



**Università Politecnica delle Marche**

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

# **Sviluppo di un Chatbot Rasa per la gestione delle prenotazioni di un Hotel**

---

Corso di Data Science

Professore

Prof. Domenico Ursino

Gruppo 1

Michele Pasqualini

Denil Nicolosi

Eris Prifti

Anno accademico 2021-2022

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Tecnologie Utilizzate</b>	<b>2</b>
<b>3</b>	<b>Framework Rasa</b>	<b>3</b>
3.1	NLU . . . . .	4
3.2	Slots . . . . .	4
3.3	Rules . . . . .	5
3.4	Stories . . . . .	5
3.5	Actions . . . . .	6
3.6	Pipeline . . . . .	7
3.7	Integrazione con Telegram . . . . .	8
<b>4</b>	<b>Funzionalità aggiuntive</b>	<b>9</b>
4.1	Test stories . . . . .	9
4.1.1	Risultati . . . . .	9
4.2	File CSV . . . . .	12
<b>5</b>	<b>Chatbot hotel</b>	<b>12</b>
5.1	Help command . . . . .	12
5.2	Visualizzazione tipologie di stanze . . . . .	13
5.3	Prenotazione di una stanza . . . . .	14
5.4	Visualizzare una prenotazione . . . . .	14
5.5	Modificare una prenotazione . . . . .	14
5.6	Eliminare una prenotazione . . . . .	15
5.7	Organizzare una pulizia . . . . .	15
5.8	Visualizzare l'orario di pulizia di una stanza . . . . .	16
5.9	Modificare la pulizia di una stanza . . . . .	16
5.10	FAQs . . . . .	16

# 1 Introduzione

In questo progetto l'obiettivo è quello di realizzare un chatbot che si occupa di svolgere le principali funzioni per la gestione di un hotel. Grazie a questo chatbot si potranno intrattenere conversazioni per ottenere servizi, simulando il comportamento di un operatore umano. In questo caso, si possono prenotare le stanze dell'hotel, la pulizia della stanza, mostrare informazioni utili sulla struttura, le modalità di check-in e check-out e altro.

Negli ultimi anni i chatbot hanno assunto sempre maggiore popolarità, grazie anche agli assistenti vocali come Siri, Alexa, e l'assistente Google. I chatbot utilizzano diverse tecnologie, come ad esempio:

- *Natural Language Processing* (NLP) : permette di dividere l'input dell'utente in frasi e parole.
- *Natural Language Understanding* (NLU) : NLU aiuta il Chatbot a capire ciò che l'utente ha detto usando sia oggetti linguistici generali che specifici del dominio come i costrutti lessici, sinonimi e temi.
- *Natural Language Generation* (NLG): fornisce un'esperienza significativa e personalizzata al di là delle risposte previste.

Principalmente i chatbot si possono suddividere in:

- Chatbot basati su regole
- Chatbot A.I.

La prima tipologia, possiamo paragonarla a delle FAQ interattive, e sono strumenti molto elementari che usano istruzioni come *if/then* e sono programmati per riconoscere alcune frasi e rispondere di conseguenza, seguendo le regole che gli sono state date.

La seconda tipologia invece, quelli basati su A.I. risultano essere più conversazionali, guidati dai dati e predittivi; questi tipi di Chatbot sono generalmente più sofisticati, interattivi e personalizzati.

Secondo lo studio di Gartner, il quale prevede, attraverso alcune stime, che il 40% delle applicazioni Chatbot/assistente virtuale di prima generazione lanciate nel 2018 verranno abbandonate entro il 2020. La causa di tale dato è dovuto alle limitazioni delle tecnologie disponibili fino ad oggi, ovvero una mancanza di interazione intelligente da parte del Chatbot e la confusione sulla proprietà dei dati. Tra i limiti principali possiamo trovare ad esempio la difficoltà di creare un appeal globale, il limite imposto dalle normative per la protezione dei dati, la scarsa comprensione nelle conversazioni e la mancanza di dati di training.

## 2 Tecnologie Utilizzate

Per lo sviluppo del progetto, come prima cosa, è stato scaricato l'ecosistema *Anaconda*, un'applicazione desktop che consente di gestire facilmente applicazioni, pacchetti e ambienti integrati. Conda è un sistema open-source di gestione dei pacchetti e dell'ambiente che funziona su Windows, macOS e Linux. Installa, esegue e aggiorna rapidamente i pacchetti e le loro dipendenze. È stato creato per i programmi Python, ma è in grado di pacchettizzare e distribuire software per qualsiasi linguaggio. Tutto ciò ci ha permesso di creare un primo chatbot di base da cui siamo partiti per sviluppare la versione finale più sofisticata. Come già anticipato, il linguaggio di programmazione utilizzato è stato *Python* con alcune librerie come *Rasa\_sdk*, *csv*, *datetime* e così via. Il bot è stato sviluppato attraverso il framework *RASA*, esso infatti è uno dei tool open source maggiormente utilizzati, anche da aziende come T-Mobile, Adobe etc. Il servizio di messaggistica con cui è stato integrato il bot è *Telegram*. L'intero progetto è stato programmato tramite *VSCODE* e sono stati utilizzati alcuni tool relativi all'*yml validator*.

