



UNIVERSITÀ POLITECNICA DELLE MARCHE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

**Progettazione e implementazione di un
chatbot multimediale in tecnologia RASA per
la gestione dei dati dei pazienti in una
struttura sanitaria**

**Design and implementation of a multimedia chatbot in
RASA technology for the management of patient data
in a healthcare facility**

**Relatore: Chiar.mo
Prof. Domenico Ursino**

**Laureando:
Fabio Carosi**

Anno Accademico 2022-23

*If people do not believe that mathematics is simple,
it is only because they do not realize how complicated life is.*

John von Neumann

Sommario

Il presente lavoro di tesi mira a sviluppare un assistente digitale basato sull'Intelligenza Artificiale e sviluppato con il framework Rasa per facilitare l'accesso e la registrazione di dati strutturati in Cartella Clinica Elettronica agli operatori sanitari. Il chatbot, integrato in applicazioni mobile e web, trova applicazione anche nei contesti dove è necessaria una registrazione delle informazioni in tempo reale senza ricorrere all'uso di tastiera o mouse, come in sala operatoria o durante un'assistenza a bordo letto, grazie all'integrazione di moduli per la trascrizione vocale. Uno dei principali obiettivi consiste nel progettare un sistema efficiente ed a basso impatto formativo che possa essere adottato capillarmente nell'ambito del contesto sanitario.

Parole chiave: *chatbot, Rasa, NLP, artificial intelligence, healthcare, EHR, voice bot, Ionic*

Indice

Introduzione	1
1 Introduzione ai chatbot	3
1.1 Concetto di chatbot	3
1.1.1 Fondamenti	3
1.1.2 Storia dei chatbot	6
1.2 Tecnologie chiave	11
1.2.1 Modelli Linguistici	11
1.2.2 Machine Learning e Intelligenza Artificiale	12
1.2.3 Assistenti vocali	13
1.3 Applicazioni e settori di utilizzo	13
1.3.1 Esempi e benefici dell'implementazioni di chatbot	14
1.3.2 Stato dell'arte nella medicina	17
2 Framework Rasa	20
2.1 Introduzione a Rasa	20
2.2 Architettura	22
2.2.1 Rasa Open Source	22
2.2.2 Rasa Action Server	23
2.2.3 Altri componenti	24
2.3 Concetti chiave	24
2.3.1 Training Data	24
2.3.2 Domain	27
2.3.3 Actions	29

2.3.4	Channel Connector	32
2.4	Setup dell'ambiente di sviluppo	34
3	Analisi dei requisiti e progettazione	36
3.1	Raccolta dei requisiti	36
3.1.1	Descrizione dell'Attore	36
3.1.2	Requisiti Funzionali	37
3.1.3	Requisiti Non-Funzionali	39
3.2	Progettazione	40
3.2.1	Chatbot	41
3.2.2	Interfaccia Utente	47
4	Implementazione e manuale utente	52
4.1	Dati di training	52
4.1.1	NLU	53
4.1.2	Regole	57
4.1.3	Storie	59
4.2	Configurazioni	62
4.2.1	Domain	62
4.2.2	Pipeline	66
4.2.3	Endpoint e Credenziali	68
4.3	Azioni	69
4.3.1	Classi	69
4.3.2	Chiamate RESTful	76
4.3.3	Strumenti	77
4.4	Front-end	78
4.4.1	Realizzazione front-end web	78
4.4.2	Realizzazione front-end smartphone	82
4.5	Manuale utente	86
5	Discussione e analisi dei risultati	91
5.1	Riepilogo delle funzionalità	91

<i>INDICE</i>	v
5.2 Analisi dei risultati	93
5.2.1 Valutazioni delle performance del chatbot	94
5.2.2 Valutazioni delle performance della trascrizione vocale	103
5.3 Confronto con lo stato dell'arte	104
6 Conclusioni e sviluppi futuri	107
Riferimenti	109

Elenco delle figure

1.1	Interfaccia di ELIZA	7
2.1	Architettura di RASA	22
3.1	Diagramma dei casi d'uso	39
3.2	Diagramma di funzionamento dell'assistente	41
3.3	Diagrammi di flusso della conversazione	42
3.4	Diagramma di sequenza con messaggio di conferma	46
3.5	Diagramma di sequenza con messaggio di negazione	46
3.6	Diagramma di sequenza con response automatica	47
3.7	Diagramma di sequenza con validazione form	47
3.8	Mockup dell'interfaccia web	48
3.9	Mockup dell'interfaccia per smartphone	49
3.10	Diagramma di sequenza nel caso di invio di messaggi tramite tastiera	50
3.11	Diagramma di sequenza nel caso di invio di messaggi tramite voce	51
5.1	Esempi di conversazioni iniziali con il chatbot	92
5.2	Esempi di conversazioni iniziali con il chatbot	93
5.3	Esempi di funzionalità del chatbot	94
5.4	Esempi di funzionalità del chatbot	95
5.5	Confusion Matrix degli Intent	97
5.6	Istogramma di distribuzione della confidenza delle predizioni	102

Listati

4.1	Intent generici	53
4.2	Intent per l'attivazione delle form	54
4.3	Intent per l'attivazione delle azioni personalizzate	55
4.4	Esempi di regole per le risposte automatiche	57
4.5	Esempio di regola per attivazione e disattivazione delle form	58
4.6	Esempio di regola per l'attivazione di un'azione personalizzata	58
4.7	Esempio di storia con la richiesta dell'elenco delle Unità Operative	59
4.8	Esempio di storia interattiva	61
4.9	Elenco degli intent nel file di dominio	62
4.10	Form e slot nel file di dominio	63
4.11	Form e slot nel file di dominio	65
4.12	Pipeline di comprensione del testo	67
4.13	Endpoint server delle azioni	68
4.14	Connessione REST e SocketIO	68
4.15	Controllo dei metadati	70
4.16	Ottenimento dei valori relativi all'Unità Operativa selezionata	71
4.17	Parte di codice per l'inserimento di un nuovo parametro vitale	72
4.18	Azzeramento degli slot	73
4.19	Richiesta di conferma dell'azione da eseguire	74
4.20	Esecuzione delle chiamate REST associate agli intent	75
4.21	Metodo per l'ottenimento degli ultimi esami svolti dal paziente	76
4.22	Funzione per chiamate POST alle API aziendali	77
4.23	Codice JavaScript per l'interfaccia Web	79

4.24	Metodo che gestisce il riconoscimento del parlato	84
4.25	Metodo che gestisce l'invio dei messaggi	85
4.26	Metodo che gestisce il text-to-speech	86
5.1	Report degli intent	98
5.2	Alcuni errori di riconoscimento degli intent	100

Introduzione

Il presente lavoro di tesi descrive le attività effettuate, nell'ambito di un tirocinio esterno, presso l'azienda NBS s.r.l. di San Benedetto del Tronto, la quale opera sul territorio nazionale realizzando soluzioni informatiche in ambito sanitario.

In un contesto in cui l'Intelligenza Artificiale e i chatbot stanno acquisendo sempre maggiore rilevanza, la creazione di un assistente digitale specificamente progettato per i professionisti del settore sanitario emerge come un'opportunità di notevole interesse.

Nasce, così, la volontà di realizzare un chatbot a supporto degli operatori sanitari per facilitare la registrazione e la lettura di dati strutturati del paziente in Cartella Clinica Elettronica. Per rendere il chatbot maggiormente accessibile, si è deciso di integrare moduli per la trascrizione e la sintesi vocale in modo da poter impiegare il sistema anche senza utilizzare la tastiera.

Il progetto trova applicazione nei contesti dove è richiesto un rapido accesso ai dati del paziente o una registrazione delle informazioni in tempo reale, come in sala operatoria o durante l'assistenza a bordo letto. Uno dei principali obiettivi consiste nel progettare un sistema efficiente ed a basso impatto formativo in grado di essere capillarmente adottato nell'ambito del Sistema Sanitario Nazionale.

Per raggiungere questi obiettivi, il chatbot è stato realizzato tramite framework Rasa, quest'ultimo incorpora avanzate tecniche di elaborazione del linguaggio naturale per comprendere e interpretare le richieste degli utenti in modo contestuale.

L'assistente digitale viene integrato in un applicativo web e in un'applicazione mobile cross-platform realizzata tramite il framework Ionic. Entrambe le interfacce front-end implementano le funzionalità di speech-to-text delle richieste dell'utente e di text-to-speech dei messaggi del chatbot.

La tesi è composta da sei capitoli, i quali vengono brevemente descritti di seguito:

1. *Introduzione ai chatbot*: esplora il viaggio storico e tecnologico dei chatbot, dai pionieri, come ELIZA, a sviluppi recenti nell'IA, evidenziando tecnologie chiave e applicazioni in settori come la medicina.
2. *Framework Rasa*: approfondisce il framework Rasa, focalizzandosi sulle componenti chiave e offrendo una guida pratica per l'avvio di progetti Rasa.
3. *Analisi dei requisiti e progettazione*: analizza i requisiti e la progettazione del chatbot, utilizzando diagrammi di sequenza e discutendo l'implementazione di regole e modelli di Machine Learning per una conversazione personalizzata.
4. *Implementazione e manuale utente*: dettaglia l'implementazione del chatbot, coprendo aspetti tecnici come la gestione dei messaggi, l'integrazione della sintesi vocale e la presentazione di un manuale utente.
5. *Discussione e analisi dei risultati*: si concentra sulla discussione e nell'analisi dei risultati del chatbot, esplorando esempi di conversazioni, valutando le performance e confrontando alternative sul mercato, offrendo uno sguardo completo sullo stato attuale delle soluzioni.
6. *Conclusioni e sviluppi futuri*: riepiloga il lavoro svolto e propone futuri miglioramenti del sistema realizzato.

Capitolo 1

Introduzione ai chatbot

Nel primo capitolo di questa tesi verrà esplorato il mondo dei chatbot, partendo dalla loro definizione per poi concentrarsi sulla loro evoluzione e l'attuale stato dell'arte della tecnologia. Si analizzeranno le tecnologie alla base degli assistenti digitali e il ruolo dell'Intelligenza Artificiale (IA) nell'evoluzione dei chatbot moderni. Sarà anche esplorato come i chatbot trovano applicazione in diversi settori e contesti, evidenziando i benefici derivanti dalla loro implementazione. Inoltre, si introdurrà il concetto di assistenti vocali, effettuando un confronto con i chatbot più classici, analizzando i pregi e i difetti di questa soluzione anche all'interno del mondo della medicina.

1.1 Concetto di chatbot

1.1.1 Fondamenti

I chatbot, talvolta declinato al femminile, definendoli in maniera più generale possibile, sono software, cioè programmi per computer, in grado di simulare una conversazione con un utente. Il termine, originariamente *chatterbot*, è coniato per la prima volta nel 1994 da Michael Mauldin, ed unisce le parole inglesi *chat* (conversazione) e *robot*. Un chatbot permette di interagire con la tecnologia utilizzando una varietà di metodi di input come testo, voce, gesture e touch.

I chatbot sono noti con una grande varietà di nomi, come bot conversazionale, agente conversazionale, assistente virtuale, assistente digitale, tra gli altri. Questa varietà di termini riflette la loro crescente presenza nella vita quotidiana e la diversità delle loro

applicazioni. Ad esempio, gli assistenti vocali, come Siri di Apple, Google Assistant o Amazon Alexa, sono tipi specifici di chatbot che si concentrano sulla risposta a domande e sull'esecuzione di comandi vocali, mentre i bot conversazionali sui siti web possono fornire supporto all'utente o assistenza nell'acquisto di prodotti e servizi.

I bot possono essere implementati con diversi gradi di funzionalità o di "intelligenza". I più semplici sono delle soluzioni front-end in grado di rispondere a domande frequenti degli utenti. I più avanzati, alimentati da algoritmi di Intelligenza Artificiale (IA) e apprendimento automatico, sono in grado di comprendere il contesto della conversazione, apprendere dai dati e migliorare continuamente le loro risposte. Questo livello di sofisticazione li rende adatti per applicazioni più complesse, come il supporto virtuale, l'automazione dei processi aziendali e molto altro. L'implementazione e lo sviluppo dei chatbot diventano parte integrante della trasformazione digitale in molteplici settori.

I chatbot offrono un modo più semplice e immediato di accedere alle informazioni da parte di un utente, costituendo un'alternativa alla navigazione in menù complessi in cui non risulta facile individuare le informazioni necessarie.

Funzionamento

La base del funzionamento di ogni chatbot è l'interazione uomo-macchina. Possiamo dividere i chatbot in due categorie distinguendo in che modo avviene la conversazione con il bot; si hanno, quindi, assistenti vocali e chatbot testuali.

La differenza principale tra le due tipologie è il metodo di input; in caso di input vocale avremo bisogno di tecnologia ASR (*Automatic Speech Recognition*), anche chiamata tecnologia STT (*Speech To Text*), che si occupa proprio di trasformare i dati vocali in testo; se il chatbot è soltanto testuale questo passaggio non risulta necessario. In modo analogo, una volta determinata la risposta da dare all'utente, l'assistente vocale avrà un modulo TTS (*Text To Speech*) per ottenere l'output sotto forma di voce.

Si analizza, ora, in che modo i chatbot riescano a comprendere il linguaggio umano, l'argomento è ripreso e ampliato nei capitoli successivi. La comprensione degli esseri umani può essere molto complicata per una macchina; il nostro modo di comunicare è spesso sfaccettato e sfumato e difficile da ricreare artificialmente: per questo si utilizzano diversi principi di linguaggio naturale.

Il *Natural Language Processing (NLP)* assume un ruolo fondamentale nella comprensione e nell'elaborazione del linguaggio naturale. Questo processo inizia con la suddivisione dell'input dell'utente in frasi e parole, creando una struttura di base per l'analisi. Tuttavia, esso va oltre: attraverso una serie di sofisticate tecniche, l'NLP standardizza il testo, applicando operazioni come la conversione in minuscolo e la correzione degli errori di ortografia. Questa fase critica determina se una parola è un aggettivo o un verbo, ma non si ferma qui. Vengono considerati, anche, altri aspetti, come il sentimento associato alle parole, che aggiunge una dimensione emotiva alla comprensione del testo.

Il *Natural Language Understanding (NLU)* aiuta il chatbot a interpretare ciò che l'utente ha detto, sfruttando sia le costruzioni linguistiche generali che quelle specifiche del dominio, come lessici, sinonimi e temi rilevanti. Queste informazioni sono fondamentali per creare flussi di dialogo significativi. Gli algoritmi e le regole entrano in gioco per guidare il chatbot nella formulazione di risposte adeguate e coerenti con il contesto.

Per fornire un'esperienza utente ancora più ricca, il *Natural Language Generation (NLG)* è cruciale. Questa fase permette al chatbot di andare oltre le risposte predefinite e generare linguaggio naturale in modo dinamico. Questa capacità è essenziale quando il chatbot deve attingere ad archivi di dati, compresi i sistemi back-end e i database di terze parti. Grazie all'NLG, il chatbot può accedere a queste informazioni e creare risposte personalizzate basate su dati reali, garantendo un'esperienza utente autentica e informativa.

La *Conversational AI Technology* solleva il Natural Language Processing e il Natural Language Understanding a un livello superiore. Questa tecnologia permette di costruire sistemi di dialogo avanzati che non solo comprendono il linguaggio, ma memorizzano anche informazioni cruciali, come preferenze personali e contesto conversazionale. Ciò consente loro di offrire interfacce di linguaggio naturale realistiche e coinvolgenti, creando conversazioni fluide e personalizzate che si adattano alle esigenze dell'utente. In questo modo, la Conversational AI rappresenta un passo avanti significativo nella creazione di interazioni uomo-macchina sempre più avanzate.

1.1.2 Storia dei chatbot

Nonostante si possa credere che i chatbot siano nati solo recentemente, in realtà la loro storia risale a decenni fa. Di seguito, si riportano i momenti salienti di questa tecnologia, partendo dalle prime ipotesi fino ad arrivare all'esplosione delle funzionalità avvenuta dall'avvento di Internet in poi.

La prima persona che si chiese se le macchine potessero pensare è stato il celebre Alan Turing, che nell'altrettanto celebre articolo del 1950 "Computing Machinery and Intelligence"¹, propose un test in cui un interrogatore doveva determinare quale giocatore fosse un umano e quale una macchina attraverso una serie di domande scritte. Nonostante alcuni difetti e alcune critiche mosse nel corso degli anni, tra tutte la più nota è l'esperimento mentale della "Stanza cinese" presentato da John Searle nel 1980², il test di Turing viene eseguito regolarmente ancora oggi per avere un'idea del grado di intelligenza del bot.

Nel 1966 Joseph Weizenbaum presenta quello che si può considerare come il primo chatterbot, ELIZA³ (figura 1.1). *ELIZA* nasce come parodia di un terapeuta rogersiano, il suo funzionamento è piuttosto basilare: le risposte sono generate dalla domanda dell'utente grazie a specifiche regole di decomposizione e assemblaggio. *ELIZA* si occupa di identificare le parole chiave, comprendere un contesto minimo e scegliere le trasformazioni adeguate da applicare; questo tipo di approccio rappresenta la base del Natural Language Processing. *ELIZA* viene anche associata all'*Effetto ELIZA*, ovvero la tendenza di proiettare caratteristiche umane (come intelligenza, esperienza, comprensione della semantica o empatia) a programmi informatici che non hanno tali caratteristiche.

Lo psichiatra Kenneth Colby, nel 1972, implementa *PARRY*⁴ nell'università di Stanford. Il chatbot presenta un modello grezzo di comportamento di una persona con schizofrenia paranoica e possiede una strategia di conversazione, risultando dunque più convincente di *ELIZA*. *PARRY* ed *ELIZA* hanno interagito diverse volte; la più nota delle conversazioni tra i due bot è avvenuta nel corso della International Conference on Computer Communications tenutasi nell'ottobre del 1972, *PARRY* ed *ELIZA* si rispondevano a vicenda, collegati tramite ARPANET.

Nel 1984 viene pubblicato il libro "*The Policeman's Beard Is Half Constructed*"⁵, il primo scritto da una "Intelligenza Artificiale". L'autore è infatti *Racter*, un bot svilup-

```

=====
EEEEEEEE L      IIIIIII ZZZZZZZ      AAA
E         L      I        Z          A  A
E         L      I        Z          A  A
EEEEEE   L      I        Z          A  A
E         L      I        Z          AAAAAA
E         L      I        Z          A  A
EEEEEEEE LLLLLLLL IIIIIII ZZZZZZZ      A  A
=====
ELIZA > (PLEASE TYPE IN ALL CAPS) WHAT'S YOUR NAME DEAR ?
AMIT
ELIZA > HI, I'M ELIZA. WHAT DO YOU WANT TO TALK ABOUT ?
AMIT > CAN I TALK ABOUT MY PROBLEM ?
ELIZA > SURE... !
AMIT > DUE TO ON GOING EXAMS IN COLLEGE, I HAVE A LOT OF STRESS.
ELIZA > PLEASE RELAX A LITTLE. SLEEP WELL.
AMIT > THANKS FOR YOUR ADVICE.
ELIZA > NO MENTION.
AMIT > BYE
ELIZA > BYE AND KEEP IN TOUCH...
=====

```

Figura 1.1: Interfaccia di ELIZA

pato da William Chamberlain e Thomas Etter, che hanno rivelato la natura dell'autore del libro. Una versione interattiva di Racter, diminutivo di *raconteur* (tr. narratore), è stata sviluppata da Inrac Corporation e rilasciata da Mindscape; questa versione rilasciata al pubblico era più simile ad un chatbot rispetto alla versione originale, mai resa pubblica, utilizzata da Chamberlain e Etter per la scrittura del libro.

