l'interfaccia pubblica di una *GateRequest*. Tramite l'utilizzo di una

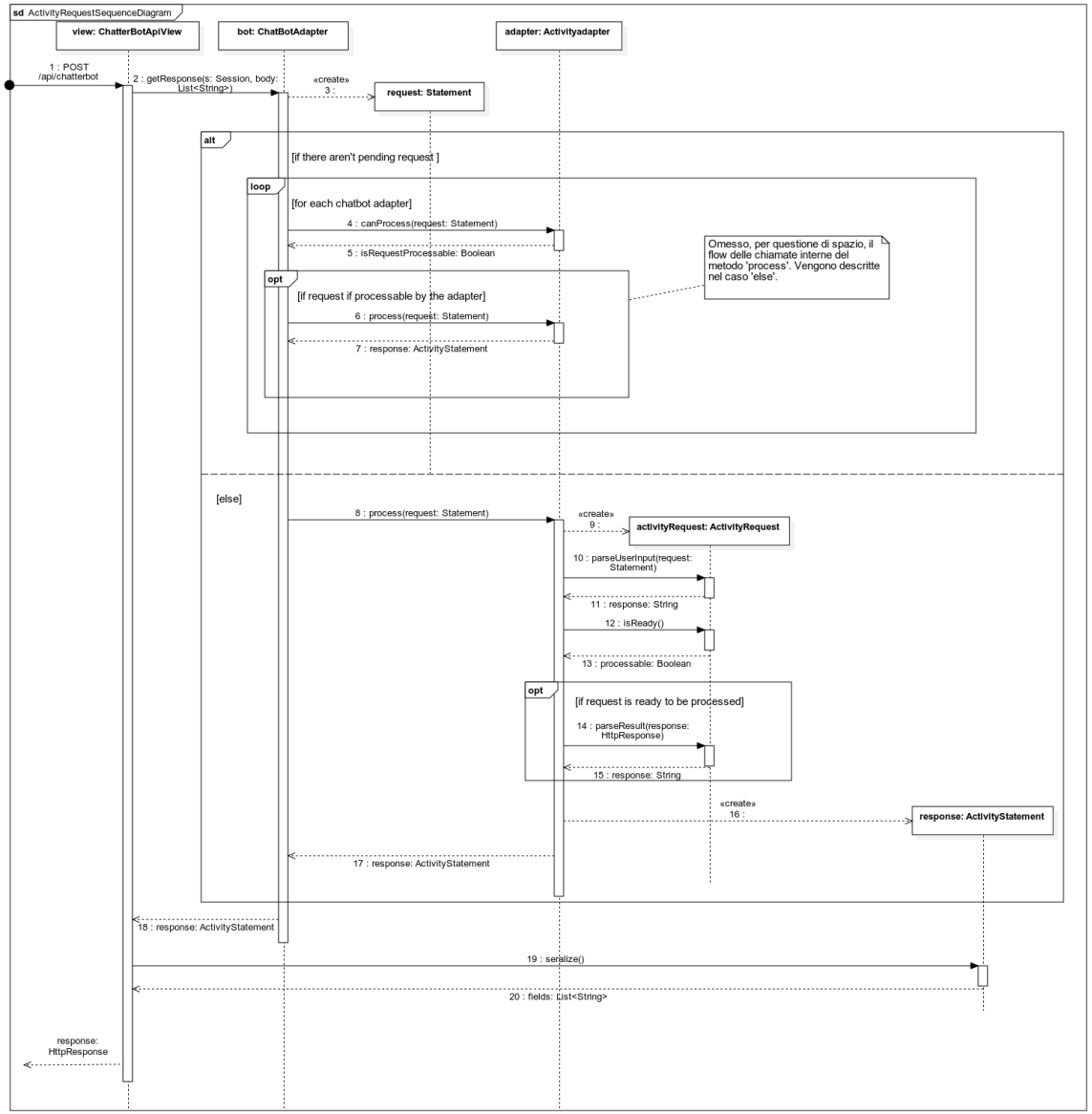
chiamata HTTP$_G$ PUT all'ENDPOINT$_G$ di Imola Informatica "/v1/locations/location/devices/device/status" invia il comando per aprire il cancello.