### 3 Framework Rasa

Nella figura 1 è possibile osservare la struttura gerarchica della cartella di progetto il quale contiene diversi file di configurazione, fondamentali per il framework Rasa. Oltre alle cartelle che vedremo in seguito nella relazione, la cartella *models* come suggerisce il nome contiene i modelli allenati tramite il comando *rasa train*, mentre in *tests* troviamo il file relativo alle test stories che ci permette, tramite il comando *rasa test*, di verificare la bontà del nostro modello. Inoltre, all'interno della directory principale si trovano alcuni file *".yaml"* tra cui *config* e *credentials* che ci permettono di definire alcuni parametri di configurazione e integrare il bot con piattaforme di messaggistica come Facebook, Telegram, etc. Lo scheletro del bot è rappresentato dal file *domain.yaml*.

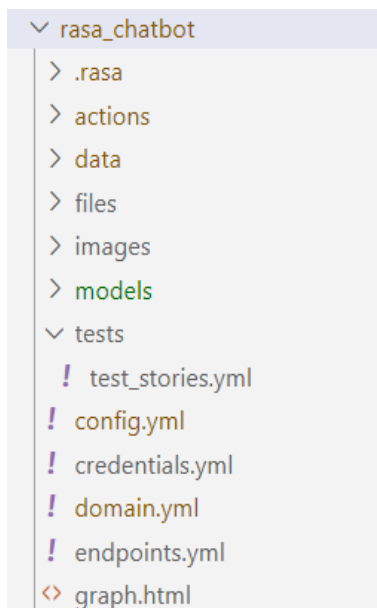


Figura 1: Framework Rasa

In dettaglio, nella figura 2 è possibile vedere come viene organizzata la cartella *actions*. Essa contiene il file *actions.py* il quale permette di svolgere alcune azioni personalizzate tramite il linguaggio Python, ovvero consente di definire azioni non banali in risposta agli intents dell'utente in input.

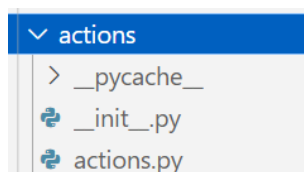


Figura 2: Framework Rasa "Actions"

Infine, come si vede dalla figura 3, nella cartella *data* sono presenti i file essenziali che rappresentano il cuore del nostro Chatbot. I tre file principali sono:

- *nlu.yaml*: definisce tutti gli intents, rappresenta l'intento dell'utente che interagisce con il bot.
- *rules.yaml*: descrivono brevi pezzi di conversazioni che dovrebbero seguire sempre lo stesso pattern.

- *stories.yml*: vengono definiti degli esempi di conversazione tra l'utente e l'assistente virtuale.

Questi file ci permettono di sviluppare le regole e definire delle liste di frasi attraverso il quale è possibile comunicare con il Chatbot.

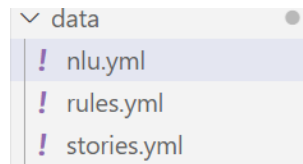


Figura 3: Framework Rasa "Data"

### 3.1 NLU

Come già anticipato in precedenza nella presentazione del framework, all'interno della directory *Data*, abbiamo il primo file principale, ovvero il file *nlu.yml*, dove al suo interno è possibile specificare i vari **intent** che verranno utilizzati nel Chatbot. La struttura prevede che venga definito l'intent di interesse e una serie di esempi di possibili frasi che l'utente potrebbe digitare in input. Questo permette al Chatbot per apprendere e comprendere al meglio l'intento che l'utente avrà quando comunicherà con lui.

```
- intent: faq_have_restaurant
  examples: |
    - does the hotel have a [restaurant](location)?
    - do you have a [restaurant](location)?
    - is there a [restaurant](location)?
    - is there a [restaurant](location) in the hotel?
    - tell me about the [restaurant](location)
    - can you tell me about any [restaurant](location) in the hotel?
    - are there [restaurants](location) in the hotel?
    - number of [restaurants](location)?
    - can i find [restaurants](location) here?
    - i heard there is a [restaurant](location) also
    - inform me about the [restaurant](location)
    - where is [restaurant](location)?
```

Figura 4: Esempio di intent NLU

Per poter comunicare con il chatbot è probabile che sia necessario inserire anche qualche informazione aggiuntiva che potrebbe essere ripetitiva. Ecco perchè, al fine di evitare di dover ripetere le medesime frasi di esempio vengono definite delle entità. Gli esempi per le entità vengono specificati utilizzando delle **lookup table**. Questo ci permette, anche in futuro, di non dover modificare l'intent, ma solo i possibili valori delle entità contenuti all'interno della lookup table.