Nel 1990, Hugh Loebner lancia il Premio Loebner, una competizione annuale che prende il formato di un test di Turing standard con i giudici che premiano il programma per computer più simile all'uomo. Sono previsti due premi speciali, un primo premio "d'argento", per il primo bot in grado di non essere distinto da un essere umano tramite una conversazione testuale; un secondo premio "d'oro" per il primo bot in grado di non essere distinto includendo anche decifrazione e comprensione di input visivi e uditivi, oltre che testuali. La competizione annuale terminerà quando verrà assegnato il premio principale; ad oggi, non è stato assegnato nessuno dei due premi, l'ultima edizione svolta è quella del 2019.

Jabberwacky è un chatbot creato dal programmatore britannico Rollo Carpenter.

Nasce nel 1981 come programma hard-coded; nel 1988 diventa un progetto di Intelligenza Artificiale chiamato 'Thoughts' e nel 1997 è lanciato su Internet come 'Jabberwacky'. Fu uno dei primi tentativi di creare Intelligenza Artificiale attraverso l'interazione umana. Il chatbot è stato progettato per "simulare la naturale chat umana in un modo interessante, divertente e umoristico". Rollo Carpenter ha vinto il Premio Loebner nel 2005 e 2006.

Dr. Sbaitso è un chatbot psicologo computerizzato con una voce digitale progettato per parlare con l'utente. Questo chatbot, dotato di una voce digitale, fu sviluppato da Creative Labs nel 1991 e rappresentava un primo esempio di sintesi vocale basata sull'Intelligenza Artificiale. Il suo obiettivo principale era quello di condurre conversazioni con l'utente, dimostrando le capacità allora impressionanti delle voci digitalizzate delle schede audio. Dr. Sbaitso ha contribuito a mettere in luce le potenzialità dell'IA nell'ambito delle interazioni vocali e ha servito da ispirazione per lo sviluppo successivo di chatbot vocali più avanzati

A.L.I.C.E. (Artificial Linguistic Internet Computer Entity), noto anche come Alicebot, o semplicemente Alice, è un chatterbot di elaborazione del linguaggio naturale che ha fatto il suo debutto nel 1995. Questo chatbot ha vinto il Loebner Award per ben tre volte, dimostrando la sua abilità nel simulare conversazioni umane. Alice fu inizialmente ispirato dal programma ELIZA e rappresenta un passo avanti significativo nella creazione di chatbot in grado di comprendere e rispondere al linguaggio umano in modo coerente.

Il chatbot *Smarterchild* è stato sviluppato presso ActiveBuddy Inc. da Robert Hoffer, Timothy Kay e Peter Levitan nel 2001. Ha rappresentato un'importante evoluzione nel campo delle conversazioni personalizzate. Smarterchild era accessibile su reti popolari, come AOL Instant Messenger e MSN Messaging. Questo chatbot offriva conversazioni simpatiche e personalizzate, anticipando in modo sorprendente le future assistenti virtuali, come Siri di Apple e S Voice di Samsung.

Nel 2005, Steve Worswick presso AIML Technology crea *Mitsuku*, un chatbot che ha guadagnato notorietà grazie alle sue vittorie nel premio Loebner negli anni 2013, 2016, 2017, 2018 e 2019, record assoluto di vittorie. Mitsuku, diventata poi Kuki AI, si presenta come un chatbot femminile adolescente proveniente da Leeds, in Inghilterra. Ciò che lo distingue è la sua capacità di ragionare su argomenti specifici, giocare e persino eseguire trucchi di magia.

Watson, sviluppato da IBM nel 2006 e denominato in onore del primo CEO dell'azienda, Thomas J. Watson, ha ottenuto notorietà vincendo il programma televisivo americano "Jeopardy!" nel 2011 contro due campioni umani. Tuttavia, le sue capacità non si sono limitate al gioco. Watson ha adottato il Natural Language Processing (NLP) e il Machine Learning per estrarre conoscenze da grandi quantità di dati. Ciò ha rappresentato un passo avanti nella capacità delle macchine di comprendere e utilizzare il linguaggio naturale in modo significativo.

Nel secondo decennio del 2000, nascono gli assistenti conosciuti anche oggi dalla massa. Si inizia nel 2010 con *Siri*. Il progetto nasce come uno spin-off del SRI International Artificial Center e come ramo del progetto CALO della statunitense DARPA (Defense Advanced Research Projects Agency); il motore di riconoscimento vocale è, invece, fornito da Nuance³⁰. Siri è rilasciata come app per iOS nel febbraio 2010 ed era previsto lo sbarco anche su Android e Blackberry prima che venisse acquisita da Apple nell'aprile dello stesso anno. Siri è diventato un'icona quando Apple ha integrato l'assistente vocale nell'iPhone 4S nel 2011. Questa mossa ha reso le applicazioni vocali parte integrante del mercato consumer mainstream, dimostrando la crescente accettazione dell'interazione vocale come mezzo principale di comunicazione uomo-macchina.

Nel 2013 è stato il turno di Google che ha rilasciato su Android e iOS il nuovo software di assistenza personale intelligente *Google Now*; questo è stato progettato come estensione di Google Search e ha aperto nuove prospettive nell'uso del linguaggio naturale per rispondere a domande, fare raccomandazioni ed eseguire azioni. La sua interfaccia utente intuitiva ha reso più accessibili le funzioni di ricerca di Google. Nel 2016 Google Now è stato sostituito dal nuovo *Google Assistant*.

Alexa, lanciato da Amazon nel 2015, ha rivoluzionato il mercato degli assistenti vocali mobili, soprattutto nel settore delle case intelligenti. Il suo successo ha dimostrato l'efficacia di dare comandi vocali per controllare dispositivi domestici intelligenti e ha contribuito a creare un mercato in crescita per gli altoparlanti intelligenti.

Anche Microsoft ha sviluppato il proprio assistente personale intelligente, *Cortana*, rilasciato nel 2014 per sistemi operativi Windows Phone, Windows 10 e Windows 11. Come le controparti, Cortana riconosce i comandi vocali naturali e sfrutta il motore di ricerca Bing per rispondere alle domande degli utenti e impostare promemoria.

I *Facebook Chatbots* sono presentati nel 2016 con il lancio sulla piattaforma di messaggistica di Facebook. La piattaforma è diventata leader per i chatbot, ospitando oltre 300.000 bot attivi sulla piattaforma Messenger di Facebook già nel 2018. Ciò ha evidenziato il ruolo cruciale dei chatbot nelle interazioni sociali online.

Uno dei casi controverso di chatbot è *Tay*, creato da Microsoft nel 2016; esso ha suscitato polemiche per il suo comportamento controverso. Nato con l'obiettivo di imitare i discorsi di una ragazza adolescente americana, fu chiuso solo 16 ore dopo il lancio a causa di tweet offensivi e razzisti, sollevando importanti questioni etiche nell'ambito dell'IA.

Negli ultimi anni gli sviluppi più entusiasmanti si sono concentrati maggiormente nel campo dell'Intelligenza Artificiale. Nel 2017 viene pubblicato un articolo da parte dei ricercatori di Google in cui si presentano i cosiddetti *Transformer*⁶. I Trasformer rappresentano un avanzamento significativo nell'ambito delle reti neurali artificiali e hanno rivoluzionato il campo dell'elaborazione del linguaggio naturale (NLP). Essi presentano un'architettura di rete neurale altamente parallela basata su meccanismi di attenzione. A differenza delle reti neurali ricorrenti (RNN) e delle reti neurali convoluzionali (CNN), i Trasformer utilizzano l'attenzione per analizzare e processare sequenze di dati in parallelo, il che li rende estremamente efficienti per l'NLP.

BERT, acronimo di "Bidirectional Encoder Representations from Transformers," è un notevole esempio di come i Trasformer abbiano rivoluzionato l'NLP. BERT è stato introdotto da Google nel 2018⁷ ed è una rete neurale preaddestrata che utilizza un modello di linguaggio bidirezionale, in cui vengono considerati sia il contesto a sinistra che quello a destra di una parola durante la previsione del significato delle frasi. Questa capacità di comprensione del contesto bidirezionale ha permesso a BERT di raggiungere risultati eccezionali in una serie di attività NLP, migliorando notevolmente la comprensione del linguaggio naturale da parte delle macchine.

I Trasformer sono strettamente collegati anche a *GPT*, ovvero "Generative Pre-trained Transformer"⁸. GPT è una famiglia di modelli di lingua creati da OpenAI, basati sulla stessa architettura di rete neurale dei Trasformer. GPT utilizza modelli di linguaggio preaddestrati per generare testo in modo coerente e contestualmente rilevante. GPT-3, ad esempio, è uno dei modelli più avanzati ed è noto per la sua capacità di

generare testo di alta qualità in risposta a input dati. La differenza chiave tra BERT e GPT è che BERT è utilizzato per comprensione del linguaggio e classificazione, mentre GPT è principalmente orientato alla generazione di testo.

1.2 Tecnologie chiave

I chatbot si possono dividere in due categorie in base al modello con il quale sono stati progettati: modello linguistico o modello di machine learning. Di seguito, viene descritto il funzionamento dei due modelli, per poi aggiungere un'analisi del funzionamento degli assistenti che utilizzano input e output vocali.

1.2.1 Modelli Linguistici

I chatbot linguistici, spesso noti come chatbot basati su regole, offrono un elevato grado di controllo e flessibilità, elementi che spesso mancano nei chatbot basati su machine learning.

Una delle chiavi del successo dei chatbot basati su regole è la capacità di anticipare e progettare risposte corrette per le domande degli utenti, consentendo inoltre di implementare test automatici per garantire la qualità e la coerenza del sistema. Questi chatbot si basano sulla logica `if/then` per creare flussi di conversazione, offrendo la possibilità di definire regole specifiche per gestire le parole, il loro ordine, sinonimi e variazioni comuni nelle formulazioni delle domande. Questo approccio garantisce che domande con significati simili ricevano risposte coerenti e adeguate.

Un vantaggio significativo dei chatbot basati su regole è la loro adattabilità, poiché, se si verifica un errore di comprensione, un essere umano può facilmente raffinare le condizioni delle regole per migliorare la conversazione.

Tuttavia, è importante notare che i chatbot basati esclusivamente su modelli linguistici possono presentare alcune limitazioni. Questo approccio può risultare rigido e richiedere tempo per essere sviluppato, poiché richiede una pianificazione dettagliata e laboriosa nella definizione delle regole. Sebbene basato sul Natural Language Processing (NLP), le interazioni con questi chatbot tendono a essere specifiche e altamente strut-

turate. Di conseguenza, i chatbot basati su regole possono assomigliare a delle FAQ interattive e le loro capacità sono di base.

Questi sono i tipi di chatbot più comuni con cui è più comune entrare in contatto; spesso si trovano su chat dal vivo, piattaforme di e-commerce o servizi di messaggistica, come Facebook Messenger o Telegram.

1.2.2 Machine Learning e Intelligenza Artificiale

I chatbot potenziati dall'Intelligenza Artificiale rappresentano una categoria più avanzata rispetto ai chatbot basati su regole, caratterizzandosi per la loro natura conversazionale, guidata dai dati e predittiva.

Questi chatbot si distinguono per la loro complessità e sofisticazione. Sono in grado di offrire un'interazione più spinta e personalizzata rispetto ai loro omologhi orientati a compiti specifici. Nel corso del tempo, accumulando dati, diventano sempre più consapevoli del contesto in cui operano. Grazie alla comprensione del linguaggio naturale e all'applicazione dell'intelligenza predittiva, sono in grado di personalizzare l'esperienza dell'utente in modo significativo.

I sistemi di conversazione basati sul machine learning possono dimostrarsi straordinariamente efficaci quando affrontano problemi ben adattati alle loro capacità. La loro forza risiede nella loro capacità di apprendere da schemi e esperienze passate; tuttavia, questa caratteristica richiede una quantità considerevole di dati di addestramento e l'intervento di esperti altamente qualificati.

Tuttavia, è importante sottolineare che i chatbot basati sul machine learning presentano alcune sfide. Spesso operano come "scatole nere", ciò significa che, se qualcosa va storto con il modello, può risultare difficile intervenire per correggere il comportamento indesiderato, figuriamoci ottimizzarlo o migliorarlo. Le risorse richieste per addestrare e mantenere questi sistemi, unite alla limitata gamma di scenari in cui gli algoritmi statistici eccellono davvero, possono rendere i chatbot basati esclusivamente sul machine learning una scelta poco pratica per molte imprese.

1.2.3 Assistenti vocali

Gli assistenti vocali, come Siri di Apple o Google Assistant, utilizzano una serie di tecnologie avanzate per comprendere e rispondere alle richieste vocali degli utenti. Uno degli elementi chiave dietro il funzionamento di questi assistenti è il modulo di Riconoscimento Automatico del Parlato (ASR), noto anche come modulo di trascrizione vocale.

Il modulo ASR è responsabile della conversione del parlato dell'utente in testo comprensibile. Questo processo inizia con la registrazione dell'audio dall'utente e il suo invio al modulo ASR. Qui, un modello di machine learning addestrato su enormi quantità di dati vocali si occupa di analizzare l'audio e identificare le parole pronunciate. Questo modello è stato precedentemente addestrato su un vasto corpus di dati vocali, che include variazioni di pronuncia, accenti, e contesti linguistici diversi per garantire una comprensione accurata. L'addestramento di questo modello richiede un notevole sforzo computazionale e dati di addestramento estensivi.

Una volta che il modulo ASR ha trascritto con successo il parlato dell'utente in forma di testo, il risultato viene inviato ad altri moduli all'interno dell'assistente vocale per l'elaborazione successiva. Questi moduli includono il Natural Language Understanding (NLU), per comprendere il significato della richiesta, e il Natural Language Generation (NLG), per generare una risposta vocale appropriata.

In sintesi, gli assistenti vocali sfruttano il modulo ASR per convertire il parlato dell'utente in testo, consentendo al sistema di comprendere la richiesta dell'utente. Questo processo è reso possibile grazie all'addestramento di modelli di machine learning su dati vocali vasti e diversificati, che garantiscono una comprensione accurata del parlato degli utenti.

1.3 Applicazioni e settori di utilizzo

Di seguito si analizzano i principali esempi di applicazioni⁹ di assistenti digitali in diversi settori e contesti, effettuando un'analisi più approfondita sul mondo della sanità.

1.3.1 Esempi e benefici dell'implementazioni di chatbot

Educazione

L'aumento della domanda di istruzione ha portato a una crescente competizione tra gli istituti di istruzione. Uno dei problemi critici che contribuisce allo scarso apprendimento e all'alto tasso di abbandono è la riduzione dell'assistenza da parte degli insegnanti a causa dell'aumento del numero di studenti. In questo contesto, l'impiego dei chatbot si è dimostrato un valido alleato per migliorare le pratiche di e-learning¹⁰.

I chatbot per il supporto all'apprendimento hanno la capacità di conservare le informazioni e ripetere le lezioni precedenti per gli studenti che le hanno perse. Inoltre, raccolgono dati durante il corso, contribuendo, così, a migliorare il processo di insegnamento e apprendimento. Gli studenti traggono vantaggio da questa tecnologia in quanto i chatbot sono in grado di rispondere alle loro domande sul materiale didattico. Inoltre, i chatbot possono assistere gli studenti nelle questioni amministrative, come l'iscrizione ai corsi, la pianificazione degli esami, la consultazione dei voti e altri aspetti legati al loro percorso di studio, alleggerendo notevolmente il carico di lavoro dei dipartimenti accademici. Si è riscontrato un aumento del numero di studenti iscritti a un corso universitario grazie all'assistenza fornita da un chatbot¹¹.

Gli studenti impegnati nell'apprendimento di una lingua straniera spesso hanno poche opportunità per praticare la lingua. Gli insegnanti di questi corsi cercano di creare esercitazioni pratiche tramite interazioni tra pari o attività di gruppo, ma incontrano spesso ostacoli, come la mancanza di risorse adeguate o la difficoltà nel fornire feedback di qualità. I risultati della ricerca dimostrano che gli studenti di lingue preferiscono interagire con i chatbot rispetto ai tutor umani, poiché si sentono più sicuri e possono usufruirne in qualsiasi momento¹². In un altro studio condotto da Jia (2004)¹³, è stato introdotto un chatbot progettato per interagire con gli studenti di inglese, utilizzando un approccio basato su analisi sintattiche e semantiche. Questo chatbot è in grado di produrre risposte basate sul contesto della conversazione, le conoscenze dell'utente, la personalità dell'utente e le informazioni semantiche, offrendo un supporto prezioso all'apprendimento delle lingue straniere.

Servizio clienti

L'avanzamento delle nuove tecnologie ha radicalmente trasformato le interazioni interpersonali e ha apportato significative modifiche anche nell'ambito delle relazioni tra le aziende e i consumatori. Il commercio elettronico, in particolare, ha rivoluzionato il modo in cui le aziende promuovono e vendono i loro prodotti, ma con ciò sono emersi alcuni problemi relativi alla qualità del servizio clienti. Nelle conversazioni in chat dal vivo, spesso si verificano prolungati tempi di attesa per ricevere risposte da parte degli operatori aziendali, e tali risposte potrebbero non sempre essere rilevanti alle specifiche esigenze dei clienti.

Numerose aziende stanno adottando i chatbot come strumento per migliorare l'assistenza clienti. Questi chatbot offrono un servizio di assistenza disponibile 24 ore su 24, consentendo ai consumatori di inviare le loro richieste in qualsiasi momento, indipendentemente dagli orari di apertura tradizionali, contribuendo ad aumentare la soddisfazione degli utenti.

In Italia, le aziende con cui le persone dichiarano di aver conversato tramite chatbot sono soprattutto operatori di telefonia e Internet (53%), fornitori di energia (29%), marchi di elettronica e tecnologia (28%) e servizi assicurativi e bancari (25%)³¹.

Con la crescente diffusione e l'ulteriore sviluppo dei chatbot, è previsto che essi diventeranno sempre più dominanti nel settore dell'assistenza clienti, migliorando l'efficienza e l'efficacia delle interazioni azienda-cliente.

Robotica

Una delle aree di ricerca di primaria importanza nell'ambito dei chatbot è l'interfaccia in linguaggio naturale, una componente fondamentale anche per i robot fisici. In effetti, le applicazioni del linguaggio naturale nel contesto dei robot fisici sono molteplici. Ad esempio, uno studio ha introdotto una nuova interfaccia in linguaggio naturale per il robot autonomo KAMRO, già a partire dal 1994¹⁴. Sono state progettate interfacce in linguaggio naturale per impartire istruzioni a un robot vision-based¹⁵ e anche per permettere agli utenti di insegnare conoscenze sulla visione e schemi di assemblaggio a un robot fisico¹⁶. Inoltre, la programmazione di robot mediante linguaggio naturale è stata esplorata in uno studio condotto da Lauria et al. (2002)¹⁷.