- ***AbstractRequest***: classe astratta che modella una particolare richiesta ricevuta da un client. Contiene al suo interno i campi necessari per eseguire la chiamata al SERVIZIO REST$_G$ corrispondente, oltre ad alcuni metodi di utilità per gestire le richieste degli utenti. Si occupa principalmente di interpretare gli *statement* ricevuti dai vari client, estrapolando informazioni utili per eseguire la richiesta ed elaborare poi una risposta. I principali metodi sono:
  - *isQuitting*: indica se l'utente ha deciso di abortire l'operazione.
  - *checkQuitting*: controllo che l'utente abbia espresso di abortire l'operazione.
  - *isReady*: controlla che la richiesta abbia tutte le informazioni di cui ha bisogno per essere processata.
  - *parseUserInput*: fa il parsing di un messaggio inviato dall'utente per estrapolarne informazioni e rispondere in maniera corretta.
  - *parseResult*: in seguito alla chiamata al MICROSERVIZIO$_G$, fa il parsing della risposta HTTP$_G$, rendendola *human-friendly*.

  Ha al suo interno un insieme di stringhe statiche che rappresentano le possibili risposte che il CHATBOT$_G$ può dare a seconda delle situazioni.
- ***WorkingHoursRequest***: rappresenta una richiesta per gestire il recupero delle attività consuntivate nel SISTEMA EMT$_G$ aziendale. Contiene dei campi di dominio quali: progetto di riferimento, data inizio ricerca e data fine ricerca. Ha al suo interno un insieme di stringhe statiche che rappresentano le possibili risposte che il CHATBOT$_G$ può dare a seconda delle situazioni.
- ***CheckRequest***: rappresenta una richiesta per gestire lo stato del CHECK-IN$_G$ / CHECK-OUT$_G$ di un determinato utente. Contiene un campo che indica la località della sede da cui si vuole effettuare l'operazione. Ha al suo interno un insieme di stringhe statiche che rappresentano le possibili risposte che il CHATBOT$_G$ può dare a seconda delle situazioni.
- ***HelpRequest***: rappresenta una richiesta per gestire il supporto che viene dato all'utente per capire come effettuare una determinata operazione. Contiene un campo che indica il tipo di richiesta per la quale l'utente sta chiedendo aiuto. Ha al suo interno un insieme di stringhe statiche che rappresentano le possibili risposte che il CHATBOT$_G$ può dare a seconda delle situazioni e della particolare richiesta per cui vuole ricevere supporto.
- ***ActivityRequest***: rappresenta una richiesta per gestire l'inserimento di una attività da consuntivare nel SISTEMA EMT$_G$ aziendale. Contiene dei campi di dominio quali: progetto di riferimento, data di svolgimento, numero di ore da consuntivare, località ed eventuali note aggiuntive. Ha al suo interno un insieme di stringhe statiche che rappresentano le possibili risposte che il CHATBOT$_G$ può dare a seconda delle situazioni.

- ***GateRequest***: rappresenta una richiesta per gestire l'apertura del cancello di una delle sedi aziendali. Contiene dei campi di dominio quali: località, e dispositivo con cui interagire. Ha al suo interno un insieme di stringhe statiche che rappresentano le possibili risposte che il CHATBOT$_G$ può dare a seconda delle situazioni.

2.3.4 Diagramma di sequenza



Il diagramma descrive il *flow* di una funzionalità offerta dal CHATBOT$_G$. È stato inserito un solo diagramma di sequenza, in quanto quelli delle altre funzionalità sono pressoché identici, fatta eccezione per le classi utilizzate (specifiche della funzionalità, ovvero *statement* e *request*).

Il funzionamento di una qualsiasi funzionalità è il seguente:

1. Il *client* effettua una chiamata POST all'ENDPOINT$_G$ "/api/chatterbot", la quale contiene nel *body* il messaggio inviato dall'utente.
2. La *view* riceve la chiamata, estrapola il messaggio dell'utente e lo inoltra al *bot*.
3. Il bot, grazie all'utilizzo delle sessioni, determina se il messaggio dell'utente è il prosieguo di una conversazione già iniziata (richiesta pendente) o l'inizio di una nuova conversazione. Nel primo caso:
   (a) Recupera l'*adapter* che stava gestendo la richiesta, e gli fa elaborare la risposta da restituire al *client*.
   (b) L'*adapter* ricostruisce lo stato della richiesta, sfruttando le informazioni che erano state precedentemente estrapolate dalla conversazione con l'utente. Durante questo processo va ad allocare un oggetto discendente da *AbstractRequest*, a seconda del tipo di *request* che è in grado di gestire.
   (c) L'*adapter* utilizza poi l'oggetto creato per fare il *parsing* del messaggio dell'utente, al fine di estrapolarne nuove informazioni utili per eseguire la richiesta.
   (d) L'*adapter*, dopo il *parsing*, controlla se l'oggetto *request* possiede ora tutte le informazioni necessarie per eseguire l'operazione richiesta. Se sì:
      i. Effettua la richiesta all'ENDPOINT$_G$ corretto dei SERVIZI REST$_G$ esposti da Imola Informatica.
      ii. Fa effettuare il *parsing* della risposta HTTP$_G$ alla *request* precedentemente allocata, in modo tale da avere una risposta in formato leggibile per l'essere umano.
   (e) Infine esso crea uno *statement*, consono con il tipo di richiesta che sta gestendo, comprensivo di risposta e informazioni estrapolate fino a quel momento sull'operazione da effettuare.

   Nel secondo caso, il *bot*:

   (a) Scorre tutti gli *adapter* allocati nel CHABOT$_G$:
      i. Chiede ad ognuno se sono in grado di gestire la richiesta fatta dall'utente.
      ii. Se sì, chiede ad esso di processarla, con procedimento analogo a 3. Nel caso vi fossero più *adapter* in grado di gestire la richiesta, viene scelto quello che da l'indice di confidenza più alto. (La confidenza è un numero compreso tra 0 e 1, che indica quanto la risposta fornita dall'*adapter* è consona rispetto alla richiesta, secondo lo stesso *adapter*).

4. Una volta ottenuta la risposta, il *bot* aggiorna le informazioni sulla sessione e restituisce lo *statement* di risposta alla *view*.
5. La *view* "serializza" lo *statement* ricevuto e impacchetta la risposta per poi spedirla al *client*.

2.3.5 Design Pattern

Data la semplicità della soluzione architetturale trovata per risolvere il problema, non è stato possibile utilizzare alcun tipo di *design pattern* specifico.
Tuttavia il FRAMEWORK$_G$ DJANGO$_G$ utilizza, per sua natura, un *desgin pattern* architetturale *MVT (Model - View - Template)*:

- ***Model***: struttura dati utilizzata dietro l'intera applicazione, spesso rappresentata da un *database*. Nella soluzione non è presente nessun modello, in quanto non è compito del CHATBOT$_G$ tenere traccia delle operazioni effettuate dagli utenti.
- ***View***: contiene le principali funzionalità dell'architettura DJANGO$_G$. Qui sono presenti le classi di *business* e quindi tutta la *business logic* che sarà responsabile di rispondere agli input ricevuti dai vari *client*. Sono inoltre presenti le "viste" (ENDPOINT$_G$ per richieste HTTP$_G$) con le quali il *server* interagisce. Qui sono state implementate tutte le classi di dominio presenti in 2.3.3.
- ***Template***: parte utilizzata per la rappresentazione di pagine HTML$_G$ dinamiche nei *web browser*. La componente è descritta in 2.2.

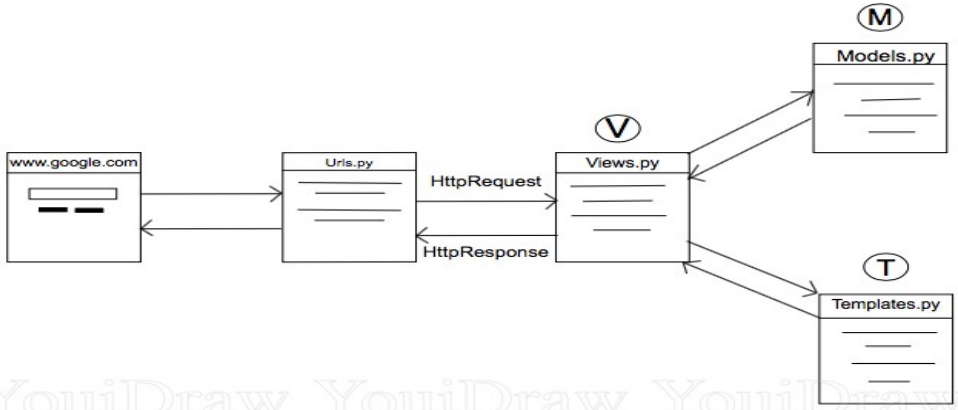


Figura 2: Breve rappresentazione dell'architettura *MVT*.