### 3.2 Slots

Le lookup table possono essere integrate con le slots che rappresentano la memoria del bot. Essi sono settati nel file *domain.yml*. Questa combinazione ci permette inoltre di processare i possibili input dell'utente tramite le varie actions definite nel file *action.py* con l'ausilio dei trackers.

```

slots:
  number:
    type: text
    influence_conversation: true
    mappings:
      - type: from_entity
        entity: number
        conditions:
          - active_loop: form_book_room
          - active_loop: form_edit_book_room

```

Figura 5: Esempio di slots

### 3.3 Rules

Nel file *rules.yml* è possibile specificare una serie di regole, per poter interagire con il Chatbot. A differenza delle stories, le rules vengono utilizzate per definire alcune piccole parti tipiche di una conversazione, come ad esempio il "buongiorno", il quale dovrebbero seguire sempre lo stesso percorso.

```

rules:

- rule: Say goodbye anytime the user says goodbye
  steps:
    - intent: goodbye
    - action: utter_goodbye

- rule: Say 'I am a bot' anytime the user challenges
  steps:
    - intent: bot_challenge
    - action: utter_iamabot

```

Figura 6: Esempio di rules

### 3.4 Stories

All'interno del file *story.yml* è possibile impostare quelle che sono delle vere e proprie storie di possibili comunicazioni fra utente e Chatbot. In particolare, il meccanismo implementato da RASA prevede di specificare le interazioni tramite una serie di intent che l'utente potrà richiedere e una serie di azioni con cui il Chatbot dovrà rispondere. Nella figura 7 viene riportato un esempio della struttura di una storia, in particolare quella di prenotare una stanza.

```
#BOOKING ROOM
- story: booking path
  steps:
  - intent: book_room
  - action: form_book_room
  - active_loop: form_book_room
  - slot_was_set:
    - requested_slot: null
  - active_loop: null
  - action: utter_ask_number
  - intent: get_number_rooms
  - action: action_set_number_rooms
  - action: utter_ask_days
  - intent: get_number_days
  - action: action_set_days
  - action: action_booking_room
  - intent: add_reservation
  - action: action_save_reservation
```

Figura 7: Storia per prenotare una stanza

### 3.5 Actions

In *action.py* è possibile definire delle custom actions, ovvero delle possibili azioni personalizzare che il bot può svolgere in modo del tutto automatico come ad esempio il salvataggio, la lettura, la modifica e la cancellazione di prenotazioni da un file CSV. Nella figura 8 viene riportata l'azione che permetti di visualizzare una prenotazione effettuata.

```

# action per visualizzare una prenotazione
class ActionSeeReservation(Action):

    def name(self) -> Text:
        return "action_see_reservation"

    def run(self, dispatcher : CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        reservation_id = str(tracker.get_slot('reservation_id'))

        if(reservation_id == "None"):
            dispatcher.utter_message(text=f'Error: specify the reservation ID!')
            return []

        df = pd.read_csv(reservations_filename)

        # get index of the row with specified order ID
        index = df.index
        condition = df['Reservation ID'] == reservation_id
        reservation_index = index[condition]
        if len(reservation_index) == 0:
            # reservation not found
            # send message to the user
            dispatcher.utter_message(response='utter_no_reservations')
        else:
            # get details
            number = df.loc[reservation_index[0], 'Number of rooms']
            room_type = df.loc[reservation_index[0], 'Room Type']
            days = df.loc[reservation_index[0], 'Days']

            dispatcher.utter_message(text=f'The reservation {reservation_id} have booked {number} {room_type} rooms for {days} days.')

        return []

```

Figura 8: Azione per visualizzare una prenotazione

### 3.6 Pipeline

In Rasa, i messaggi scritti dagli utenti al bot vengono processati da una sequenza di componenti. Tali componenti possono essere specificati attraverso la pipeline che si trovano nel file *config.yml*. Scegliere una pipeline ci permette di personalizzare il nostro modello, adattandolo al dataset. Si è fatta la scelta di utilizzare la pipeline di default a cui poi si sono aggiunti altri componenti come si può vedere in figura 9.



```

# action per visualizzare una prenotazione
class ActionSeeReservation(Action):

    def name(self) -> Text:
        return "action_see_reservation"

    def run(self, dispatcher : CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        reservation_id = str(tracker.get_slot('reservation_id'))

        if(reservation_id == "None"):
            dispatcher.utter_message(text=f'Error: specify the reservation ID!')
            return []

        df = pd.read_csv(reservations_filename)

        # get index of the row with specified order ID
        index = df.index
        condition = df['Reservation ID'] == reservation_id
        reservation_index = index[condition]
        if len(reservation_index) == 0:
            # reservation not found
            # send message to the user
            dispatcher.utter_message(response='utter_no_reservations')
        else:
            # get details
            number = df.loc[reservation_index[0], 'Number of rooms']
            room_type = df.loc[reservation_index[0], 'Room Type']
            days = df.loc[reservation_index[0], 'Days']

            dispatcher.utter_message(text=f'The reservation {reservation_id} have booked {number} {room_type} rooms for {days} days.')

        return []