Nel settore dell'educazione, vi sono numerosi studi significativi che dimostrano l'utilità dei robot fisici. Ad esempio, un robot umanoide può raccontare storie agli studenti, contribuendo a rendere l'apprendimento più coinvolgente¹⁸. Altri studi come quelli di So et al. (2018)¹⁹ e Vogt et al. (2017)²⁰, evidenziano una riduzione dell'ansia negli studenti impegnati nell'apprendimento di una lingua straniera quando un robot fisico svolge il ruolo di tutor. In un diverso contesto, un robot fisico è stato utilizzato per assistere i bambini nell'apprendimento del vocabolario, come riportato da Roy et al. (2018)²¹. Questi esempi illustrano come i robot fisici integrati con il linguaggio naturale possano offrire nuove opportunità educative e interattive.

Industria

Nella fase attuale di rapida evoluzione tecnologica, i chatbot sono ampiamente diffusi e adottati da numerose aziende e organizzazioni. Di seguito, sono forniti alcuni esempi per illustrare concretamente l'utilizzo dei chatbot in diversi settori:

- *Settore Bancario*: nel settore bancario, i chatbot sono diventati preziosi alleati per fornire assistenza ai clienti. Essi interagiscono con quest'ultimi, fornendo informazioni sul saldo del conto, agevolando il pagamento delle bollette, suggerendo strategie di risparmio e facilitando l'attivazione delle carte. Inoltre, svolgono un ruolo essenziale nella raccolta dei feedback da parte dei clienti. Alcuni esempi noti di chatbot bancari includono Erika di Bank Of America, EVA di HDFC e Ceba di Bank of Australia.
- *Settore Alimentare*: nel settore alimentare, i chatbot sono impiegati per accettare e monitorare gli ordini, gestire i dettagli delle consegne, effettuare prenotazioni, raccogliere feedback dai clienti, fornire informazioni su promozioni e sconti e rispondere alle domande basate sulle FAQ dell'azienda. Chatbot come il Pizza-bot di Dominos, il Messenger bot di Whole Foods, il chatbot di Subway e il chatbot di Burger King offrono un servizio più efficiente ed efficace ai clienti.
- *Settore della Moda*: Zalando, un marchio di moda, sfrutta i chatbot per il monitoraggio degli ordini, semplificando il processo di acquisto per i clienti.

- *Settore dei Trasporti*: la National Railroad Passenger Corporation statunitense si avvale di un chatbot, di nome Julie, per semplificare il processo di prenotazione dei biglietti ferroviari.

Questi sono solo alcuni esempi di come i chatbot vengono utilizzati in vari settori per migliorare l'esperienza dei clienti, semplificare le operazioni e fornire assistenza tempestiva. Inoltre, per chi desidera solo chattare per il puro divertimento, esistono chatbot come Kuki, Replika, Evie, Elbot e Cleverbot, che offrono conversazioni interattive e intrattenimento online.

1.3.2 Stato dell'arte nella medicina

Nell'ambito dell'assistenza sanitaria, i chatbot rivestono un ruolo significativo nell'offrire un'ampia gamma di servizi personalizzati ai pazienti. Questi chatbot sono progettati per fornire informazioni dettagliate sulla salute, terapie, prodotti e servizi correlati al paziente, e sono in grado di offrire diagnosi preliminari e suggerire trattamenti in base ai sintomi presentati dai pazienti³². Uno dei chatbot notevoli in questo campo è *OneRemission*, che si concentra sull'educazione e l'informazione riguardo al cancro, aiutando le persone a comprendere meglio la loro situazione. *Youper* si occupa della salute emotiva degli utenti, offrendo un supporto prezioso per il benessere psicologico. *Florence* funge da assistente virtuale per i pazienti, ricordando loro di assumere i farmaci prescritti. Questi sono solo alcuni esempi dei chatbot dedicati alla cura delle persone.

L'utilizzo dei chatbot in ambito sanitario offre vantaggi in diversi contesti. Essi incoraggiano il processo decisionale informato e forniscono un supporto medico continuo. Inoltre, aiutano a promuovere l'esercizio fisico, supportano la terapia cognitivo-comportamentale e affrontano disturbi somatici, garantendo trattamenti efficaci con un grado di precisione comparabile a quello dei medici umani²².

I pazienti spesso trovano i chatbot più affidabili dei medici umani poiché condividono facilmente informazioni personali e sintomi. Tuttavia, la percezione di coerenza e trasparenza rappresenta una sfida per i chatbot nell'assistenza sanitaria, e talvolta i pazienti preferiscono incontri con medici umani per situazioni più complesse. Dall'altro lato, i medici riconoscono che i chatbot sono preziosi per le attività amministrative, come la

gestione degli appuntamenti e la fornitura di promemoria per le prescrizioni. Tuttavia, vi è preoccupazione per la possibilità di errori nella conoscenza medica, e quindi i medici ritengono che i chatbot non possano sostituire decisioni mediche complesse che richiedono la consulenza di un professionista.

I chatbot sono legati al campo della psichiatria fin dai tempi di ELIZA (1.1.2, pag. 6). Essi vengono utilizzati nella prevenzione del suicidio e nell'intervento cognitivo-comportamentale, offrendo un modo più accessibile ed efficace per affrontare le sfide legate alla salute mentale²³. Tra i diversi esempi si cita Woebot, sviluppato da Woebot Labs a partire dal 2017, esso è un chatbot terapeutico basato sull'Intelligenza Artificiale. Il suo obiettivo è aiutare gli utenti a comprendere le proprie emozioni attraverso un "monitoraggio intelligente dell'umore". Questo dimostra come i chatbot possano essere utilizzati per scopi terapeutici, offrendo un supporto in ambito emotivo.

In alcune situazioni, i chatbot possono addirittura superare le aspettative dei pazienti rispetto ai medici umani, poiché non sono influenzati da pregiudizi legati a età, sesso o razza. Inoltre, essi non sono limitati dalla necessità di attendere un appuntamento, sono economicamente efficienti e possono essere attivi 24 ore su 24 per rispondere alle esigenze dei pazienti. La loro capacità di comunicare in diverse lingue li rende estremamente versatili nell'affrontare le specifiche necessità dei pazienti.

Refertazione Vocale

Si esamina, ora, il mondo degli assistenti vocali a servizio dei medici, in particolar modo per la refertazione vocale. La stesura di una relazione dettagliata a seguito di un esame o di un intervento è un lavoro necessario, ma particolarmente *time consuming* per un medico; per tale ragione sono molti i tentativi di velocizzare questa pratica. Le soluzioni praticabili sono diverse; alcuni studi²⁴ si pongono l'obiettivo di un modello di riconoscimento vocale specializzato per il supporto ai medici, al fine di ottenere uno strumento che diminuisca il tempo necessario a scrivere o a digitare un referto.

Altri lavori prevedono la creazione di un dispositivo hardware che lavora in maniera analoga ad uno smart speaker²⁵ per poi utilizzare moduli di riconoscimento vocale per ottenere del testo, da trasformare poi in dati strutturati da inserire all'interno della Cartella Clinica Elettronica.

Alcune soluzioni²⁶ si appoggiano all'infrastruttura di Amazon, utilizzando Alexa per il riconoscimento della lingua parlata. Questa soluzione permette di utilizzare l'assistente da diversi dispositivi già in commercio, come smartphone, smart speaker, ma anche smartwatch e PC; inoltre, ha il vantaggio di disporre di un modello di riconoscimento vocale con una accuracy molto alta. Alexa nel 2019 è diventata la prima assistente vocale a superare i requisiti di sicurezza dell' HIPAA (Health Insurance Portability and Accountability Act), certificando la possibilità di adoperarla in ambito medico.

È, però, importante notare come diverse comparazioni e studi scientifici^{27,28,29} affermino che la refertazione vocale non è sempre più veloce rispetto alla digitazione. Infatti, a seguito dell'utilizzo del riconoscimento vocale per la stesura di lunghi referti, è spesso necessario impiegare molto tempo per controllare nuovamente il testo generato e correggere errori di riconoscimento; proprio questa operazione è quello che spesso rende la trascrizione vocale non ottimale. Ad ogni modo, la tecnologia del riconoscimento vocale è sempre più utilizzata per la documentazione clinica. Sono state condotte ricerche per esaminare gli effetti di questa tecnologia sull'accuratezza dei referti e sulla produttività dei medici. Tuttavia, permane la necessità di una ricerca per comprendere meglio l'usabilità della Speech Recognition quando è integrata con la Cartella Clinica Elettronica (CCE) o altre piattaforme, il suo impatto sulla qualità della documentazione, l'efficienza e i costi e la soddisfazione dell'utente nel tempo e nei diversi contesti clinici. Sono, inoltre, necessari metodi di valutazione standardizzati e completi per aiutare a identificare le soluzioni per il miglioramento continuo.

Capitolo 2

Framework Rasa

In questo secondo capitolo si analizzerà Rasa, il framework scelto per lo sviluppo del chatbot oggetto di questo lavoro di tesi. Dopo una breve introduzione sul framework e una spiegazione sui motivi per cui è stato preferito rispetto ad altri competitor, analizzeremo l'architettura di Rasa e i suoi server. Si procederà con la descrizione di alcuni elementi e concetti alla base del funzionamento dell'assistente digitale, concludendo, infine, con la descrizione della preparazione dell'ambiente di lavoro

2.1 Introduzione a Rasa

Rasa è un framework AI conversazionale, basato su Python, progettato per semplificare lo sviluppo di chatbot e assistenti virtuali avanzati. Grazie ad un'architettura modulare e flessibile, esso è in grado di offrire una piattaforma completa per la creazione di interfacce conversazionali intelligenti. Rasa è caratterizzato da una serie di funzionalità potenti e da una insieme di strumenti che permettono la realizzazione di chatbot altamente personalizzati e adattabili alle esigenze specifiche delle aziende e degli sviluppatori.

Al cuore di Rasa c'è il suo approccio focalizzato sul machine learning e sulle tecniche di Natural Language Understanding (NLU) e Natural Language Processing (NLP). Ciò permette al framework di comprendere in modo accurato e contestuale le richieste degli utenti, consentendo ai chatbot di fornire risposte precise e rilevanti. La capacità di Rasa di comprendere il contesto della conversazione e le intenzioni degli utenti rende possibile la creazione di esperienze conversazionali più fluide e realistiche.

Un elemento distintivo di Rasa è la sua capacità di gestire il contesto della conversazione tramite il tracciamento del flusso di dialogo. Grazie al suo sistema di gestione dei contesti, il framework consente ai chatbot di ricordare lo stato delle conversazioni passate e di adattare dinamicamente le risposte future in base a tale contesto. Questo approccio incrementale al dialogo aiuta a creare interazioni più naturali e significative tra l'utente e il chatbot.

Inoltre, il suo approccio modulare e scalabile consente agli sviluppatori di estendere le funzionalità del framework e di personalizzarle in base alle esigenze specifiche del progetto. Con una vasta gamma di strumenti e librerie, Rasa consente agli sviluppatori di integrare facilmente nuove funzionalità, migliorare le prestazioni e adattare il chatbot per soddisfare requisiti unici di settore e di utilizzo.

Rasa offre diversi strumenti per gli sviluppatori; il lavoro di tesi si concentra sulla versione gratuita e open source di Rasa, ma sono disponibili anche altri prodotti come Rasa Pro, Rasa X/Enterprise, Rasa Platform e Rasa Studio, che offrono servizi per professionisti e aziende che desiderano maggiori funzionalità.

Sono molte le caratteristiche che fanno preferire Rasa rispetto agli altri competitor. Innanzitutto, è giusto ribadire che Rasa è una soluzione gratuita e open source, nonché uno strumento semplice da utilizzare, flessibile e "customizzabile". L'infrastruttura di Rasa è versatile e permette di riutilizzare molte componenti per diversi assistenti, applicando il lavoro già svolto a nuovi progetti.

La caratteristica più importante, nel contesto del progetto svolto, è indubbiamente la sua natura *On-premises*. Il deploy avviene sul proprio server e tutti i dati restano nella macchina locale, questo è fondamentale per risolvere problemi di privacy, soprattutto in un caso come quello relativo alla presente tesi in cui si entra in contatto di dati estremamente sensibili come i dati sanitari.

Rasa, inoltre, dichiara³³ un ROI pari al 181%, ovvero un ritorno dell'investimento in meno di un anno, e un numero di download superiore a cinquanta milioni, riunendo, nel corso degli anni, una grande comunità di appassionati che condivide consigli ed esperienze con il framework, facilitando la risoluzione di problemi.

2.2 Architettura

La figura 2.1 fornisce una panoramica dell'architettura di Rasa.

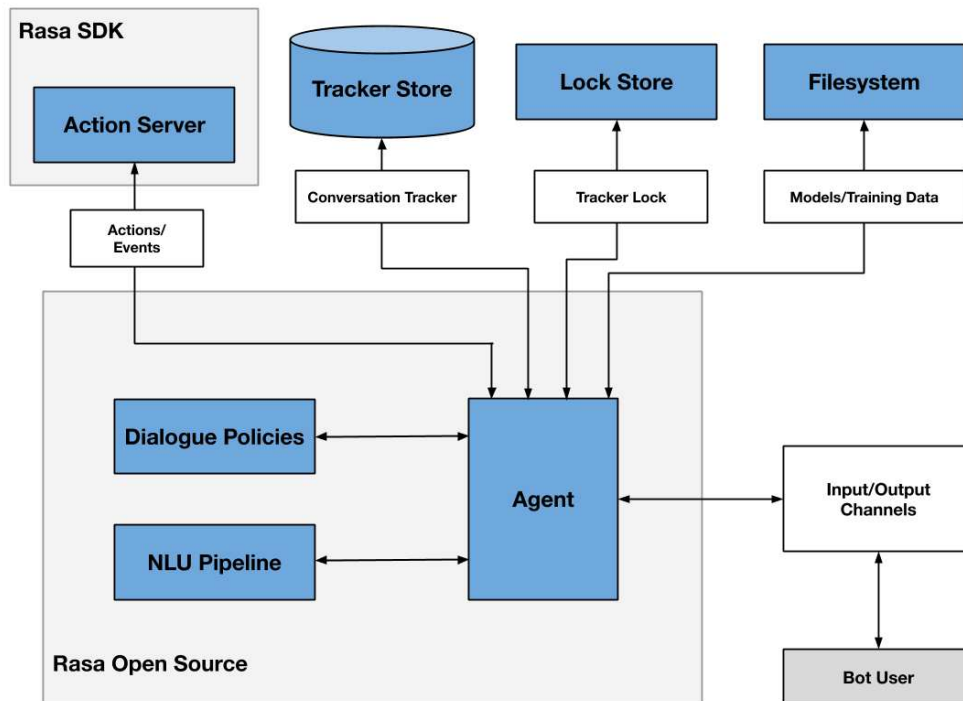


Figura 2.1: Architettura di RASA

2.2.1 Rasa Open Source

Rasa si compone di più parti, la più importante delle quali è Rasa Open Source, a sua volta divisa in tre componenti fondamentali per il funzionamento dell'assistente: l'agente, il Natural Language Understanding (NLU) e la gestione del dialogo, anche detto Rasa Core.

Agent

L'agente è l'unico componente che si interfaccia con l'utente finale, attraverso i canali di comunicazione predisposti. Ogni altro componente è collegato all'agente, il quale ha il ruolo di coordinare le varie parti.

Rasa NLU

Rasa NLU è il componente che gestisce la classificazione degli intenti dei messaggi dell'utente, l'estrazione delle entità in essi presenti e il recupero delle risposte. Viene chiamato "NLU Pipeline" perché elabora le frasi degli utenti utilizzando un modello NLU generato dalla pipeline addestrata.

Rasa Core

Il componente di gestione del dialogo decide la prossima azione basandosi sul contesto della conversazione. Esso è rappresentato nel diagramma con il nome "Dialogue Policies".

2.2.2 Rasa Action Server

Rasa Action Server è il server che si occupa di eseguire le azioni codificate dal programmatore, ovvero le cosiddette *custom actions*, che vengono analizzate più nel dettaglio in seguito.

Quando l'assistente prevede un'azione personalizzata, il server Rasa invia una richiesta POST al server delle azioni e il payload JSON della chiamata include il nome dell'azione da eseguire, l'ID della conversazione, lo stato della conversazione e il contenuto del dominio; questi concetti vengono analizzati nel seguito di questo capitolo.

Rasa SDK

Rasa SDK è un SDK Python per l'esecuzione di azioni personalizzate. Oltre a implementare le API richieste, offre metodi per interagire con il tracker delle conversazioni e comporre eventi e risposte. Esso rappresenta la scelta più immediata per il server delle azioni; è comunque possibile utilizzare un server personale, anche con linguaggi di programmazione diversi, purché accetti richieste dal server Rasa e restituisca un risposta compatibile.

2.2.3 Altri componenti

Tracker Store

Tracker Store è un componente di Rasa che memorizza e gestisce i tracker durante le conversazioni. Esso conserva lo storico degli eventi delle interazioni utente-bot, inclusi gli intenti, le azioni e gli slot, permettendo al sistema di monitorare il contesto della conversazione e di mantenere lo stato del dialogo.

Lock Store

Lock Store è un componente di Rasa che gestisce i lock per evitare la concorrenza nei sistemi multi-threading o multi-utente. Esso è responsabile della gestione dei lock e garantisce l'accesso esclusivo a determinate risorse o dati durante l'esecuzione di operazioni parallele o concorrenti.

Filesystem

Il *Filesystem* è il componente che si riferisce al sistema di archiviazione dei file utilizzato da Rasa. Esso consente di gestire e archiviare i dati, inclusi modelli di linguaggio, configurazioni, dati di training e altri file necessari per il funzionamento del chatbot, all'interno del filesystem dell'ambiente di esecuzione.

2.3 Concetti chiave

2.3.1 Training Data

Rasa utilizza file in formato YAML per la scrittura dei dati di training. Il formato YAML (*YAML Ain't Markup Language*) è un linguaggio di serializzazione dei dati leggibile dall'uomo e facilmente interpretabile da una macchina, utilizzato per la rappresentazione dei dati in modo gerarchico attraverso l'indentazione e l'uso di chiavi e valori.

Ogni file di configurazione deve contenere una o più chiavi con i corrispondenti dati di training. Ogni file può contenere più chiavi, ma ogni chiave principale (`version`,

`nlu`, `stories`, `rules`) deve essere presente al più una sola volta per ogni file. La chiave `version` rappresenta la versione di formattazione dei dati di training.

Di seguito, si spiegano in dettaglio le diverse parti dei dati di training del chatbot.

NLU Training Data

I dati di training per il riconoscimento del linguaggio naturale sono formati da esempi di possibili enunciati dell'utente. Le frasi vengono raggruppate in base all'*intent* che le caratterizza, ovvero all'intento o al significato che la frase assume. Gli esempi possono contenere *entity*; queste sono informazioni strutturate che possono essere estratte dal messaggio dell'utente. Contestualmente, si possono includere sinonimi, espressioni regolari e tabelle di ricerca il cui scopo è aiutare il modello a identificare correttamente l'intenzione dell'utente.