2.4 API REST$_G$ CHATBOT$_G$

Il sito web da cui è possibile accedere al CHABOT$_G$ è: imola-bot4me.herokuapp.com. Questo è l'indirizzo base per indicare gli API ENDPOINT$_G$.

2.4.1 Inziare una conversazione con il CHATBOT$_G$

API ENDPOINT$_G$

/api/chatterbot/

Chiamata HTTP$_G$

GET

Parametri della richiesta

Nessuno.

HTTP$_G$ Headers

- **Content-type**: "*application/json*"
- **Authorization**: *api_key* (facoltativa)

Risposte

HTTP$_G$ Status Code	Body (JSON)	Descrizione
200	`{ "text": string }`	La sessione è stata correttamente inizializzata e viene ritornato al *client* un messaggio di benvenuto.

Descrizione

All'inizio di ogni conversazione con il CHATBOT$_G$ il client dovrà fare una richiesta a questo ENDPOINT$_G$, in modo tale che venga registrata una sessione per l'utente che vuole dialogare con esso. Il *client* riceverà come risposta i *cookie* di sessione ("*Set-Cookie*") e un messaggio di benvenuto.

2.4.2 Conversare con il CHATBOT$_G$

API ENDPOINT$_G$

/api/chatterbot/

Chiamata HTTP$_G$

POST

Parametri della richiesta

Tipo	Body (JSON)	Descrizione
JSON Object	`{ "text": string }`	L'attributo "*text*" contiene il messaggio inviato dall'utente al CHATBOT$_G$.

HTTP$_G$ Headers

- **Content-type**: "*application/json*"
- **Authorization**: *api_key* (facoltativa, vincolante affinché l'operazione vada a buon fine).

Risposte

HTTP$_G$ Status Code	Body (JSON)	Descrizione
200	`{ "text": string, "in_response_to": string, "created_at": string }`	Il messaggio è stato correttamente interpretato. Nel *body* è contenuta la risposta per l'utente. Sono stati riportati solamente i campi essenziali, la lista completa degli attributi è equivalente ai campi dato dei singoli *statement*, che variano a seconda del tipo di richiesta effettuata dall'utente.
400	`{ "text": string }`	Non è stato specificato un campo "*text*" nella domanda, quindi il CHATBOT$_G$ non è in grado di determinare quale sia l'input dell'utente.

Descrizione

ENDPOINT$_G$ dove il CHATBOT$_G$ riceve i messaggi inviati dall'utente. Da qui interpreta il suo messaggio, e se tutto va per il meglio, risponde con la risposta elaborata.

2.5 API REST$_G$ Imola Informatica

Le API REST$_G$ fornite da Imola Informatica consentono di svolgere tutte le operazioni basilari supportate dal CHATBOT$_G$. Queste sono state sviluppate direttamente dal proponente, seguendo la filosofia di un'architettura a microservizi.
Imola ha messo a disposizione una SWAGGER$_G$ UI, da cui è semplice:

- Capire quali SERVIZI REST$_G$ sono disponibili.
- Capire quali parametri sono necessari per eseguire le chiamate a questi servizi.
- Capire l'esito di una particolare chiamata.
- Testare i vari servizi.

Lo schema delle API$_G$ esposte è facilmente reperibile a questo link: API BOT4ME - Imola Informatica.

3 Tecnologie per lo sviluppo

Vengono di seguito elencate le tecnologie che sono state utilizzate per lo sviluppo delle componenti precedentemente descritte.

3.1 App client

3.1.1 Kotlin

Linguaggio di programmazione *general purpose* ad alto livello, fortemente tipizzato e orientato alla *Object Oriented Programming*. È stato utilizzato per sviluppare l'APPLICAZIONE ANDROID$_G$ (*client*) che si interfaccia con il CHATBOT$_G$ (*server*). Sono state utilizzate altre librerie (*Volley*, ...) a supporto delle funzionalità da esporre.
L'IDE$_G$ utilizzato per lo sviluppo con questo linguaggio di programmazione è stato ANDROID STUDIO$_G$. Si è utilizzato in parte anche il linguaggio XML per definire parte della *user interface* tramite ANDROID STUDIO$_G$.