```

Figura 9: Codice della pipeline

### 3.7 Integrazione con Telegram

Tramite il file *credentials.yml* è possibile attuare l'integrazione con i diversi social media. Nel nostro caso verrà realizzata un'interazione con Telegram. Per fare ciò sarà necessario avere a disposizione un account Telegram, tramite il quale attraverso il bot **BotFather** è possibile creare il nostro bot. Sarà sufficiente quindi specificare il nome del bot e il suo username, così da avere l'access token da impostare all'interno del file di configurazione. Per realizzare tale integrazione è necessario l'utilizzo di ngrok, il quale ha permesso di realizzare il *tunneling*, cioè esporre al mondo esterno un server interno alla rete, occupandosi quindi sia dei NAT che dei possibili firewall, garantendo così una massima sicurezza. Il comando che è stato utilizzato per l'operazione di tunneling è stato **ngrok http 5005** come si può vedere in figura 10.

```
C:\Users\Utente\Downloads>ngrok http 5005
```

Figura 10: Comando ngrok utilizzato per attivare il tunnel

Mentre nella figura 11 si può osservare il risultato del comando visto nella figura 10 che ci fornisce importanti informazioni per la comunicazione tra il bot e Telegram.

```
ngrok
Join us in the ngrok community @ https://ngrok.com/slack

Session Status      online
Session Expires     1 hour, 59 minutes
Terms of Service     https://ngrok.com/tos
Version             3.0.6
Region              Europe (eu)
Latency              32ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://cdb4-79-30-23-225.eu.ngrok.io -> http://localhost:5005

Connections
  ttl   opn   rt1   rt5   p50   p90
    0    0    0.00  0.00  0.00  0.00
```

Figura 11: Caratteristiche del tunneling

Nella figura 12 si può vedere come viene configurato in Python il Chatbot per integrarlo con Telegram.

```
telegram:
  access_token: 5522250265:AAFxBxAlFteS9uKoCIkQJbc5AHLXkZcbYqs
  verify: Hotel_Milan_Bot
  webhook_url: "https://0b2e-79-30-23-225.eu.ngrok.io/webhooks/telegram/webhook"
```

Figura 12: Configurazione integrazione con Telegram

## 4 Funzionalità aggiuntive

### 4.1 Test stories

Un'ulteriore implementazione è stata quella di inserire delle storie di test, così da valutare il Chatbot realizzato su storie specifiche. Le storie di test sono state realizzate nel file *test\_stories.yml* specificando l'input dell'utente e l'intent che si sta realizzando, specificando poi le azioni che ci aspettiamo il bot realizzi. Sulla base di queste storie di test è stato possibile valutare il bot a seguito dei vari training effettuati, senza dover ripetere manualmente le storie in modalità shell. Il comando che ha permesso di valutare le storie di test è *rasa test*. Nell'immagine 13 è possibile vedere una delle story di test che abbiamo implementato.

#### 4.1.1 Risultati

Una volta inserite le stories di test essenziali, viene lanciato il comando *rasa test*, in modo tale da validare il modello precedentemente allenato su tali storie e valutarne i risultati con appositi grafici e metriche. Nei file *failed\_test\_stories.yml* e *stories\_with\_warning.yml* viene indicato rispettivamente se ci sono errori e warning nelle storie di test, nel nostro caso entrambi i file sono vuoti perché i test sono andati tutti a buon fine. Di seguito vengono riportati invece alcuni dei grafici ottenuti dopo il test del modello. Nella figura 14 è riportata la Intent Confusion Matrix. Come è possibile osservare si ha che la quasi totalità dei valori si trovano sulla diagonale della matrice, vengono quindi classificati correttamente dal modello. Nella figura 15 viene riportato un istogramma che permette di visualizzare la confidenza per tutte le varie predizioni mostrando quelle corrette e quelle errate (rispettivamente, in verde e in rosso).

```
#BOOKING ROOM
- story: booking path
  steps:
  - user : |
    | i want to book a room
    | intent: book_room
  - action: form_book_room
  - active_loop: form_book_room
  - slot_was_set:
    | - requested_slot: room_type
  - slot_was_set:
    | - room_type: Simple
  - slot_was_set:
    | - requested_slot: number
  - slot_was_set:
    | - number: '1'
  - slot_was_set:
    | - requested_slot: days
  - slot_was_set:
    | - days: '1'
  - slot_was_set:
    | - requested_slot: null
  - active_loop: null
  - action: action_booking_room
  - action: action_save_reservation
```