I dati di training NLU sono definiti sotto la chiave `nlu` all'interno dei file di configurazione. Gli oggetti da aggiungere sotto la chiave sono i seguenti:

- *Esempi di training*, raggruppati in diversi `intent` ed elencati sotto alla chiave `examples`. Il modello ha bisogno di almeno tre esempi associati ad un `intent` per classificarlo, mentre non esiste un limite massimo di frasi; tuttavia un insieme di esempi troppo ampio o eterogeneo può portare a fenomeni di overfitting.
- *Sinonimi*, servono a normalizzare i dati "mappando" un insieme di possibili valori di entità estratte ad una unica parola con lo stesso significato.
- *Espressioni regolari*, utilizzano le componenti `RegexFeaturizer` e `RegexEntityExtractor` per migliorare la classificazione degli `intent` e l'estrazione delle `entity`. Le componenti creano una rappresentazione vettoriale del messaggio dell'utente utilizzando le espressioni regolari.
- *Lookup table*, sono liste di parole utilizzate per generare pattern di espressioni regolari in maniera *case insensitive*. Tutto il contenuto delle *lookup table* viene aggregato in un'unica espressione regolare; per ogni esempio di training si utilizza la *regex* per verificare un'eventuale corrispondenza con le voci della tabella. Queste tabelle operano in maniera analoga alle espressioni regolari specificate direttamente nei dati.

Conversation Training Data

Le conversazioni tra utente e assistente sono rappresentate attraverso *story* e *rule*. Le "storie" vengono utilizzate per addestrare il modello di apprendimento a riconoscere dei pattern nelle conversazioni e a generalizzare i percorsi di conversazioni mai incontrati prima. Le "regole", invece, descrivono piccoli parti di conversazione che devono sempre seguire lo stesso percorso.

Le storie sono composte da tre sotto chiavi:

- **story**: nome arbitrario dato all'esempio di conversazione,
- **metadata**: elemento opzionale per inserire informazioni utili,
- **steps**: una lista di messaggi e azioni che formano il dialogo.

Nello specifico, ogni passo della conversazione può essere:

- un (**intent**) ricevuto dall'utente;
- un'istruzione **or** che contiene due o più possibili messaggi;
- una **action**, ovvero un'azione del bot, che a sua volta, può essere una risposta automatica o un'azione personalizzata.
- una **form**, ossia una particolare azione che esegue un ciclo su un insieme di slot, chiedendo all'utente le informazioni necessarie per definire i valori degli slot richiesti;
- uno **slot**, specificando **slot_was_set** si indica che l'utente ha associato il valore allo slot, come richiesto dalla form.
- un **checkpoint**, ossia un modo per connettere due storie tra loro.

Le regole vanno elencate sotto la chiave **rules** e somigliano alle storie; anche loro possiedono degli step. Le *rule* possono contenere anche le chiavi **conversation_started** e **conditions**, utilizzate per specificare le condizioni sotto le quali è necessario applicarle.

2.3.2 Domain

Il dominio, o *domain*, costituisce il contesto operativo dell'assistente. Esso delinea quali sono gli intenti, le entità, gli slot, le risposte, le form e le azioni essenziali che il bot deve comprendere. Inoltre, definisce una configurazione per le sessioni di conversazione.

Il dominio può essere configurato in un singolo file YAML, oppure può essere diviso in più file nella stessa cartella. Nel caso in cui venga diviso, il suo contenuto verrà automaticamente letto e unito insieme.

Intents

Nel file del dominio è importante elencare tutti gli `intent` definiti nei dati di training (sezione 2.3.1). Per ciascuno degli intent si possono specificare le entità da ignorare o da prendere in esplicita considerazione; ciò avviene attraverso l'utilizzo delle chiavi `ignore_entities` oppure `use_entities`.

Entities

L'elenco di tutte le `entities` va anch'esso definito nel file di configurazione. Ogni entità può avere come attributo un ruolo o un gruppo all'interno del dialogo per migliorare l'estrazione delle informazioni.

Slots

Gli slot rappresentano la memoria del bot. Essi agiscono come delle coppie chiave-valore che possono essere utilizzate per memorizzare le informazioni fornite dall'utente. Ogni slot ha un proprio nome, un tipo (`text`, `bool`, `float`, `categorical`, `list`, `any`) e un metodo di estrazione dell'informazione, cioè un *mapping*, che può essere:

- `from_entity`, per "mappare" gli slot dalle entità estratte;
- `from_text`, per riempire lo slot dall'ultimo testo inviato dall'utente;
- `from_intent`, per salvare un valore `my_value`, definito nel file, se l'intent dell'ultimo messaggio corrisponde ad un nome dell'intent specificato;

- `from_trigger_intent`, per riempire lo slot con un valore definito nel file se viene attivata una form da un messaggio dell'utente con un certo intent specificato.

Forms

Va definito l'elenco di *form*, il particolare tipo di azione che permette all'assistente di raccogliere le informazioni, tramite dei questionari in cui l'utente deve fornire i dati richiesti.

Responses

Nel domain vanno specificate anche le *response*. Esse sono delle risposte automatiche che il chatbot è in grado di inviare come risposta, senza dover eseguire nessun codice e senza dovere ritornare degli eventi.

In sostanza, questa tipologia di risposta è recuperata direttamente da Rasa NLU (sezione 2.2.1); non è necessario inviare una richiesta al server delle azioni (sezione 2.2.2).

Actions

Le *action* rappresentano tutto ciò che il bot è in grado di fare. Le azioni personalizzate possono essere di vario tipo, ad esempio una risposta all'utente, chiamata ad una API esterna, una query del database o qualunque altro codice eseguibile.

Nel file di dominio vanno elencati i nomi delle azioni personalizzate. La codifica delle azioni personalizzate viene analizzata in seguito.

Ulteriori parametri di configurazione

Ulteriori parametri da definire riguardano la sessione e gli slot. Una sessione può avere inizio con un messaggio dell'utente inviato al bot, con un messaggio inviato dopo un periodo di inattività o tramite uno specifico messaggio che innesca l'intent `/session_start`. Il parametro `session_expiration_time` definisce il tempo di inattività dopo il quale si avvia una nuova sessione.

Il parametro `carry_over_slots_to_new_session` determina se i valori degli slots salvati vanno mantenuti nella nuova sessione oppure no.

Nel file di dominio, la chiave di configurazione `store_entities_as_slots` controlla il comportamento relativo alla gestione delle entità come slot durante la lettura e la trasformazione delle storie nel tracker. Se impostato a `true`, questo parametro determina che gli slot vengano automaticamente impostati dalle entità pertinenti presenti nelle storie. Quando un'entità corrisponde al mapping `from_entity` dello slot, `store_entities_as_slots` stabilisce se il suo valore deve essere inserito in tale slot. Questo parametro, di fatto, sostituisce l'aggiunta manuale dello slot in un passaggio esplicito della storia; per impostazione predefinita questa funzionalità è abilitata.

2.3.3 Actions

Dopo ogni messaggio dell'utente, il modello prevede un'azione che l'assistente dovrà eseguire. Nel seguito si procede con un riepilogo di tutte le differenti tipologie di azioni, approfondendo quelle non ancora trattate.

Responses

Le *response* sono le risposte automatiche che l'assistente invia all'utente a seguito di specifici messaggi. Le risposte sono generalmente testuali, ma possono includere anche immagini, video o pulsanti per risposte multiple.

Le risposte, definite nel domain, possono contenere anche delle variabili che vengono valorizzate con informazioni fornite dall'utente, ad esempio con il suo nome. Si possono, poi, prevedere delle variazioni sulle risposte, elencando più risposte possibili tra le quali l'assistente potrà scegliere casualmente quale fornire all'utente; si possono, altresì, prevedere risposte diverse a seconda di condizioni specificate nel file di dominio o in base al canale sul quale sta avvenendo la conversazione, sia esso Telegram, Socketio, Slack, etc.

Forms

Le *Forms* sono uno dei più comuni pattern di conversazione per ottenere informazioni dall'utente al fine di attivare delle azioni con le informazioni raccolte. Il meccanismo è fortemente legato agli slot, i quali vengono riempiti tramite i questionari.

Attraverso la scrittura di regole o storie (sezione 2.3.1) è possibile determinare le condizioni di attivazione, specificando quale messaggio dell'utente innesca il ciclo del form e quali azioni compiere dopo che l'utente ha fornito l'ultimo slot richiesto.

Azioni di convalida degli slots

Le azioni di convalida degli slot rappresentano una particolare tipologia di azioni personalizzate che si occupano di gestire l'estrazione o la validazione dei valori degli slot. Esistono tre diversi modi per validare i dati:

- `action_validate_slot_mappings`; rappresenta un'azione per la convalida e l'estrazione personalizzata degli slot, eseguita dopo `action_extract_slots`, senza la necessità di specificare esplicitamente l'azione nell'impostazione dei mapping degli slot.
- `validate_<form_name>`; rappresenta azioni personalizzate eseguite automaticamente se il modulo specificato è attivato, richiedendo la creazione di una classe di azioni equivalente all'SDK Rasa se quest'ultimo non è utilizzato. L'azione eredita da `FormValidationAction` per il Rasa SDK.
- Azione personalizzata classica; questa consente l'estrazione personalizzata degli slot in modo flessibile ed è utile se le alternative precedenti non soddisfano i requisiti. È richiesta la specifica dell'azione nel mapping degli slot e nelle action del dominio.

Azioni personalizzate

Le azioni personalizzate (*custom action*) consentono di eseguire qualunque codice Python si voglia. Esse vengono eseguite nell'Action Server (sezione 2.2.2).

La classe `Action` è la classe base per qualsiasi azione personalizzata. Per definire un'azione personalizzata occorre creare una sottoclasse di `Action` e sovrascrivere i due metodi richiesti, `name` e `run`.

Il metodo `name` definisce il nome dell'azione. Il nome che viene ritornato dal metodo è lo stesso che deve essere usato nel dominio del bot (sezione 2.3.2).

Il metodo `run` esegue in maniera effettiva l'azione. Esso prevede i seguenti parametri:

- **dispatcher**; utilizzato per inviare messaggi all'utente;
- **tracker**; rappresenta lo stato della conversazione con l'utente; esso permette di accedere agli slot o ai messaggi ricevuti;
- **domain**; rappresenta il dominio del bot.

Azioni predefinite

Le azioni predefinite rappresentano le azioni integrate nel gestore del dialogo di default; la maggior parte di esse vengono automaticamente predette in base a determinate situazioni di conversazione. È possibile personalizzare tali action per modificare il comportamento standard del bot.

Le azioni predefinite sono molte, di seguito si riassumo solamente le più significative ed utili al fine del progetto.

- **action_listen**: questa azione viene attivata quando il bot deve solo attendere un messaggio da parte dell'utente, senza fare altro.
- **action_restart**: questa azione effettua il reset l'intera cronologia di conversazione, inclusi gli slot salvati. Si attiva con il messaggio `/restart` oppure specificando un intent.
- **action_session_start**: questa azione avvia una nuova conversazione. È utile modificarla per permettere di intercettare i metadati inviati dall'utente.
- **action_default_fallback**: questa azione annulla l'ultima interazione utente-bot e invia la risposta di default, se definita. Essa viene attivata da una bassa confidenza nella previsione dell'azione.
- **action_unlikely_intent**: questa azione si attiva in caso di intent non previsti, si può modificare il parametro della tolleranza per variare la frequenza con cui far partire quest'azione.

2.3.4 Channel Connector

Rasa permette di collegare il proprio assistente a diversi canali, questi possono essere canali testuali, vocali o personalizzati. Le funzionalità dei canali vocali sono al momento disponibili solo per utenti di Rasa Pro e non vengono, dunque, presi in esame. I possibili canali testuali per il collegamento tra utente e bot sono i seguenti:

- Facebook Messenger
- Slack
- Telegram
- Twilio
- Google Hangouts Chat
- Microsoft Bot Framework
- Cisco Webex Teams
- RocketChat
- Mattermost
- Interfaccia Web

Al fine di questo progetto, si analizza unicamente come avviene la connessione tra il chatbot e una propria interfaccia web.

Rasa fornisce un widget già pronto da inserire nella propria pagina web³⁴; tuttavia, è possibile costruire autonomamente la propria interfaccia con l'assistente.

REST

Sono disponibili due canali per l'integrazione tramite REST; ovvero `RestInput` e `Call-backInput`.

Il canale REST fornisce un endpoint da cui è possibile ricevere e inviare messaggi. È sufficiente specificare il canale REST nel file di configurazione `credentials.yml`. I messaggi possono essere inviati all'indirizzo `http://<host>:<port>/webhooks/rest/webhook` tramite chiamata di tipo POST con il seguente formato:

```
{
  "sender": "mario_user", //ID dell'utente
  "message": "Ciao, Bot!" //Messaggio
}
```

La risposta ritorna dal bot sotto forma di file JSON.

```
[
  {"text": "Hey Rasa!"},
  {"image": "http://example.com/image.jpg"}
]
```

Il canale Callback opera in maniera simile al canale REST, con la sola eccezione di effettuare una chiamata a un URL specificato per l'invio dei messaggi bot, anziché restituire direttamente i messaggi bot alla richiesta HTTP del mittente.

Per attivare il canale callback è necessario inserire l'URL nel file `credentials.yml`.

Invio e ricezione dei messaggi avvengono in maniera analoga al canale REST.

Websocket

Il canale SocketIO si avvale di Websocket e opera in tempo reale. Per configurare e utilizzare il canale SocketIO è necessario inserire le credenziali nel file `credentials.yml` con i primi due parametri di configurazione che definiscono i nomi degli eventi utilizzati da Rasa per l'invio e la ricezione dei messaggi tramite SocketIO.

Il client Socket ha la possibilità di trasmettere un metadato per fornire ulteriori informazioni al canale. È possibile impostare una chiave alternativa mediante l'impostazione `metadata_key`.

Sarà possibile inviare messaggi a `http://<host>:<port>/socket.io` sostituendo "host" e "port" con i valori appropriati del server Rasa in esecuzione.

Per impostazione di default, il canale SocketIO utilizza l'ID del Socket come `sender_id`, il che comporta il reset della sessione ad ogni ricarica della pagina. Si può modificare questo comportamento modificando il parametro della persistenza e lasciando generare al front-end un ID persistente.

2.4 Setup dell'ambiente di sviluppo

L'unico pre-requisito necessario per l'utilizzo di Rasa è quello di avere un ambiente Python correttamente configurato. Le versioni di Python attualmente supportate da Rasa sono 3.7, 3.8, 3.9 e 3.10; quest'ultima versione è supportata solo dalle Versioni 3.4.x o superiori di Rasa. La versione di Python utilizzata per il progetto è la 3.9.

È buona norma, ma non strettamente necessario, installare Python, Rasa ed altre dipendenze utili al progetto in un ambiente virtuale. Nello specifico, per la realizzazione di questo progetto, è stato utilizzato Miniconda, un installer gratuito e minimale per la realizzazione di virtual environment tramite `conda`.

Dopo essersi assicurati, da terminale, di utilizzare la versione più recente di `pip` tramite il comando:

```
pip3 install -U pip
```

si può procedere con l'installazione di Rasa Open Source tramite comando:

```
pip3 install rasa
```

A questo punto il sistema è pronto per l'avvio di un progetto Rasa; tale avvio può essere effettuato tramite il comando

```
rasa init
```

In fase di avvio del progetto, è conveniente installare le dipendenze che si utilizzeranno. Per la realizzazione di questo chatbot è stata utilizzata `spaCy`, una libreria gratuita e open-source per l'elaborazione del linguaggio naturale in Python. `SpaCy` supporta oltre 73 lingue, implementa un apprendimento multi-task con Transformer pre-allenati come BERT. Il modulo è utilizzato per la tokenizzazione e per il riconoscimento delle parole, risultando una soluzione robusta e con una precisione rigorosamente valutata. Il modulo per il riconoscimento delle parole italiane può essere installato come segue:

```
python -m spacy download it_core_news_sm
```

Al termine di tutte le installazioni di dipendenze utili, il sistema è pronto per lo sviluppo dell'assistente.

Capitolo 3

Analisi dei requisiti e progettazione

Il terzo capitolo inizierà con la raccolta dei requisiti; in questa sezione si analizzeranno gli attori che utilizzeranno l'applicazione, ovvero i medici; proseguiamo, poi, con l'elencazione dei requisiti funzionali del chatbot e dei requisiti non funzionali, come la sicurezza, la privacy e le performance. Dopodiché, procederemo con la descrizione delle scelte progettuali per la realizzazione dell'assistente digitale e delle interfacce front-end; infine, mostreremo i diagrammi di flusso e diagrammi di sequenza realizzati in fasi di progettazione.

3.1 Raccolta dei requisiti

Il progetto da realizzare consiste nella creazione di un chatbot in tecnologia RASA per la gestione dei dati dei pazienti in una struttura sanitaria. Il chatbot desiderato deve avere una duplice modalità di utilizzo:, ovvero:

- *testuale*: il medico utilizza la tastiera per conversare con l'assistente, ottenendo la risposta, sempre in maniera testuale, su interfaccia di messaggistica
- *vocale*: il medico utilizza la voce per impartire l'ordine al chatbot, il quale risponde riproducendo il messaggio di risposta.

3.1.1 Descrizione dell'Attore

L'attore principale che si relaziona con l'assistente digitale è il professionista sanitario che vuole gestire le attività delle cure, come il diario sanitario, i parametri vitali, la

somministrazione delle terapie ed altre funzioni utili nel lavoro tra i reparti.

Gli operatori sanitari si dividono in tre possibili categorie.

- *Comparto*: comprende il personale non medico e non dirigente, dipendente dalle aziende sanitarie ed ospedaliere del Servizio sanitario nazionale.
- *Coordinatore*: colui che, tra i Medici Competenti, svolge le funzioni di coordinamento. È presente in casi di azienda medica con più Unità Operative o gruppi di impresa.
- *Direzione medica*: il medico a cui è conferita la direzione sanitaria del presidio ospedaliero.

Nonostante i tre livelli di autenticazione, le funzionalità del chatbot sono le medesime. La differenza principale tra le diverse categorie risiede nelle diverse Unità Operative a cui possono accedere; in linea generale i livelli superiori dell'organigramma medico hanno accesso ad un numero maggiore di Unità Operative. Altre differenze non rientrano negli obiettivi di questo lavoro di tesi, come, ad esempio, l'accesso a fatture e documenti amministrativi.

L'assistente si relaziona, poi, con altri sistemi, come il database dai dati sanitari, attraverso l'utilizzo di API.

Si prevede che le interazioni vocali con l'assistente avvengano mediante l'utilizzo di auricolari *Bluetooth* o cablati, al fine di migliorare il riconoscimento delle parole in fase di trascrizione e, contemporaneamente, evitare che persone non autorizzate ascoltino informazioni personali sui pazienti. Tuttavia, il rispetto o meno di questa indicazione è a totale arbitrio del medico che utilizza il chatbot.