3.1.2 ChatKit

Libreria ANDROID$_G$ per l'implementazione e la gestione di un'interfaccia utente simile a quella di una *chat* di una normale app di messaggistica. L'alta flessibilità e possibilità di personalizzazione hanno permesso di utilizzarla al meglio per fornire all'utente un'interfaccia semplice e intuitiva.
L'IDE$_G$ utilizzato per sviluppo delle componenti grafiche e la loro gestione tramite *ChatKit* è stato ANDROID STUDIO$_G$.

3.1.3 Gradle

Strumento di *build automation* per lo sviluppo software che permette un ampio pacchetto di azioni, supporta Kotlin. Tra le azioni vi sono: compilazione, testing, deployment, pubblicazione artefatti e gestione delle dipendenze. È stato utilizzato come strumento di *build automation* per lo sviluppo dell'APP ANDROID$_G$, in particolare per gestire le dipendenze con librerie esterne.

3.2 Web client

3.2.1 HTML5

Linguaggio di *markup*, standard W3C per documenti visualizzabili attraverso un web browser. È stato utilizzato per definire la pagina web con cui l'utente si interfaccia per dialogare con il CHATBOT$_G$.
L'IDE$_G$ utilizzato per sviluppo della singola pagina web è stato PYCHARM$_G$, attraverso l'utilizzo di un *plugin* che estende l'utilizzo dell'IDE$_G$.

3.2.2 CSS3

Linguaggio di formattazione per documenti HTML5. È stato utilizzato per definire la parte grafica della pagina web esposta all'utente, ponendo l'attenzione sull'usabilità e prediligendo il paradigma *mobile-first*.

L'IDE$_G$ utilizzato per sviluppo della singola pagina web è stato PyCharm$_G$, attraverso l'utilizzo di un *plugin* che estende l'utilizzo dell'IDE$_G$.

3.2.3 JavaScript

Linguaggio di programmazione utilizzato per rendere dinamica una pagina web. È stato utilizzato per definire degli *handler* a particolari eventi scatenati dall'utente all'interno della pagina web.
L'IDE$_G$ utilizzato per sviluppo della singola pagina web è stato PyCharm$_G$, attraverso l'utilizzo di un *plugin* che estende l'utilizzo dell'IDE$_G$.

3.2.4 Bootstrap

Framework$_G$ CSS *open-source* orientato allo sviluppo *responsive, mobile-first*. È stato utilizzato per definire in modo più semplice e veloce l'interfaccia utente della pagina web.

3.2.5 JQuery

Libreria JavaScript$_G$ designata per semplificare la gestione del DOM$_G$, della sua *appearance* e degli eventi. È stato utilizzato in particolare per gestire gli eventi scatenati dall'utente e per effettuare le chiamate HTTP$_G$ verso il *server*.

3.3 Server

3.3.1 Python

Linguaggio di programmazione *general purpose* ad alto livello, adatto allo sviluppo di applicazioni distribuite e orientato alla *Object Oriented Programming*. È stato utilizzato per sviluppare il *back-end* del sistema. Sono state utilizzate altre librerie offerte da *python* per agevolare lo sviluppo.
L'IDE$_G$ utilizzato per lo sviluppo con questo linguaggio di programmazione è stato PyCharm$_G$.

3.3.2 Django

Web framework$_G$ *open-source* scritto in python$_G$ per lo sviluppo di applicazioni web. È stato utilizzato per l'esposizione di un *server* in grado di interfacciarsi con i vari client attraverso uno strato di *API REST* e per gestire le chiamate HTTP$_G$ ai servizi REST$_G$ di Imola Informatica.
L'IDE$_G$ utilizzato per lo sviluppo con questo framework$_G$ è stato PyCharm$_G$: esso fornisce un *setup* iniziale e un ambiente di esecuzione virtuale comodo per le fasi di sviluppo e di *debugging*.

3.3.3 Chatterbot

Libreria python$_G$ per generare risposte automatiche a messaggi inseriti dall'utente, utilizzando algoritmi di intelligenza artificiale per trovare la migliore risposta possibile. È stata utilizzata per emulare il comportamento di un chatbot$_G$ nel *server*. Grazie alla sua flessibilità è stato facilmente possibile implementare degli *adapter* che modellano e gestiscono le varie richieste a cui il chatbot$_G$ deve saper rispondere.

3.3.4 Heroku

Platform-as-a-service che offre la possibilità di avere gratuitamente una spazio *web* con un interprete PYTHON$_G$. Il servizio è stato utilizzato per effettuare il *deploy* del *server* nel *web*.