Figura 13: Esempio di test story

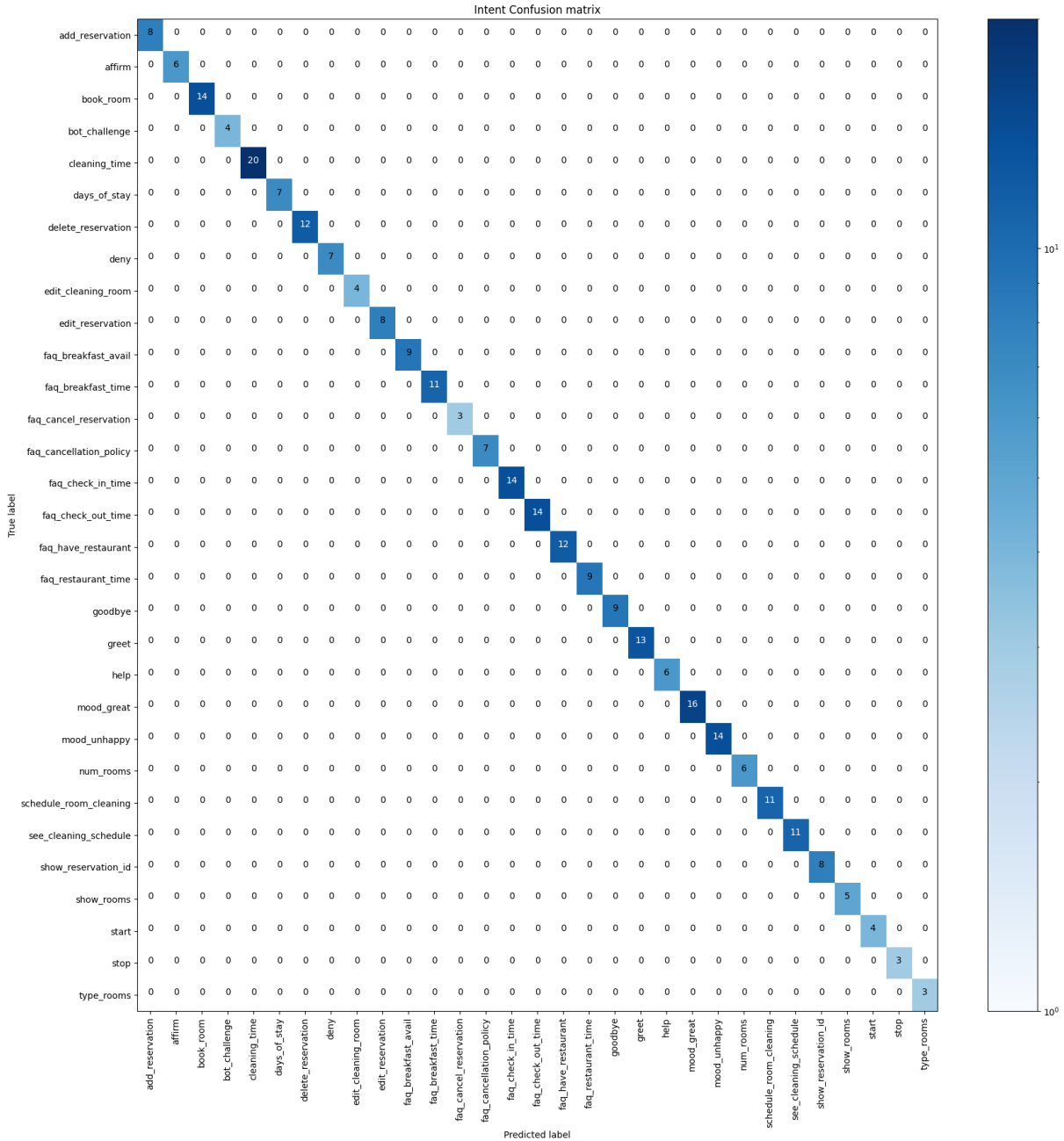


Figura 14: Intent Confusion Matrix

Come è ben visibile dall'immagine 15, la maggior parte delle predizioni corrette si trovano nell'area in alto. Questo indica che per quelle predizioni si hanno valori elevati di confidenza. Viceversa, un numero esiguo di predizioni con un basso valore di confidenza si trovano nella parte bassa. Nel complesso, si sono ottenuti dei buoni risultati per il modello testato.