3.1.2 Requisiti Funzionali

Le funzioni che devono essere implementate nell'assistente sono le seguenti:

- fornire l'elenco delle Unità Operative selezionabili dall'utente;
- selezionare e ricordare, tramite il meccanismo degli slot, l'Unità Operativa presso cui sta lavorando l'operatore sanitario;

- fornire l'elenco dei pazienti in una specifica Unità Operativa;
- selezionare il paziente specificato dall'utente;
- accedere alle informazioni generali del paziente selezionato;
- accedere ai parametri vitali del paziente;
- salvare nuovi parametri vitali all'interno della cartella clinica del paziente;
- leggere i risultati di esami svolti dal paziente;
- confrontare i risultati di uno stesso esame nel corso del tempo;
- salvare un nuovo risultato di un esame;
- leggere le note salvate nel diario clinico del paziente;
- salvare una nuova nota nel diario clinico;
- elencare le allergie del paziente;
- elencare le terapie a cui è sottoposto il paziente;
- elencare le terapie a cui sottoporre il paziente nella fascia oraria
- fornire una panoramica del numero di terapie da somministrare nelle diverse Unità Operative.

Il diagramma dei casi d'uso, mostrato nella Figura 3.1, riassume le funzionalità che il chatbot è in grado di compiere. L'attore, ovvero l'operatore sanitario, inizialmente ha un numero ridotto di funzionalità disponibili. Appena si avvia il chatbot si può soltanto conoscere l'elenco delle Unità Operative, sapere il numero di terapie da somministrare nei diversi reparti, oppure selezionare un'Unità Operativa, la quale viene salvata negli slot dell'assistente.

Selezionare un'Unità Operativa è necessario per svolgere ulteriori funzioni, come conoscere l'elenco dei pazienti nel reparto selezionato oppure salvare uno specifico paziente negli slot.

Le azioni eseguite su un singolo paziente, ovvero la maggioranza di tutte le azioni disponibili, sono attivabili solo dopo aver selezionato il paziente. Queste azioni, dunque, rappresentano una estensione delle funzionalità a cui l'utente può accedere prima della selezione del paziente stesso.

Tutti i casi d'uso del chatbot necessitano il collegamento, attraverso l'utilizzo di API aziendali, con il database medico, rappresentato nel diagramma come un attore collegato all'intero sistema del chatbot.

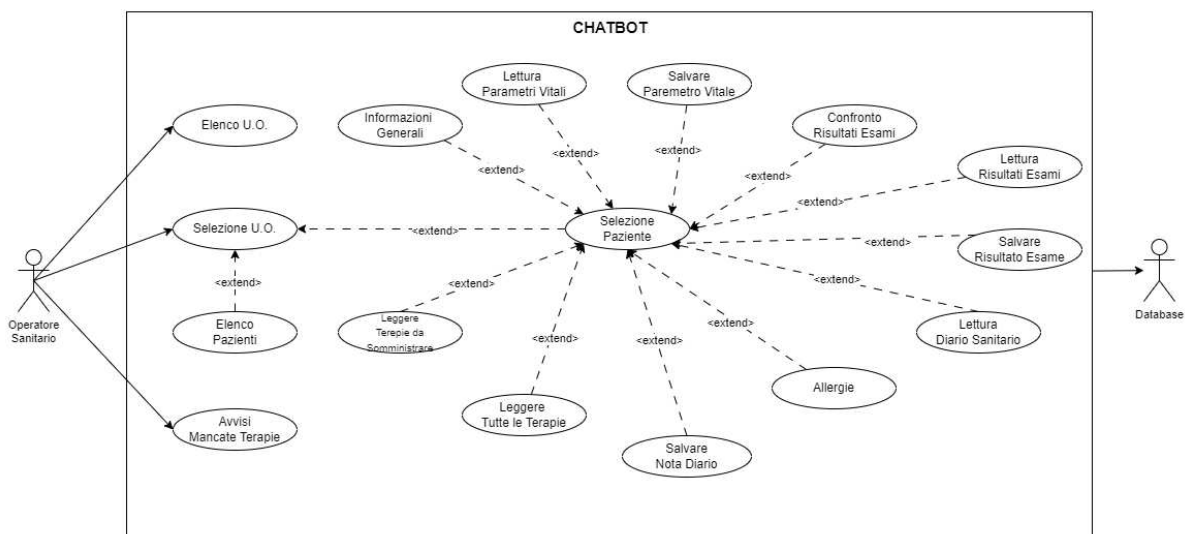


Figura 3.1: Diagramma dei casi d'uso

3.1.3 Requisiti Non-Funzionali

Requisiti di sicurezza

È importante rimarcare quanto sia fondamentale l'aspetto della sicurezza dei dati e della privacy dei pazienti quando si lavora con dati sensibili, come quelli sanitari.

In primo luogo, solo il personale sanitario di una struttura ospedaliera può utilizzare l'assistente; per questa ragione è necessaria un'autenticazione dell'utente per accedere alle funzionalità del chatbot.

È importante che i dati, ovvero le conversazioni che avvengono tra l'utente e l'assistente digitale, rimangano all'interno di server aziendali; nel caso di questo lavoro il

server è gestito dall'azienda produttrice del software. Proprio per tale ragione è stato scelto il framework Rasa per la creazione del chatbot; esso, infatti, grazie alla sua natura on-premises, permette di far funzionare il modello su un proprio server, come visto nella Sezione 2.1.

Sempre per la stessa ragione, per la realizzazione del supporto vocale del chatbot, sono stati utilizzati dei componenti nativi di *text-to-speech* (TTS) e *speech-to-text* (STT) per la conversione messaggio testuale/voce, e viceversa. Tali componenti funzionano in locale e non necessitano di server esterni.

Requisiti di privacy

I requisiti di privacy sono strettamente correlati ai requisiti di sicurezza; molte delle soluzioni adottate per la sicurezza si estendono anche alla protezione della riservatezza dei dati dell'utente.

Un'ulteriore accortezza che è stata richiesta è la necessità che il chatbot richieda messaggi di conferma prima di eseguire ogni azione. Ciò risulta fondamentale per almeno due motivi: innanzitutto per evitare di eseguire azioni su un paziente errato, ma anche per accertarsi che, durante l'utilizzo dell'assistente vocale nei reparti, i pazienti non ascoltino dati di altre persone erroneamente. L'utente può, infatti, fermare l'esecuzione di una determinata azione da parte del chatbot se quest'ultimo lo avverte che sta eseguendo le azioni su un paziente diverso da quello che ha di fronte in reparto,

Requisiti di performance

Data la natura anche vocale dell'assistente, è emersa la necessità di ridurre il più possibile i tempi di risposta del chatbot, in modo da avere conversazioni il più realistiche possibili. Pur non essendo stato definito in maniera formale un limite massimo per l'intervallo di tempo tra la richiesta da parte dell'utente e la risposta dell'assistente digitale, nel corso della realizzazione si è tenuta in considerazione l'importanza di questo aspetto.

3.2 Progettazione

Lo schema di funzionamento generale dell'assistente è mostrato nella Figura 3.2

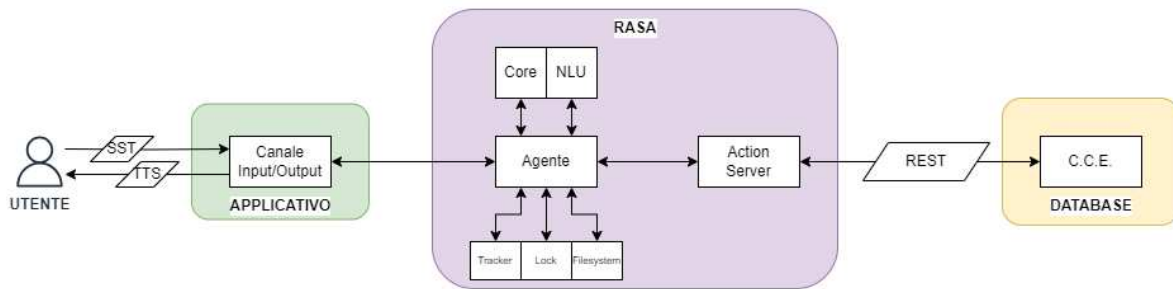


Figura 3.2: Diagramma di funzionamento dell'assistente

L'utente ha due possibilità per comunicare con l'assistente; può scrivere e inviare il suo messaggio utilizzando la tastiera oppure può accendere il microfono dell'interfaccia e parlare naturalmente; in quest'ultimo caso le sue frasi verranno tradotte in testo grazie al componente speech-to-text. Il testo del messaggio arriva all'agente grazie al canale SocketIO (Sezione 2.3.4).

L'agente analizza il messaggio tramite Rasa Core e Rasa NLU; a seguito di ciò, effettua una chiamata all'*action server* per attivare l'azione corrispondente al messaggio ricevuto dall'utente.

Se l'azione attivata lo prevede, si effettua una chiamata REST per accedere ai dati della struttura sanitaria e dei pazienti. A tale scopo si utilizzano le API già sviluppate dall'azienda per accedere al database.

Dopo aver ottenuto tutte le informazioni necessarie per formulare la risposta da dare al messaggio dell'utente, l'agente può restituire il messaggio a quest'ultimo, sempre tramite socket. Nel caso in cui l'utente desideri ascoltare la risposta, egli può attivare la lettura del messaggio tramite un'apposita icona; il messaggio viene convertito in voce grazie al componente nativo text-to-speech del dispositivo.

3.2.1 Chatbot

Scelte progettuali

In fase di progettazione è stato deciso di non utilizzare entità all'interno delle frasi per l'estrazione di dati non strutturati da parte del modello. Nonostante l'estrazione di entità sia uno dei principali vantaggi di Rasa rispetto ad altri competitor, questa decisione è

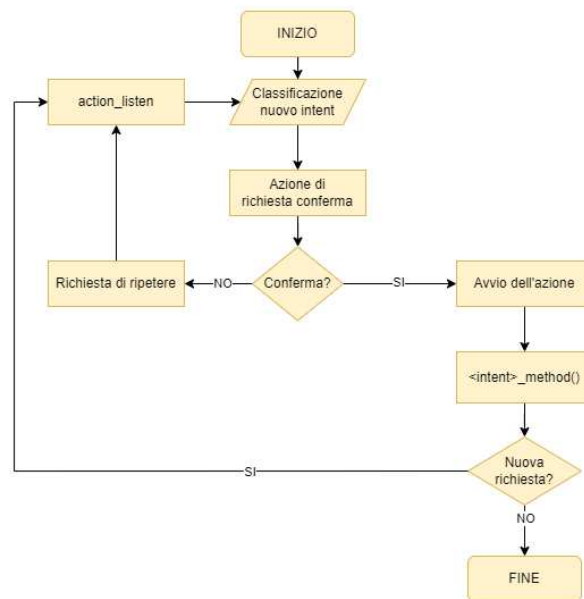


Figura 3.3: Diagrammi di flusso della conversazione

stata presa per ridurre al minimo gli errori di comprensione. Molte delle possibili entità da estrarre, come nomi e cognomi di pazienti, codici di cartelle cliniche, nomi di Unità Operative ed altre, potevano essere difficili da riconoscere per il modello, e spesso mal interpretabili; ad l'utente potrebbe invertire nome e cognome di un paziente portando l'agente a commettere errori.

La mancata estrazione di una entità, oppure l'estrazione di un dato errato, avrebbero generato in maniera persistente dei problemi nell'esecuzione delle azioni; questi a loro volta avrebbero reso la conversazione con il chatbot meno scorrevole.

Per questa ragione, si è deciso di raccogliere le informazioni tramite delle form, in modo da strutturare all'interno di determinate regole l'estrazione dei valori richiesti.

Diagramma di flusso della conversazione

La progettazione del chatbot è basata sul digramma di flusso mostrato in Figura 3.3; tale diagramma mostra come dovrebbe avvenire una conversazione tipo tra l'utente e il bot.

Al momento dell'invio del primo messaggio, l'agente riconosce il primo intent. Se-

guendo i requisiti non funzionali richiesti (Sezione 3.1.3), il bot chiede all'utente se l'intent è stato correttamente interpretato. Nel caso di risposta negativa, il chatbot chiede all'utente di ripetere il comando e resta in ascolto della nuova frase. Altrimenti, nel caso di risposta affermativa, l'agente avvia l'azione da eseguire richiamando funzioni specifiche in base all'intent della frase.

Una volta che il bot ha terminato l'azione fornendo all'utente le informazioni richieste, esso ritorna in ascolto del prossimo messaggio. Il ciclo di conversazione termina quando l'utente conclude la conversazione con il bot.

La conversazione si può discostare dal diagramma di flusso mostrato in due possibili casi; ovvero:

- ci sono messaggi che non necessitano di conferma, come, ad esempio, i messaggi di saluto che attivano risposte automatiche (Sezione 2.3.3);
- ci sono messaggi che attivano delle form di Rasa, come, ad esempio, il salvataggio di nuovi parametri vitali per i quali si utilizzano meccanismi di validazione differenti specifici per le form (Sezione 2.3.3).

Riconoscimento degli intent

Al fine di eseguire tutte le funzionalità, l'agente deve essere in grado di riconoscere tre principali categorie di intent da parte dell'utente, ovvero: intent di messaggi di conversazione generici (Tabella 3.1); intent per il salvataggio di nuovi dati (Tabella 3.2), e intent per la chiamate REST al database (Tabella 3.3).

Gli intent per i messaggi di conversazione più generali non prevedono l'esecuzione di azioni personalizzate; il bot risponde all'utente tramite response automatiche.

Gli intent per le richieste di nuovi salvataggi attivano delle form per l'inserimento dei valori richiesti. I dati forniti vengono utilizzati per il salvataggio delle informazioni negli slot e/o nel database medico tramite chiamate REST.

Infine, la restante parte degli intent prevede l'invio di chiamate REST per l'ottenimento di informazioni mediche; ogni chiamata avrà bisogno di dati all'interno del body della richiesta, come il token di autenticazione del medico oppure il codice di riconoscimento del paziente e dell'Unità Operativa. Alcuni dei valori richiesti per la costruzione del

Nome intent	Descrizione
saluto	Messaggio di saluto iniziale
addio	Messaggio di congedo
conferma	Messaggio di conferma
negazione	Messaggio di negazione
aiuto	Richiesta di informazioni sul funzionamento del chatbot
conversazione_bot	Messaggio per capire se si sta parlando con un chatbot

Tabella 3.1: Intent generici

Nome intent	Descrizione
request_unita_operativa	Richiesta di selezione di una Unità Operativa
request_paziente	Richiesta di selezione di un paziente
request_parametro_vitale	Richiesta di salvataggio di un nuovo parametro vitale
request_registrazione_diario	Richiesta di registrazione di una nuova nota nel diario clinico
request_andamento_esami	Richiesta dei risultati di una tipologia di esame nel corso del tempo

Tabella 3.2: Intent per attivazioni form

corpo della chiamata REST vengono estratti automaticamente dai metadati della conversazione; altri, invece, vengono recuperati dagli slot precedentemente salvati tramite le form.

Diagramma di sequenza

Nella Figura 3.4 è mostrato il diagramma di sequenza nel caso di scambio di messaggi con risposta affermativa al messaggio di conferma.

Innanzitutto, si vede come all'avvio del chatbot ci sia il salvataggio dei dati di autenticazione del medico; questo avviene tramite l'override della funzione `action_session_start()`. Tramite questa funzione è possibile intercettare i metadati, contenenti

Nome intent	Descrizione
<code>res_get_uo</code>	Richiesta dell'elenco delle Unità Operative
<code>res_elenco_ricoverati</code>	Richiesta dell'elenco dei ricoverati
<code>res_parametri_vitali_codificati</code>	Richiesta dell'elenco dei valori dei parametri vitali
<code>res_terapie_assistito</code>	Richiesta dell'elenco delle terapie a cui è sottoposto il paziente
<code>res_somministrazione_terapia</code>	Richiesta dell'elenco delle terapie da somministrare al paziente in quella determinata fascia oraria
<code>res_allergie_assistito</code>	Richiesta dell'elenco delle allergie del paziente
<code>res_info_assistito</code>	Richiesta delle informazioni generali sul paziente
<code>res_diario_assistito</code>	Richiesta di lettura delle note del diario clinico del paziente
<code>res_avvisi_terapia</code>	Richiesta dell'elenco del numero di terapie da somministrare nei diversi reparti
<code>res_compara_esami</code>	Richiesta dell'elenco di risultati degli ultimi esami svolti dal paziente

Tabella 3.3: Intent per chiamate REST

i dati di autenticazione, token dell'utente e id del dispositivo; essi vengono memorizzati negli slot di Rasa. Tale meccanismo avviene ad ogni nuovo avvio della conversazione.

A questo punto, una volta che l'utente invia un messaggio, esso viene ricevuto dall'agente tramite il canale SocketIO. L'agente riconosce l'intent e avvia l'azione per la richiesta di conferma, memorizzando negli slot l'intent ricavato. Questo dato negli slot viene ripreso dopo il messaggio di conferma da parte dell'utente; l'intent salvato è utile per sapere quale metodo richiamare nella classe `action_start`. Ad ogni intent è associato un metodo che fa partire la specifica chiamata REST per ottenere i dati sanitari.

La Figura 3.5 mostra il diagramma di sequenza nel caso in cui l'utente non confermi

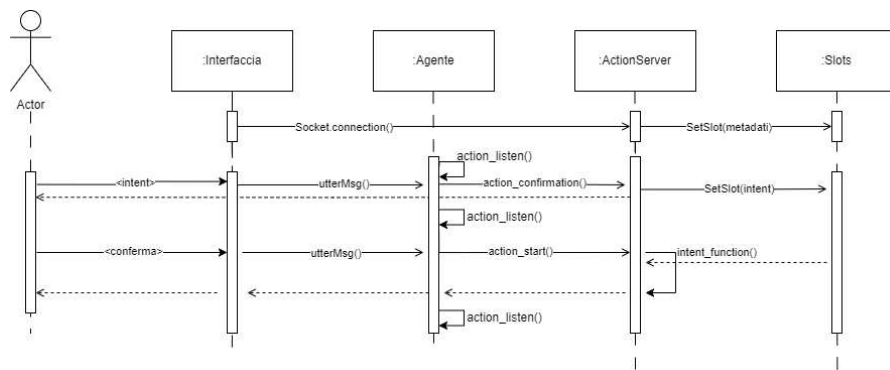


Figura 3.4: Diagramma di sequenza con messaggio di conferma

l'esecuzione dell'azione. In questo caso, dopo il messaggio di negazione, l'agente invia un messaggio all'utente chiedendo di riformulare la frase. Lo slot memorizzato, ormai inutilizzato, sarà aggiornato dall'intent successivo.

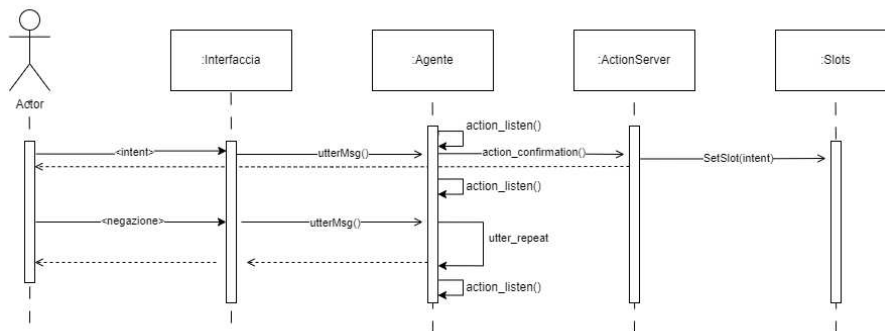


Figura 3.5: Diagramma di sequenza con messaggio di negazione

La Figura 3.6 mostra il diagramma di sequenza nel caso in cui il messaggio inviato dell'utente non richieda una conferma, come nel caso del messaggio di saluto. In questo caso, l'agente invia all'utente un messaggio predefinito impostato dal programmatore.