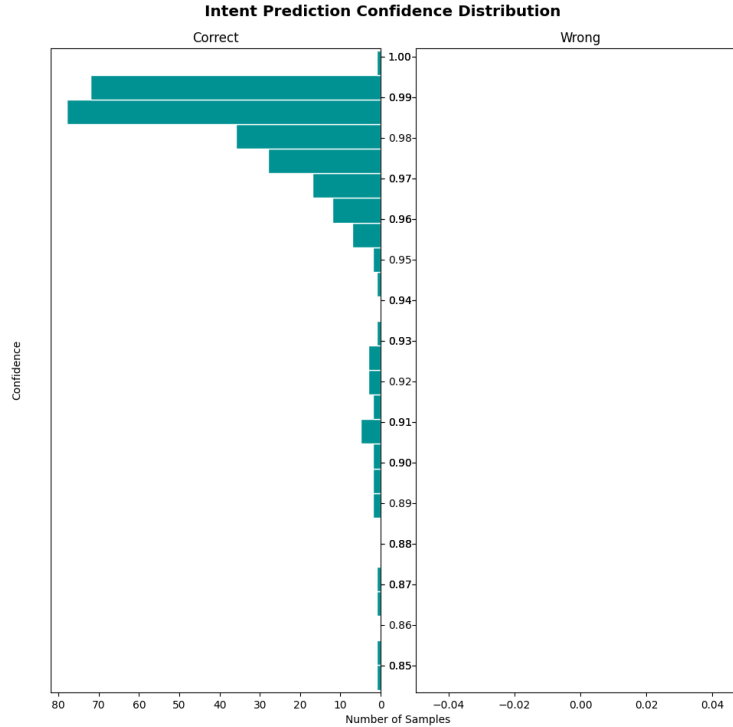


Figura 15: Intent Prediction Confidence Distribution

## 4.2 File CSV

Per rendere persistenti i dati delle prenotazioni anche tra diverse sessioni e conversazioni è stato scelto di utilizzare dei file *CSV* in cui memorizzare queste informazioni. In particolare, i dati persistenti sono:

- La prenotazione, memorizzata nel file *reservation.csv* con i campi: ID prenotazione, tipo di stanza, numero di stanze e numero dei giorni di soggiorno
- La pulizia della camera, memorizzata nel file *cleaning.csv* con i campi: ID pulizia, ID prenotazione, ora, minuti e am/pm.

## 5 Chatbot hotel

### 5.1 Help command

Per garantire all'utente una maggiore facilità di utilizzo, abbiamo pensato fosse utile riportare la funzionalità Help. Digitando questo comando, il bot risponderà con un messaggio che contiene una lista di frasi che mostrano tutte le possibili azioni che l'utente può effettuare inserendo in input gli esempi mostrati sottostanti.

```
1 You're welcome. Here is the list of things you can do by us:
2 #####
3 - Book a room -> (e.g.: i want to book a room)|
4 #####
5 - Show reservation -> (e.g.: show reservation with ID <<AA12345>>)
6 #####
7 - Show rooms -> (e.g.: show all rooms of the hotel)
8 #####
9 - Modify reservation -> (e.g.: edit reservation with ID <<AA12345>>)
```

```

10 #####
11 - Delete reservation -> (e.g.: delete reservation with ID <<AA12345>>)
12 #####
13 - Schedule cleaning room -> (e.g.: I want to have my room with ID <<AA54367>> cleaned)
14 #####
15 - Visualize cleaning schedule -> (e.g.: Cleaning schedule with ID <<CC54367>>)
16 #####
17 - Plan cleaning time -> (e.g.: Could you send someone in 2 hours?)
18 #####
19 - Edit cleaning schedule -> (e.g.: I want to edit the cleaning schedule with reservation ID
    <<AA54367>>)
20 #####
21 - FAQs check-in time -> (e.g.: what is the check-in time?)
22 #####
23 - FAQs check-out time -> (e.g.: what is the check-out time?)
24 #####
25 - FAQs cancel reservation -> (e.g.: how do i cancel my reservation?)
26 #####
27 - FAQs cancellation policy -> (e.g.: what is your cancellation policy?)
28 #####
29 - FAQs restaurant -> (e.g.: do you have a restaurant?)
30 #####
31 - FAQs breakfast -> (e.g.: does the hotel offer breakfast?)
32 #####
33 - FAQs breakfast time -> (e.g.: what is the time for breakfast?)
34 #####
35 - FAQs restaurant time -> (e.g.: what is the time of the restaurant?)
36 #####
37 - Quit the application using command <</stop>>
38 #####

```

## 5.2 Visualizzazione tipologie di stanze

L'utente può visualizzare la tipologia di stanze di cui l'hotel dispone, in particolare verranno visualizzate tre immagini che rappresentano rispettivamente una stanza semplice, una stanza deluxe e una stanza suite. Nella figura 16 viene raffigurata la risposta all'intent *show rooms* il quale mostra le stanze disponibili nell'hotel, in questo caso solamente la tipologia simple.

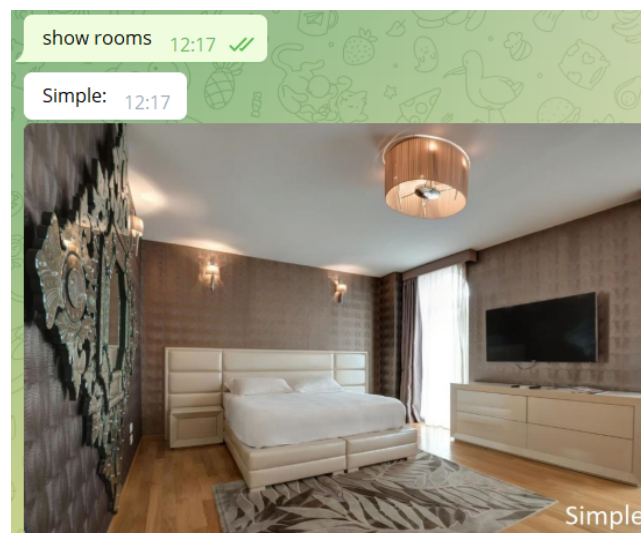
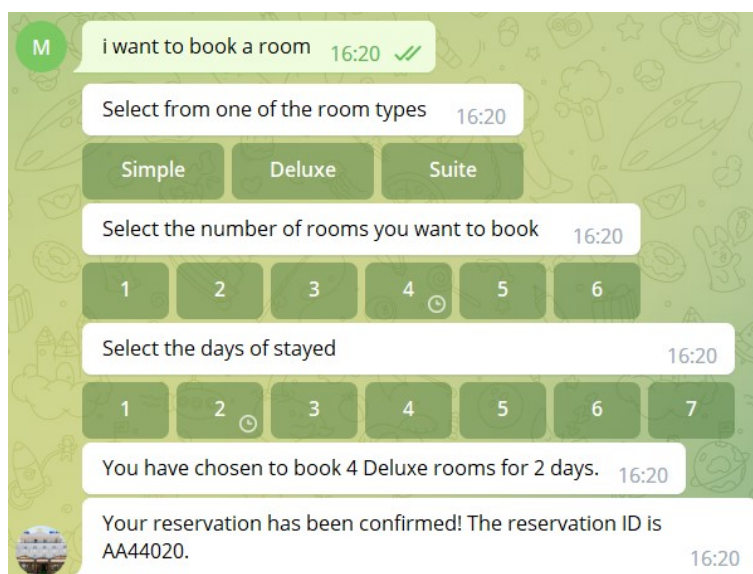


Figura 16: Stanza simple su Telegram

### 5.3 Prenotazione di una stanza

L'intent *book room* ha lo scopo di far prenotare all'utente una stanza dell'hotel. Quando l'utente digiterà una frase che attiverà tale intent, verrà attivata una form con tre bottoni (Simple, Deluxe e Suite) che consente di scegliere la tipologia di stanza. Successivamente il bot chiederà il numero di stanze che si desidera prenotare e il numero di giorni. Quando l'utente confermerà, la prenotazione verrà aggiunta su un file csv con un id generato casualmente che tiene traccia di tutte le prenotazioni effettuate.



M i want to book a room 16:20 ✓

Select from one of the room types 16:20

Simple Deluxe Suite

Select the number of rooms you want to book 16:20

1 2 3 4 5 6

Select the days of stayed 16:20

1 2 3 4 5 6 7

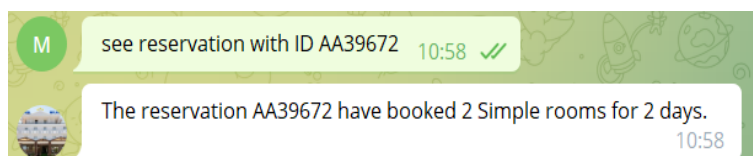
You have chosen to book 4 Deluxe rooms for 2 days. 16:20

Your reservation has been confirmed! The reservation ID is AA44020. 16:20

Figura 17: Prenotazione di una stanza

### 5.4 Visualizzare una prenotazione

L'utente può visualizzare una prenotazione effettuata in precedenza specificando l'id. Nella figura 18 viene mostrato quanto detto.



M see reservation with ID AA39672 10:58 ✓

The reservation AA39672 have booked 2 Simple rooms for 2 days. 10:58

Figura 18: Visualizzazione prenotazione

### 5.5 Modificare una prenotazione

L'utente può modificare una prenotazione effettuata in precedenza specificando l'id. In particolare, all'utente verrà presentato nuovamente un forms che chiede la tipologia di stanza, il numero di stanze e il numero di giorni di soggiorno. Il bot risponderà che la prenotazione con quel id specifico è stata aggiornata. Automaticamente tale prenotazione verrà aggiornata anche sul file *reservation.csv*.