La Figura 3.6 mostra il diagramma di sequenza nel caso in cui ci sia l'attivazione di una form di Rasa. In questo caso la validazione è effettuata da funzioni predisposte all'interno del server delle azioni. Una volta attivata la form, all'utente viene chiesto di fornire le informazioni utili a "valorizzare" gli slot richiesti. Ad ogni slot inserito, la classe `validate_form` attiva i controlli sul tipo e sul valore di dati. In caso di inserimento

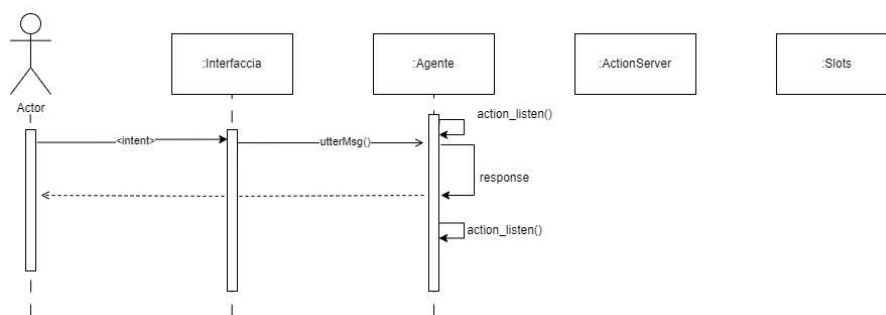


Figura 3.6: Diagramma di sequenza con response automatica

di un valore errato, il bot richiede nuovamente di effettuare l’inserimento specificando il tipo di errore.

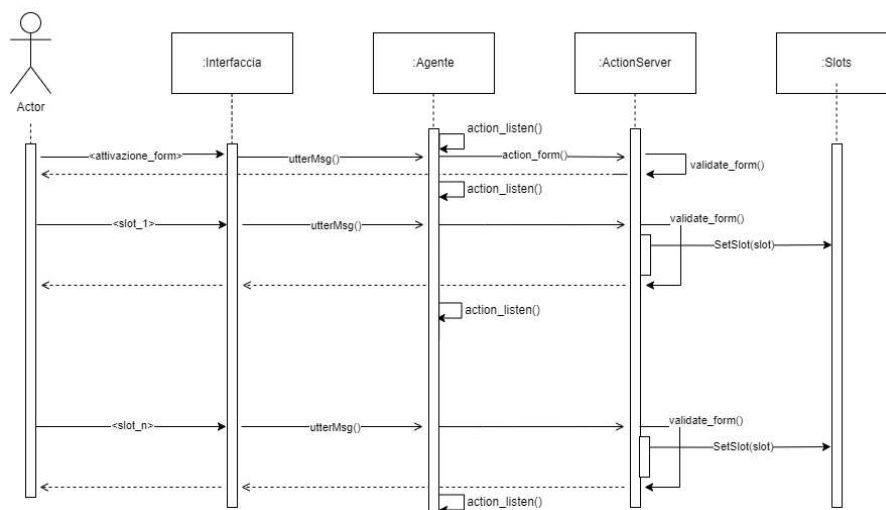


Figura 3.7: Diagramma di sequenza con validazione form

3.2.2 Interfaccia Utente

Sono state progettate due possibili interfacce per ospitare il chatbot; ovvero:

- *Interfaccia Web*; essa è scritta in HTML, CSS e JavaScript e realizza un layout semplice e minimale per la messaggistica istantanea con l’assistente digitale.

- *Applicazione Mobile*; sviluppata tramite framework Ionic per realizzare una soluzione cross-platform (Android e iOS).

Interfaccia web

La pagina HTML, il cui mockup iniziale è visibile nella Figura 3.8, è composta da un ridotto numero di elementi che compongono l'interfaccia di messaggistica con l'agente; questi sono:

- una *text-box* dove inserire il messaggio da inviare al bot;
- un pulsante per inviare il messaggio, in alternativa al pulsante "Invio" della tastiera;
- un pulsante per attivare o disattivare il microfono per la dettatura;
- un pulsante per attivare o disattivare la lettura dei messaggi ricevuti;
- un insieme di riquadri con i messaggi inviati e ricevuti.

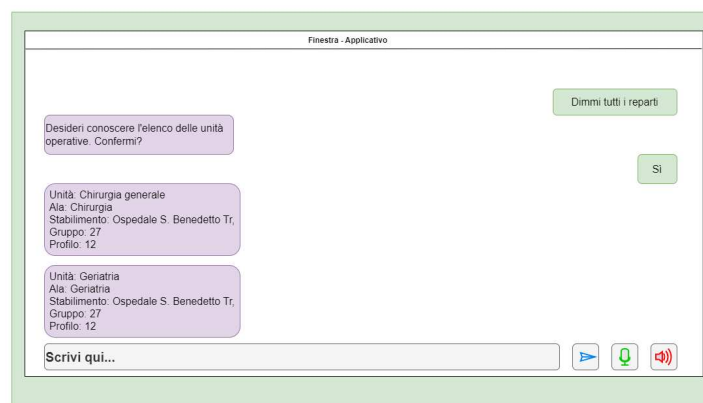


Figura 3.8: Mockup dell'interfaccia web

Per la trascrizione e la sintesi vocale si utilizzano i metodi `SpeechSynthesis` e `SpeechRecognition` delle `Web Speech API` integrate nel browser.

Applicazione mobile

L'applicazione per smartphone rappresenta il modo principale per interagire con l'assistente vocale. A tal fine si è progettata la realizzazione di una pagina dedicata al chatbot all'interno dell'applicazione "Smart4Hospital"; essa è sviluppata e rilasciata da NBS s.r.l. In fase di progettazione si è preferito aggiungere le funzionalità del chatbot all'interno dell'applicazione già presente sul mercato ed utilizzata, piuttosto che rilasciare una nuova applicazione.

In modo analogo al front-end per il browser, la schermata dell'app contiene:

- una *text-box* dove inserire il messaggio da inviare al bot;
- un pulsante per inviare il messaggio, in alternativa al pulsante "Invio" della tastiera;
- un pulsante per attivare o disattivare il microfono per la dettatura;
- un pulsante per attivare o disattivare la lettura dei messaggi ricevuti;
- un insieme di riquadri con i messaggi inviati e ricevuti.

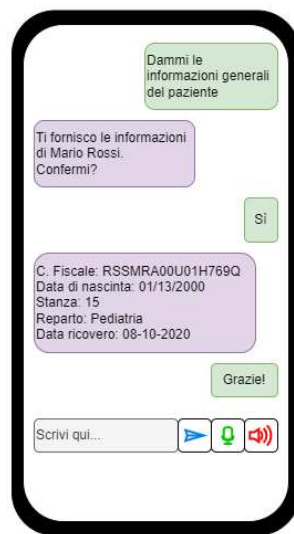


Figura 3.9: Mockup dell'interfaccia per smartphone

Per la trascrizione e la sintesi vocale si utilizzano i plugin `@capacitor-community/text-to-speech` e `@capacitor-community/speech-recognition` compatibili con Ionic.

Diagramma di sequenza

Nella Figura 3.10 viene mostrato il digramma di sequenza nel caso in cui l'utente utilizzi la tastiera per comunicare con il chatbot.

Nella Figura 3.11 viene mostrato, invece, il digramma di sequenza nel caso in cui l'utente utilizzi la voce per inviare e l'altoparlante per ascoltare i messaggi.

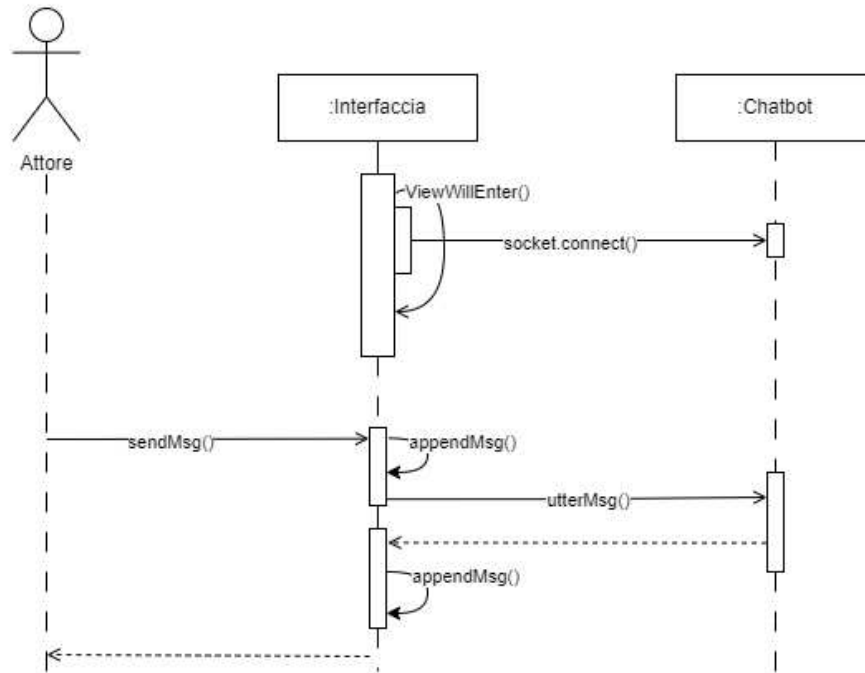


Figura 3.10: Diagramma di sequenza nel caso di invio di messaggi tramite tastiera

I due diagrammi sono sostanzialmente i medesimi sia per la versione mobile sia per la versione web del front-end. Innanzitutto, all'avvio dell'interfaccia, avviene automaticamente la connessione con il chatbot tramite socket. A questo punto, l'utente può iniziare ad inviare i messaggi.

Nel caso in cui si utilizza la tastiera, si attiva un metodo per aggiungere il messaggio alla lista dei messaggi inviati e per inviare il messaggio all'agente. Quando si ottiene una risposta dall'agente, il messaggio ricevuto viene aggiunto alla lista dei messaggi ricevuti permettendo la visualizzazione della risposta all'utente. Invece, se l'utente utilizzasse il microfono per dettare i messaggi, questi ultimi vengono tradotti in testo tramite il com-

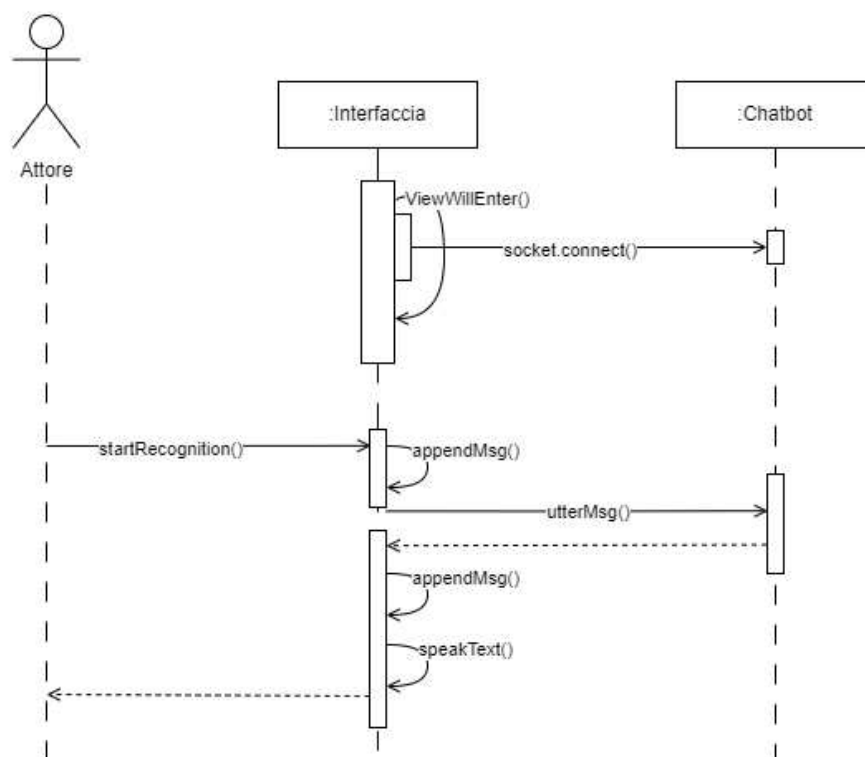


Figura 3.11: Diagramma di sequenza nel caso di invio di messaggi tramite voce

ponente speech-to-text, per poi proseguire come nel caso precedente. Se l'utente attiva la sintesi vocale delle risposte, esse vengono riprodotte vocalmente tramite il componente text-to-speech al momento della loro ricezione.

Capitolo 4

Implementazione e manuale utente

Il quarto capitolo parlerà dello sviluppo del chatbot. Inizialmente, si mostrerà come viene allenato il modello e verranno illustrate le configurazioni del framework Rasa per una corretta realizzazione del progetto. Si procederà, successivamente, con la descrizione dei codici Python necessari per lo svolgimento delle azioni personalizzate. In seguito, verrà descritta la realizzazione delle due interfacce front-end previste, quella per applicativi web e quella per smartphone. Infine, si riporterà un manuale utile all'utente per il corretto utilizzo dell'assistente digitale.

4.1 Dati di training

I dati di training, precedentemente descritti nella Sezione 2.3.1, si dividono in tre file principali:

- `nlu.yml`, contenenti esempi di intent da riconoscere;
- `stories.yml`, contenenti esempi di percorsi di conversazione;
- `rules.yml`, contenenti regole che il modello deve rispettare.

Nelle prossime sezioni si descrivono le composizioni di questi file, utilizzati per l'allenamento del modello.

4.1.1 NLU

Gli intent che il modello deve riconoscere, elencati nella Sezione 3.2.1, possono essere divisi in tre categorie principali:

- *frasi generiche*, che non sono associate a nessuna azione personalizzata;
- *frasi per l'attivazione delle form*;
- *frasi per l'attivazione delle azioni personalizzate*.

Il Listato 4.1 mostra la prima parte del file `nlu.yml`; in particolare, si specificano la versione utilizzata per la stesura del documento e l'elenco dei primi intent utili al riconoscimento di saluti, conferme, negazioni e richieste di aiuto.

```
1 version: "3.1"
2 nlu:
3   - intent: saluto
4     examples: |
5       - ciao
6       - buongiorno
7       [...]
8   - intent: addio
9     examples: |
10      - ci vediamo
11      - arrivederci
12      [...]
13  - intent: conferma
14    examples: |
15      - si
16      - esatto
17      [...]
18  - intent: negazione
19    examples: |
20      - no
21      - mai
```

```
22     [...]
23 - intent: aiuto
24   examples: |
25     - quali sono i comandi da utilizzare?
26     - Aiutami a capire cosa fare
27     [...]
28 - intent: conversazione_bot
29   examples: |
30     - tu chi sei?
31     - con chi sto parlando
32     [...]
```

Listato 4.1: Intent generici

Il Listato 4.2 mostra due esempi per ciascuno degli intent associati all'attivazione delle form. Il prefisso 'request' è previsto dalla documentazione di Rasa per gli intent il cui riconoscimento è associato all'avvio di una form.

```
1 - intent: request_paziente
2   examples: |
3     - seleziona un assistito
4     - seleziona assistito
5     [...]
6 - intent: request_unita_operativa
7   examples: |
8     - seleziona unita operativa
9     - seleziona reparto
10    [...]
11 - intent: request_parametro_vitale
12   examples: |
13     - Aggiungi nuovo parametro vitale
14     - Crea nuovo salvataggio di parametri
15     [...]
16 - intent: request_registrazione_diario
17   examples: |
```

```
18     - salva nuova nota nel diario
19     - crea nuova voce nel registro
20     [...]
21 - intent: request_andamento_esami
22 examples: |
23     - voglio sapere l'andamento di un esame
24     - qual e' l'andamento di un esame specifico
25     [...]
```

Listato 4.2: Intent per l'attivazione delle form

In modo analogo, il Listato 4.3 colleziona una coppia di esempi per tutti gli intent utilizzati per attivare le azioni personalizzate. Il nome degli intent si ricollega al nome delle azioni richiamate; il prefisso 'res_' sta ad indicare la natura RESTful delle chiamate svolte nelle azioni personalizzate associate.

```
1 - intent: res_get_uo
2   examples: |
3     - qual e' l'elenco di unita operative presenti?
4     - dimmi tutte le unita operative
5     [...]
6 - intent: res_elenco_ricoverati
7   examples: |
8     - quali sono i ricoverati?
9     - dimmi elenco pazienti
10    [...]
11 - intent: res_parametri_vitali_codificati
12   examples: |
13     - Dimmi tutti i parametri vitali del paziente
14     - Quali sono i parametri del paziente?
15    [...]
16 - intent: res_terapie_assistito
17   examples: |
18     - A quali terapie e' sottoposto il paziente?
19     - Terapie dell'assistito
```



```
20     [...]
21 - intent: res_somministrazione_terapia
22 examples: |
23     - somministrazioni al paziente
24     - farmaci da dare
25     [...]
26 - intent: res_allergie_assistito
27 examples: |
28     - Quali sono le allergie dell'assistito?
29     - A cosa e' allergico il paziente?
30     [...]
31 - intent: res_info_assistito
32 examples: |
33     - Dimmi le informazioni della paziente
34     - Informazioni generali del ricoverato
35     [...]
36 - intent: res_diario_assistito
37 examples: |
38     - Qual e' l'ultima nota sul paziente?
39     - Leggi note del diario
40     [...]
41 - intent: res_avvisi_terapia
42 examples: |
43     - Quali sono gli avvisi terapia?
44     - Dimmi gli avvisi delle terapie nei diversi reparti
45     [...]
46 - intent: res_compara_esami
47 examples: |
48     - Quali sono gli ultimi esami effettuati
49     - Dimmi l'elenco degli ultimi esami svolti dall'assistito
50     [...]
```

Listato 4.3: Intent per l'attivazione delle azioni personalizzate

4.1.2 Regole

All'interno del file `rules.yml` vengono definite ventisette regole per la corretta gestione della conversazione. In fase di definizione è fondamentale che non ci siano due o più regole in conflitto tra loro. Il Listato 4.4 mostra la parte iniziale del file; viene specificata la versione utilizzata e sono inserite le regole per attivare le risposte automatiche definite nel file di dominio; queste ultime verranno analizzate nei prossimi paragrafi. Il file prosegue con ulteriori regole, le più significative delle quali vengono mostrate di seguito.

```
1  version: "3.1"
2
3  rules:
4    - rule: saluta
5      steps:
6        - intent: saluto
7          - action: utter_saluto
8
9    - rule: addio
10     steps:
11       - intent: addio
12         - action: utter_addio
13
14    - rule: aiuto
15     steps:
16       - intent: aiuto
17         - action: utter_aiuto
18
19    - rule: conversazione_bot
20     steps:
21       - intent: conversazione_bot
22       - action: utter_conversazione_bot
```

Listato 4.4: Esempi di regole per le risposte automatiche

Nel Listato 4.5 si riporta un esempio di regola per attivare la form per conoscere l'andamento dei risultati di un esame e la regola per la disattivazione della stessa al termine del ciclo di riempimento dei valori di slot richiesti.

```

1  - rule: Activate andamento_esami_form
2  steps:
3    - intent: request_andamento_esami
4    - action: action_resetSlots
5    - action: andamento_esami_form
6    - active_loop: andamento_esami_form
7
8  - rule: Submit andamento_esami_form
9  condition:
10   - active_loop: andamento_esami_form
11  steps:
12   - action: andamento_esami_form
13   - active_loop: null
14   - slot_was_set:
15     - requested_slot: null
16   - action: utter_submit

```

Listato 4.5: Esempio di regola per attivazione e disattivazione delle form

Per attivare un'azione personalizzata viene utilizzata una regola come quella nel Listato 4.6; nell'esempio mostrato, si può notare che dapprima viene salvato nello slot `latest_intent` il nome dell'azione richiesta dall'utente; successivamente, si "valorizza" lo slot `action_executed` a `false` per indicare che l'azione richiesta dall'utente non è ancora stata svolta dal chatbot; infine, si attiva l'azione `action_confirmation`.

```

1  - rule: allergie
2  steps:
3    - intent: res_allergie_assistito
4    - slot_was_set:
5      - latest_intent: res_allergie_assistito
6    - slot_was_set:
7      - action_executed: false

```

```
- action: action_confirmation
```

Listato 4.6: Esempio di regola per l'attivazione di un'azione personalizzata

4.1.3 Storie

Nel file `stories.yml` vengono forniti esempi di conversazioni che contribuiscono a guidare il modello nella comprensione delle azioni da eseguire in specifici contesti. A differenza delle regole definite precedentemente, le storie sono indicazioni meno vincolanti per l'addestramento del modello. Pur consentendo al modello di sviluppare una maggiore autonomia nella decisione delle risposte, è importante costruire storie ben formulate per evitare l'indecisione del modello e il conseguente aumento degli errori di risposta.

Il Listato 4.7 rappresenta due storie che delineano scenari interattivi tra l'utente e il chatbot. La prima storia, denominata "Elenco uo con conferma", inizia con il riconoscimento dell'intento `res_get_uo`, ossia la richiesta dell'elenco delle unità operative disponibili. Si prosegue con l'esecuzione dell'azione `action_confirmation`. Successivamente, vengono impostati gli slot `latest_intent` e `action_executed`, per memorizzare l'intento dell'utente e specificare che l'azione non è ancora stata eseguita. La storia procede con il rilevamento dell'intento `conferma`, ovvero l'utente conferma che vuole conoscere l'elenco delle unità operative. Dopo la conferma esplicita, si avvia l'azione `action_start`; quest'ultima, in base all'intento richiesto, produrrà la risposta desiderata. Contestualmente, si specifica che l'azione è stata eseguita e si richiama l'azione di default `action_listen` per specificare che il chatbot deve attendere una nuova richiesta dopo aver risposto alla richiesta dell'utente.

La seconda storia, denominata "Elenco uo con negazione", segue un percorso simile, iniziando anch'essa con il riconoscimento dell'intento `res_get_uo`; tuttavia, in questo caso, l'utente non conferma l'esecuzione dell'azione. La conversazione deve, dunque, terminare con il chatbot che chiede di ripetere la richiesta e rimane in ascolto di un nuovo messaggio da parte dell'utente.

```
1 version: "3.1"  
2  
3 stories:
```

```
4 - story: Elenco uo con conferma
5   steps:
6     - intent: res_get_uo
7     - action: action_confirmation
8     - slot_was_set:
9       - latest_intent: res_get_uo
10    - slot_was_set:
11      - action_executed: false
12    - intent: conferma
13    - action: action_start
14    - slot_was_set:
15      - action_executed: true
16    - action: action_listen
17
18 - story: Elenco uo con negazione
19   steps:
20     - intent: res_get_uo
21     - action: action_confirmation
22     - slot_was_set:
23       - latest_intent: res_get_uo
24     - slot_was_set:
25       - action_executed: false
26     - intent: negazione
27     - action: utter_ripeti
```

Listato 4.7: Esempio di storia con la richiesta dell'elenco delle Unità Operative

Vengono descritte storie analoghe a queste per ogni intento che attiva l'azione personalizzata `action_start`; sarà, poi, quest'ultima a produrre la risposta.

Tra le storie sono inseriti anche percorsi generati tramite conversazioni interattive con il chatbot; ne è un esempio il Listato 4.8. Tramite il comando `rasa interactive`, si effettua una conversazione guidata con il bot, in cui è possibile correggere eventuali errori. Il percorso della conversazione svolta viene automaticamente aggiunto nei file di training; ciò permette di allenare il modello su discorsi più lunghi e ramificati.

```
1 - story: interactive_story_1
2   steps:
3     - intent: saluto
4     - action: utter_saluto
5     - intent: request_paziente
6     - action: action_resetSlots
7     - action: assistito_form
8     - active_loop: assistito_form
9     - slot_was_set:
10      - requested_slot: assistito
11     - intent: nome_cognome
12     - slot_was_set:
13      - assistito: DE2200113
14     - action: assistito_form
15     - slot_was_set:
16      - requested_slot: null
17     - active_loop: null
18     - action: utter_submit
19     - intent: request_unita_operativa
20     - action: action_resetSlots
21     - action: unita_operativa_form
22     - active_loop: unita_operativa_form
23     - slot_was_set:
24      - requested_slot: unita_operativa
25     - intent: nome_uo
26     - slot_was_set:
27      - unita_operativa: PSICHIATRIA
28     - action: unita_operativa_form
29     - slot_was_set:
30      - requested_slot: null
31     - active_loop: null
32     - action: utter_submit
33     - intent: res_elenco_ricoverati
```

```
34     - action: action_confirmation
35     - slot_was_set:
36         - latest_intent: res_elenco_ricoverati
37     - slot_was_set:
38         - action_executed: false
39     - intent: conferma
40     - action: action_start
41     - slot_was_set:
42         - action_executed: true
43     - intent: res_parametri_vitali_codificati
44     - action: action_confirmation
45     - slot_was_set:
46         - latest_intent: res_parametri_vitali_codificati
47     - slot_was_set:
48         - action_executed: false
49     - intent: conferma
50     - action: action_start
51     - slot_was_set:
52         - action_executed: true
```

Listato 4.8: Esempio di storia interattiva

4.2 Configurazioni

Si procede, ora, con la descrizione di altri elementi importanti nella realizzazione del chatbot, come il file di dominio, la pipeline per il riconoscimento del testo e la configurazione degli endpoint e delle credenziali per i canali di comunicazione con il bot.

4.2.1 Domain

Nel dominio si raccolgono gli elementi costitutivi del bot. Innanzitutto, si elencano gli intent validi, come mostrato nel Listato 4.9.

```
1 version: "3.1"
```

```
2 intents:
3   - addio
4   - conferma
5   - negazione
6   - aiuto
7   - saluto
8   [...]
9   - request_paziente
10  - request_unita_operativa
11  [...]
12  - res_get_uo
13  - res_info_assistito
14  - res_parametri_vitali_codificati
15  - res_terapie_assistito
16  [...]
17  - res_compara_esami
18  - reset_slots
```

Listato 4.9: Elenco degli intent nel file di dominio

Il Listato 4.10 evidenzia come sono definite le form, ovvero specificando il nome e gli slot richiesti da ciascuna di esse. Per ogni slot si dettagliano la tipologia, l'influenza che lo slot ha nella conversazione e come viene ottenuto il valore; in particolare, nel chatbot realizzato, il mapping avviene in maniera custom, ovvero tramite codice Python delle azioni, oppure tramite il testo del messaggio; in quest'ultimo caso si specifica che il mapping può avvenire solo se è effettivamente in corso la compilazione di un questionario.

```
1 forms:
2   assistito_form:
3     required_slots:
4       - assistito
5     [...]
6   andamento_esami_form:
7     required_slots:
8       - parametro_esame
```



```
9   registrazione_diario_form:
10     required_slots:
11       - nota
12
13 slots:
14   assistito:
15     type: text
16     influence_conversation: true
17     mappings:
18       - type: from_text
19       conditions:
20         - active_loop: assistito_form
21           requested_slot: assistito
22   [...]
23   nota:
24     type: text
25     influence_conversation: true
26     mappings:
27       - type: from_text
28       conditions:
29         - active_loop: registrazione_diario_form
30           requested_slot: nota
31   id_assistito:
32     type: any
33     influence_conversation: false
34     mappings:
35       - type: custom
36   [...]
37   latest_intent:
38     type: text
39     influence_conversation: false
40     mappings:
41       - type: custom
```

```
42  action_executed:
43    type: bool
44    influence_conversation: false
45    mappings:
46      - type: custom
```

Listato 4.10: Form e slot nel file di dominio

Infine, nel file `domain.yml`, si includono: (i) le risposte automatiche, definite con il prefisso `utter_`; (ii) le domande per la richiesta degli slot nei questionari, definite come `utter_ask_<slot>` e, infine, (iii) quelle personalizzate realizzate, comprese le azioni di validazione delle form definite con il prefisso `validate_`. Il Listato 4.11 presenta alcuni esempi delle categorie appena elencate.

```
1  responses:
2    utter_saluto:
3      - text: Ehi! Ben trovato. Come ti posso aiutare?
4    utter_addio:
5      - text: A presto!
6    [...]
7    utter_ripeti:
8      - text: Ripeti il comando per favore
9    utter_ask_assistito:
10     - text: Dimmi il numero di pratica o il cognome del paziente
11    utter_ask_unita_operativa:
12     - text: In che unita' operativa ti trovi?
13    [...]
14    utter_ask_parametro_esame:
15     - text: Vuoi conoscere l'andamento di quale esame?
16    utter_submit:
17     - text: Resto in attesa della prossima richiesta!
18    utter_unita_operativa_values:
19     - text: Unita' operativa {unita_operativa} salvata
20       correttamente!
21    utter_assistito_values:
```

```
21     - text: Assistito salvato correttamente!  
22     [...]  
23     utter_andamento_esami_values:  
24     - text: Ecco l'andamento dei valori di {parametro_esame} negli  
        ultimi esami  
25 actions:  
26     - action_start  
27     - utter_submit  
28     - action_confirmation  
29     - utter_unita_operativa_values  
30     - utter_assistito_values  
31     [...]  
32     - validate_andamento_esami_form  
33     - action_session_start
```

Listato 4.11: Form e slot nel file di dominio

4.2.2 Pipeline

La pipeline di Rasa per la comprensione del testo, visibile nel Listato 4.12, presenta una serie di componenti progettati per affrontare diverse fasi del processo di elaborazione del linguaggio naturale. Inizialmente, si utilizza il modulo `SpacyNLP` con il modello `it_core_news_sm` per eseguire la tokenizzazione e l'analisi grammaticale dei testi in lingua italiana. Successivamente, il tokenizzatore `SpacyTokenizer` segmenta il testo in token, mentre `SpacyFeaturizer` converte i token in vettori di funzionalità basati sulle rappresentazioni di parole `Spacy`.

La pipeline incorpora anche diversi featurizer aggiuntivi, tra cui il `RegexFeaturizer` e il `LexicalSyntacticFeaturizer`, che arricchiscono la rappresentazione delle frasi con informazioni di tipo sintattico e basate su espressioni regolari. L'utilizzo del `CountVectorsFeaturizer` consente la creazione di rappresentazioni vettoriali basate sulla frequenza di parole, sia per le parole stesse che per le caratteristiche di tipo carattere (`char_wb`).

La fase successiva della pipeline è dedicata al training del classificatore con il componente `DIETClassifier`. Questo modello utilizza un approccio di embedding neurale e viene configurato con parametri specifici, tra cui il numero di epoche, la confidenza del modello, la lunghezza della classifica, e la regolarizzazione. In questa configurazione specifica, l'entità di riconoscimento è disabilitata.

Successivamente, la pipeline include il componente `EntitySynonymMapper` per mappare sinonimi delle entità riconosciute, il componente `ResponseSelector` per la selezione delle risposte del chatbot, e il componente `FallbackClassifier`, che agisce come sicurezza di fallback quando la confidenza delle previsioni è inferiore a una soglia specificata. Infine, è presente il componente `RegexEntityExtractor`, per l'estrazione di entità basata su espressioni regolari e lookup table.

L'insieme di questi componenti offre un approccio completo per il riconoscimento del testo all'interno del framework Rasa, integrando modelli di linguaggio, featurizer e classificatori per supportare un'ampia gamma di interazioni e contesti linguistici.

```
1 pipeline:
2   - name: SpacyNLP
3     model: it_core_news_sm
4   - name: SpacyTokenizer
5   - name: SpacyFeaturizer
6   - name: RegexFeaturizer
7   - name: LexicalSyntacticFeaturizer
8   - name: CountVectorsFeaturizer
9   - name: CountVectorsFeaturizer
10  analyzer: "char_wb"
11  min_ngram: 1
12  max_ngram: 4
13  - name: DIETClassifier
14    epochs: 100
15    constrain_similarities: true
16    model_confidence: softmax
17    ranking_length: 10
18    entity_recognition: False
```

```
19     use_masked_language_model: False
20     BILOU_flag: False
21     entity_interpolation: linear
22     regularization_constant: 0.001
23 - name: EntitySynonymMapper
24 - name: ResponseSelector
25     epochs: 100
26     constrain_similarities: true
27     model_confidence: softmax
28 - name: FallbackClassifier
29     threshold: 0.3
30     ambiguity_threshold: 0.1
31 - name: RegexEntityExtractor
32     case_sensitive: False
33     use_lookup_tables: True
34     use_regexes: True
```

Listato 4.12: Pipeline di comprensione del testo

4.2.3 Endpoint e Credenziali

Nel file `endpoints.yml` si specifica l'indirizzo del server delle azioni. Nel caso del progetto realizzato, come mostrato nel Listato 4.13, il server viene eseguito all'interno di un'immagine Docker della quale si specifica l'indirizzo.

```
1 action_endpoint:
2   url: http://app:5055/webhook
```

Listato 4.13: Endpoint server delle azioni

Infine, nel file `connections.yml`, visibile nel Listato 4.14, si specificano le connessioni REST, a cui non occorrono ulteriori parametri al di fuori della dichiarazione, e la connessione al canale SocketIO indicando i parametri che dovranno coincidere con quelli nel front-end utilizzato.

```
1 rest:
```

```
2
3 socketio:
4     bot_message_evt: bot_uttered
5     session_persistence: false
6     user_message_evt: user_uttered
7     metadata_key: metadata
```

Listato 4.14: Connessione REST e SocketIO

4.3 Azioni

Le azioni personalizzate sono state gestite dividendo il codice Python in tre file per aumentare la leggibilità e la manutenibilità del programma. In particolare:

- `actions.py` contiene tutte le classi associate alle azioni definite nel dominio, come le classi per la validazione delle form, le classi di default con override dei metodi, quelle per la richiesta di conferma dell'intent e per l'esecuzione del comportamento richiesto;
- `api.py` viene richiamato nelle classi e contiene tutti i metodi per eseguire le chiamate RESTful ai servizi API aziendali al fine di ottenere i risultati richiesti;
- `tools.py`, dal quale vengono richiamati dei metodi utili in diverse parti di codice per eseguire diverse operazioni, tra le quali pulizia, formattazione e filtraggio degli oggetti inviati e ricevuti.

Inoltre, la maggior parte dei messaggi statici contenuti nel codice sono gestiti come variabili globali contenute in un codice Python esterno, in modo da semplificare la manutenzione, agevolare una possibile traduzione, migliorare la leggibilità, facilitare la gestione e promuovere la riutilizzabilità del codice.

4.3.1 Classi

Si procede con una descrizione delle funzionamento di ciascuna classe contenuta nel file `actions.py`; per le più significative di esse viene anche riportato il codice.

Classe di Avvio della Sessione (ActionSessionStart)

La classe `ActionSessionStart` contiene l'override della funzione omonima, la quale è responsabile dell'inizializzazione della sessione. Se vengono forniti metadati validi (`deviceID`, `userID`, e `token`) durante l'inizio della sessione, vengono impostati con tali valori gli slot appropriati. In caso contrario, viene comunicato che i metadati sono mancanti.

I metadati sono inviati contestualmente ai messaggi; da essi si recuperano l'identificativo del dispositivo, dell'utente e il token ad esso associato. Queste informazioni sono fondamentali per l'autenticazione e l'autorizzazione dell'utente, senza le quali non sarebbe possibile eseguire le operazioni. Il Listato 4.15 mostra il momento del salvataggio dei metadati all'interno degli slot di Rasa.

```
1 if(metadata["deviceID"] and metadata["userID"] and metadata["token
   "]):
2     return [SessionStarted(), ActionExecuted("action_listen"),
3           SlotSet("id_dispositivo", metadata["deviceID"]),
4           SlotSet("id_utente", metadata["userID"]),
5           SlotSet("token", metadata["token"])]
6 else:
7     # Se mancano dati nel metadata, comunica che sono mancanti.
8     dispatcher.utter_message(text="Metadati mancanti")
```

Listato 4.15: Controllo dei metadati

Validazione della Form dell'Unità Operativa

Questa classe gestisce la validazione dell'unità operativa inserita. Essa verifica la presenza e la lunghezza dell'input, quindi cerca l'unità operativa nel contesto degli altri slot. Se l'unità operativa è valida, viene restituito un messaggio di conferma.

Il blocco di Listato 4.16 chiama la funzione `set_gruppo_profilo` per ottenere l'elenco delle unità operative, il gruppo, il profilo e il codice di risposta. Se la risposta non è 200, indicando un fallimento, viene inviato un messaggio di errore con il codice di errore restituito. In caso di successo, viene costruito un messaggio informativo sulla lista delle

unità operative e vengono restituiti i valori rilevanti come parte del risultato, incluso il nome dell'unità operativa, nonché i codici del gruppo e del profilo ad essa associati.

```
1 unita_operativa, gruppo, profilo, gproCode = set_gruppo_profilo(  
    slots, unita_operativa) #si richiama l'elenco delle uo  
2 if gproCode != 200: #response code diverso da 200 significa che  
    non e' andata a buon fine  
3     dispatcher.utter_message(text=f"{g_strings.U0notfound}{  
        g_strings.errorCode}{gproCode}")  
4     return {"unita_operativa": None, "requested_slot": None}  
5 msg = g_strings.utter_unita_operativa_values.format(  
    unita_operativa)  
6 dispatcher.utter_message(text=msg)  
7 return {"unita_operativa": unita_operativa, "assistito": None, "  
    id_gruppo": gruppo, "id_profilo": profilo}
```

Listato 4.16: Ottenimento dei valori relativi all'Unità Operativa selezionata

Validazione della Form del Nome dell'Assistito

La classe `ValidateAssistitoForm` gestisce la validazione del nome dell'assistito. Essa controlla la presenza e la lunghezza dell'input, quindi cerca l'assistito nel contesto degli altri slot. Se il nome dell'assistito è valido, viene restituito un messaggio di conferma.