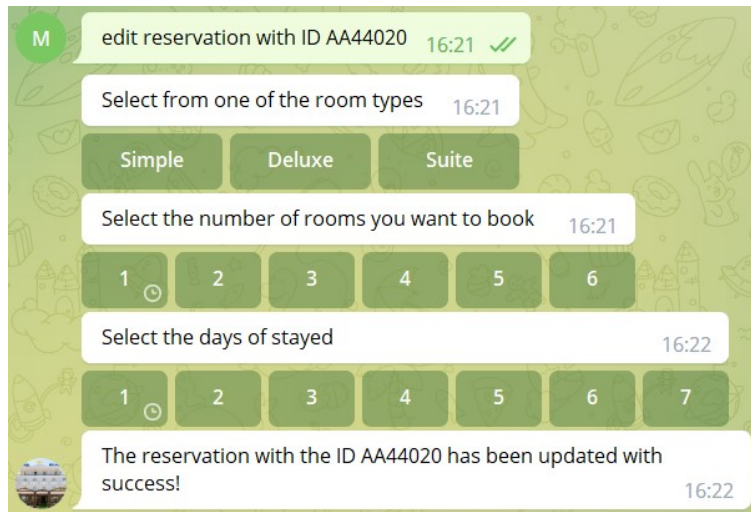


Figura 19: Modifica di una prenotazione

## 5.6 Eliminare una prenotazione

L'utente può eliminare una prenotazione effettuata in precedenza specificando l'id. Nella figura 20 viene mostrato quanto detto.

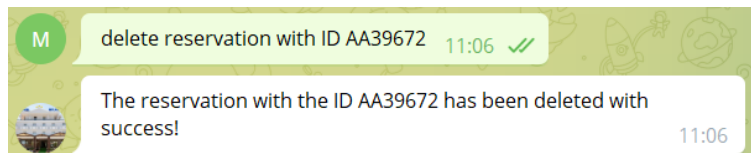


Figura 20: Cancellazione prenotazione

## 5.7 Organizzare una pulizia

L'utente dispone della possibilità di pulire la camera. Specificando l'id della prenotazione, dovrà solamente dire al bot che intende avere la camera pulita. Il bot risponderà con un messaggio il quale consente di scegliere l'orario in cui verrà organizzata la pulizia della stanza, specificando se eseguirla tra un range di ore oppure specificare l'ora esatta in cui essa verrà eseguita.

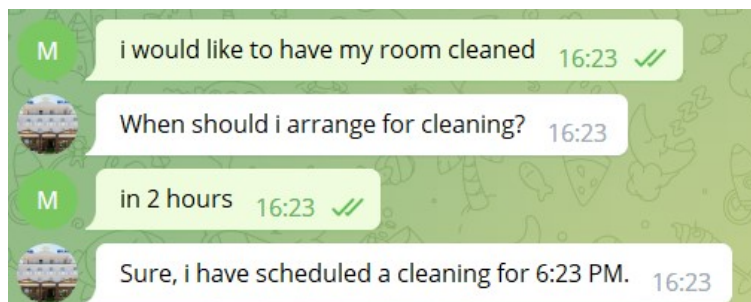


Figura 21: Prenotazione di una pulizia stanza



## 5.8 Visualizzare l'orario di pulizia di una stanza

Attraverso questo comando, l'utente è in grado di visualizzare la prenotazione della pulizia della stanza che ha prenotato. In particolare, specificando l'id della prenotazione (*AA12345*) o l'id della pulizia (*CC12345*), il bot mostra l'orario in cui essa è stata programmata.

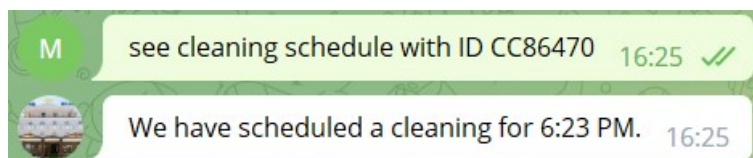


Figura 22: Visualizzare la pulizia della stanza

## 5.9 Modificare la pulizia di una stanza

L'utente può modificare l'orario in cui verrà eseguita una pulizia sulla sua stanza specificando l'id della prenotazione (*AA12345*) o l'id della pulizia (*CC12345*).

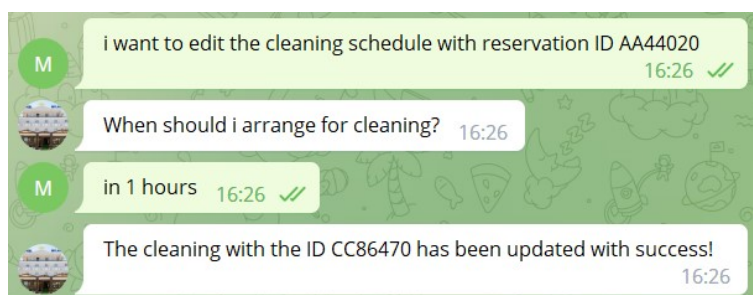


Figura 23: Modifica della pulizia di una stanza

## 5.10 FAQs

Digitando le opportune frasi, vengono mostrate alcune FAQ su, ad esempio, i possibili orari di check-in e check-out, la possibilità di effettuare la colazione in loco, oppure se l'hotel dispone di un ristorante e così via tutta una serie di altre informazioni.



Figura 24: Esempio di domande FAQ