Validazione della Form per l'Inserimento di un Parametro Vitale

Questa classe gestisce la validazione del parametro vitale e del suo valore. Essa verifica la presenza degli slot necessari e controlla se il parametro da inserire è coerente con l'elenco parametri ammissibili e dei relativi attributi, ottenuti da una chiamata ai servizi REST. Se il valore è all'interno del range definito per la specifica tipologia di parametro, viene effettuata una chiamata POST per il salvataggio.

Il Listato 4.17 verifica se il valore del parametro vitale è compreso nel range; quest'ultimo viene ottenuto dai campi dell'oggetto JSON che descrive il parametro. Se il valore è al di fuori del range, viene inviato un messaggio di errore tramite il dispatcher.

Altrimenti, se il valore è valido, viene costruito un corpo JSON per la chiamata POST e, successivamente, viene effettuata la richiesta di salvataggio attraverso la funzione `post_call`, descritta più avanti. In caso di successo, viene inviato un messaggio di conferma e vengono visualizzati i dettagli del parametro vitale.

```
1  #si verifica che il valore sia all'interno del range
2  if valore_parametro <= valoreMin or valore_parametro >= valoreMax:
3      #se fuori soglia allora messaggio di errore
4      dispatcher.utter_message(text=g_strings.outRangeValue)
5      return {"parametro_vitale": None, "valore_parametro": None}
6  #altrimenti si effettua la chiamata POST per il salvataggio
7  else:
8      body = {
9          [...]
10         "parametri": [
11             {
12                 "idParametro": str(idParametro),
13                 "um": str(umParametro),
14                 "valore": str(valore_parametro),
15                 "idTipoCombo": str(idTipoCombo),
16                 "idValore": "",
17             }
18         ],
19     }
20     body = json.dumps(body)
21     response = post_call(g_strings.registra_parametri_vitali, body
22         , slots[6])
23     [...]
```

Listato 4.17: Parte di codice per l'inserimento di un nuovo parametro vitale

Validazione della Form della Registrazione Diario

La classe `ValidateRegistrazioneDiarioForm` gestisce la validazione della nota inserita durante la registrazione del diario. Essa verifica la presenza degli slot necessari ed effettua una chiamata POST per salvare la nota.

Validazione della Form dell'Andamento degli Esami

Questa classe gestisce la validazione del parametro dell'esame durante la registrazione dell'andamento degli esami. Essa verifica la presenza degli slot necessari e restituisce un messaggio di conferma con i risultati degli esami.

Classe di Reset degli Slot

La classe `ActionResetSlots` resetta gli slot salvati prima della compilazione di un nuovo form per garantire che non vengano salvati nuovamente dati precedentemente inseriti. Essa può resettare specifici slot o tutti gli slot, a seconda dell'intent ricevuto. Il codice corrispondente viene illustrato nel Listato 4.18.

```
1 if intent == "request_andamento_esami":
2     return [
3         SlotSet("latest_intent", intent),
4         SlotSet("action_executed", False),
5         SlotSet("parametro_esame", None),
6     ]
7
8     [...]
9
10 elif intent == "reset_slots":
11     dispatcher.utter_message(text=f"{g_strings.deletedData}")
12     return [AllSlotsReset()]
13 else:
14     return
```

Listato 4.18: Azzeramento degli slot

Classe di Conferma dell'Azione (ActionConfirmation)

Questa classe gestisce la conferma delle azioni richieste dall'utente. Essa ripete all'utente i parametri coinvolti e chiede l'autorizzazione a eseguire l'azione. Il Listato 4.19 delinea come avviene la richiesta di approvazione.

```
1  if intent in g_strings.intent_azioni: #l'azione deve essere una
    tra quelle valide
2      #mostro all'utente su quali parametri viene applicata l'azione
3      dispatcher.utter_message(text=f"{g_strings.intentscheck}{
        intentConverter(intent)} ")
4      if assistito != None:
5          dispatcher.utter_message(text=f"Paziente: {assistito}")
6      if (unita_operativa) != None:
7          dispatcher.utter_message(text=f"Unita operativa: {
            unita_operativa}")
8          dispatcher.utter_message(text=f"{g_strings.confirmation}")
9  else:
10     dispatcher.utter_message(text=f"{g_strings.intentnotfoud}")
11
12  return [
13     SlotSet("latest_intent", intent), #salvo l'intent negli slots
        per riprenderlo dopo il messaggio di conferma
14     SlotSet("action_executed", False), #mi indica che sono in
        attesa di eseguire l'azione
15  ]
```

Listato 4.19: Richiesta di conferma dell'azione da eseguire

Classe di Esecuzione dell'Azione (ActionStart)

La classe `ActionStart` esegue la funzione associata a ciascun intent dell'utente. Essa verifica la presenza degli slot necessari ed esegue la funzione corrispondente all'intent richiesto.

Il Listato 4.20 gestisce diverse intenzioni in una conversazione. In base all'intent rilevato dal modello si eseguono i metodi relativi alle chiamate REST associate. Se l'intenzione non corrisponde a nessuna azione programmata, si invia un messaggio di errore al fine di far ripetere correttamente il comando. Se la chiamata ha ottenuto risposta positiva, si inviano i messaggi di risposta all'utente, fino ad un massimo di 10 messaggi.

```
1 if intent == g_strings.diario_assistito:
2     if check_slots():
3         code, response = res_diario_assistito(slots)
4     [...]
5 elif intent == g_strings.compara_esami:
6     if check_slots():
7         code, response = res_compara_esami(slots)
8 else:
9     dispatcher.utter_message(text=f"{g_strings.intentnotfoud}")
10    code = 404
11    return [SlotSet("action_executed", True)]
12 if code == 200:
13    msgOut = jsonToText(response)
14    for i, msg in enumerate(msgOut):
15        if i < 10: dispatcher.utter_message(text=msg)
16        else: break
17 elif code == 204:
18    dispatcher.utter_message(text=f"{g_strings.noResponse}")
19 else:
20    dispatcher.utter_message(text=f"{g_strings.errorCode}{str(code)}" )
21 return [SlotSet("action_executed", True)]
```

Listato 4.20: Esecuzione delle chiamate REST associate agli intent

4.3.2 Chiamate RESTful

I metodi richiamati dalla classe `ActionStart` in seguito alla conferma esplicita dell'utente, sono contenuti nel file `api.py` nella medesima cartella del file `actions.py`. Questi metodi, seppur ciascuno di essi richiami diversi endpoint REST, hanno tutti la medesima struttura, analoga a quella mostrata nel Listato 4.21; si costruisce il body necessario per le chiamate POST, si ottiene la response dalla chiamata e, infine, si naviga il JSON di risposta estraendo le informazioni utili per costruire la risposta da inviare all'utente.

```
1 def res_compara_esami(slots):
2     body = {
3         "idDispositivo": slots[4],
4         "utente": slots[4],
5         "pratica": slots[2],
6         "idAssistito": slots[3]
7     }
8     body = json.dumps(body) # dati in json
9     response = post_call(g_strings.compara_esami, body, slots[6])
10    code = response.status_code
11    if code != 200:
12        msgOut = [f"Errore compara_esami"]
13        return code, msgOut
14    else:
15        response = response.json()
16        esami = response["elenco"]
17        keywords = [
18            "data",
19            "esame",
20            "risultato",
21            "udm",
22            "inrange"
23        ]
24        max_date = datetime.min #inizializzo la data piu' recente
           come la minima possibile
```

```
25     for esame in esami: #ciclo sull'elenco degli esami
26         data_str = esame.get('data') #verifico data
27         if data_str:
28             # Converti la data in oggetto datetime
29             data_obj = datetime.strptime(data_str, "%d/%m/%Y %
30                                     H:%M")
31
32             # Confronta con la data massima attuale
33             if data_obj > max_date: #se la data e' piu recente
34                 di quella attuale
35                 max_date = data_obj #salvo la nuova data
36                 recente
37
38             #seleziono solo gli esami effettuati piu' recentemente
39             ultimi_esami = [esame for esame in esami if esame.get('
40                 data') and datetime.strptime(esame['data'], "%d/%m/%Y %
41                 H:%M") == max_date]
42
43             jsonOut = select_fields(ultimi_esami, keywords)
44             return code, jsonOut
```

Listato 4.21: Metodo per l'ottenimento degli ultimi esami svolti dal paziente

4.3.3 Strumenti

All'interno del codice vengono impiegati alcuni metodi strumento, i quali semplificano varie operazioni quali il filtraggio, la pulizia e la formattazione dei dati JSON, e non solo. Tra questi, il metodo più significativo è `post_call` (Listato 4.22), il quale assume un ruolo di rilievo poiché effettua le chiamate effettive alle API aziendali e restituisce la risposta. È da questa risposta che vengono estratti i messaggi destinati all'utente.

```
1 def post_call(route, body, token):
2     # Costruisci l'URL completo unendo il percorso 'route' con l'
3     URL di base.
4     url = "<url-API-aziendali>" + route
```

```
4      # Imposta l'intestazione (header) con il token di
      autorizzazione.
5      headers = {"Authorization": f"Bearer {token}"}
6      # Stampa l'URL, l'intestazione e il corpo della richiesta (per
      debug).
7      print("\n", url, "\n", headers, "\n", body)
8      # Effettua una richiesta HTTP POST all'URL specificato con l'
      intestazione e il corpo dati.
9      response = requests.post(url, headers=headers, data=body)
10     return response # Restituisce la risposta ottenuta dalla
      richiesta POST.
```

Listato 4.22: Funzione per chiamate POST alle API aziendali

4.4 Front-end

4.4.1 Realizzazione front-end web

Il front-end sviluppato per le applicazioni web ha l'obiettivo di evidenziare una potenziale integrazione con gli applicativi pre-esistenti nell'ambito aziendale in cui è stato effettuato il tirocinio. Parallelamente, svolge una funzione utile per le attività di testing in sostituzione dell'applicazione mobile. Nonostante la rilevanza strategica di questo front-end, è fondamentale notare che, al momento, costituisce soltanto un'opzione secondaria rispetto all'applicazione mobile. Pertanto, la sua realizzazione rappresenta più una dimostrazione tecnica che un'implementazione operativa.

L'interfaccia è stata realizzata con codice HTML, CSS e JavaScript. Di seguito, il Listato 4.23 presenta le funzioni principali per la gestione dell'interfaccia.

Il codice stabilisce la connessione con il server socket su cui è in esecuzione il modello del chatbot. L'interfaccia utente è composta da elementi come il campo di input per i messaggi, un'icona per attivare/disattivare la sintesi vocale e un'area per visualizzare i messaggi. La funzione `toggleVoice` gestisce il cambio di stato della sintesi vocale e visualizza le icone corrispondenti.

Il codice implementa anche la funzione `getSessionId`, che recupera o genera un identificativo di sessione utilizzando il socket ID e lo memorizza in locale. La funzione `utter` invia i messaggi dell'utente al server Rasa tramite il socket. La funzione `appendMessage` aggiunge dinamicamente i messaggi inviati o ricevuti all'interfaccia utente, gestendo anche la sintesi vocale per i messaggi ricevuti, se abilitata.

La parte finale del codice gestisce gli eventi di invio del modulo, connessione al server SocketIO, errori di connessione e la ricezione dei messaggi dal chatbot Rasa. In particolare, quando il chatbot emette un messaggio, il codice lo visualizza dinamicamente nell'interfaccia utente, supportando anche immagini e risposte automatiche.

```
1  const socket = io("<url-canale-socket>");
2  const messages = document.getElementById("messages");
3  const form = document.getElementById("form");
4  const messageInput = document.getElementById("message-input");
5
6  let voiceEnabled = true;
7  const iconVolumeOn = document.getElementById("icon-volume-on");
8  const iconVolumeOff = document.getElementById("icon-volume-off");
9
10 function toggleVoice() {
11   if (voiceEnabled) {
12     voiceEnabled = false;
13     iconVolumeOn.style.display = "none";
14     iconVolumeOff.style.display = "block";
15   } else {
16     if ("speechSynthesis" in window) {
17       voiceEnabled = true;
18       iconVolumeOn.style.display = "block";
19       iconVolumeOff.style.display = "none";
20     } else {
21       alert("Sorry, your browser doesn't support text to speech.");
22       ;
23     }
24   }
25 }
```



```
24 }
25 iconVolumeOn.addEventListener("click", toggleVoice);
26 iconVolumeOff.addEventListener("click", toggleVoice);
27
28 [...]
29
30 function getSessionId() {
31     const storage = localStorage;
32     const storageKey = <RASA_SESSION_ID>;
33     const savedId = storage.getItem(storageKey);
34     if (savedId) {
35         return savedId;
36     }
37     const newId = socket.id;
38     storage.setItem(storageKey, newId);
39     return newId;
40 }
41
42 function utter(msg) {
43     socket.emit("user_uttered", {
44         message: msg,
45         session_id: getSessionId(),
46         metadata: {
47             userID: getUserId(),
48             deviceID: getDeviceId(),
49             token: getUserToken(),
50         },
51     });
52 }
53
54 function appendMessage(msg, type) {
55     const item = document.createElement("div");
56     msg = msg.replace(/\n/g, " <br> ");
```

```
57     item.innerHTML = msg;
58     item.classList.add("message");
59     item.classList.add('message_${type}');
60     messages.appendChild(item);
61     scrollToBottom();
62
63     if (voiceEnabled && type === "received") {
64         const voiceMsg = new SpeechSynthesisUtterance();
65         // Impostare la voce e la lingua desiderate
66         voiceMsg.voice = speechSynthesis.getVoices().filter(function (
67             voice) {
68                 return voice.name === "Microsoft Elsa - Italian (Italy)";
69             })[0];
69         voiceMsg.lang = "it-IT";
70         voiceMsg.text = msg;
71         window.speechSynthesis.speak(voiceMsg);
72     }
73 }
74
75 [...]
76
77 function stopSpeaking() {
78     speechSynthesis.cancel();
79 }
80
81 form.addEventListener("submit", function (e) {
82     e.preventDefault();
83     const msg = messageInput.value;
84     if (msg) {
85         utter(msg);
86         messageInput.value = "";
87         appendMessage(msg, "sent");
88     }
```

```
89 });
90
91 socket.on("connect", function () {
92     console.log("Connected to Socket.io server.");
93     socket.emit("session_request", {
94         session_id: getSessionId(),
95     });
96     console.log('Session ID: ${getSessionId()}');
97 });
98
99 socket.on("connect_error", (error) => {
100     // Write any connection errors to the console
101     console.error(error);
102 });
103
104 socket.on("bot_uttered", function (response) {
105     console.log("Bot uttered:", response);
106     if (response.text) {
107         appendMessage(response.text, "received");
108     }
109     if (response.attachment) {
110         appendImage(response.attachment.payload.src, "received");
111     }
112     if (response.quick_replies) {
113         appendQuickReplies(response.quick_replies);
114     }
115 });
```

Listato 4.23: Codice JavaScript per l'interfaccia Web

4.4.2 Realizzazione front-end smartphone

Come spiegato nella Sezione 3.2.2, si è deciso di integrare il chatbot all'interno dell'applicazione "Smart4Hospital", realizzata dall'azienda NBS s.r.l. Per la realizzazione è

stato scelto il framework Ionic, consentendo di realizzazione una web app ibrida e cross-platform, utilizzando i linguaggi HTML e SCSS, per il front-end, e, TypeScript, per il back-end dell'applicazione.

Di seguito, un riassunto delle sue funzionalità chiave:

- *Connessione socket*: il componente si connette a un server SocketIO per interagire con il backend del chatbot.
- *Inizializzazione*: durante la connessione, viene inizializzato il session ID e vengono gestiti eventuali errori di connessione.
- *Riconoscimento Vocale*: il codice integra il riconoscimento vocale tramite il plugin Capacitor Speech Recognition, consentendo agli utenti di interagire anche vocalmente.
- *Text-to-Speech*: questo componente utilizza il plugin Capacitor Text-to-Speech per leggere i messaggi ricevuti in forma vocale.
- *Ricezione Messaggi*: i messaggi ricevuti vengono gestiti tramite la creazione dinamica di elementi nell'interfaccia
- *Gestione Utterance*: i messaggi ricevuti sono memorizzati in un array per essere letti tramite text-to-speech; l'utente può attivare o disattivare questa funzionalità.
- *Invio Messaggi al Backend*: i messaggi dell'utente vengono inviati al bot attraverso il socket.
- *Controllo del Flusso*: viene gestito il flusso della conversazione, mantenendo la lista dei messaggi visualizzati e scorrendo automaticamente la schermata verso il basso quando nuovi messaggi vengono aggiunti.

Il codice utilizza in modo efficace i plugin di Ionic e le funzionalità di Angular per gestire l'interazione con il backend del chatbot, nonché l'integrazione di funzionalità vocali.

Il metodo `startRecognition()`, mostrato nel Listato 4.24, gestisce l'avvio asincrono del riconoscimento vocale. Inizialmente, verifica la disponibilità del servizio e controlla

se il riconoscimento è abilitato e se non si sta già leggendo un messaggio. Se tutte le condizioni sono soddisfatte, avvia il riconoscimento vocale con le impostazioni specificate, acquisisce il primo risultato rilevato e lo invia come messaggio. Il processo continua in modo ricorsivo se il riconoscimento è ancora abilitato. Eventuali errori durante l'esecuzione sono gestiti e registrati sulla console.

```
1  async startRecognition() {
2      try {
3          if (this.available && this.recognition && !this.isReading)
4              {
5                  SpeechRecognition.start({
6                      popup: false,
7                      partialResults: false,
8                      language: "it-IT",
9                  }).then(
10                     (success: any) => {
11                         this.msg = success.matches[0];
12                         this.changeDetectorRef.detectChanges();
13                         this.sendMsg();
14                         if (this.recognition) {
15                             this.startRecognition();
16                         }
17                     },
18                     (error: any) => {
19                         console.log(error);
20                         if (this.recognition) {
21                             this.startRecognition();
22                         }
23                     }
24                 );
25             }
26         } catch (error) {
27             console.error("Errore durante l'inizio della registrazione
28                 :", error);
29         }
```

```
27     }
28 }
```

Listato 4.24: Metodo che gestisce il riconoscimento del parlato

Il Listato 4.25 mostra il metodo che viene richiamato al momento della pressione del pulsante 'Invio'; dopo aver controllato che sia stato effettivamente scritto un messaggio, si definisce un'interfaccia `Message` che conterrà il testo e la tipologia del messaggio, 'sent' per i messaggi inviati e 'received' per le risposte dal chatbot; a questo punto, il metodo `appendMsg` crea un nuovo box con il messaggio inviato, mentre `utterMsg` invia il testo al bot tramite socket.

```
1  sendMsg() {
2      if (this.msg.length > 0) {
3          this.message = {
4              text: this.msg,
5              type: "sent",
6          };
7          this.appendMsg(this.message);
8          this.utterMsg(this.message.text);
9          this.msg = "";
10     }
11 }
```

Listato 4.25: Metodo che gestisce l'invio dei messaggi

Il metodo `speakText()`, mostrato nel Listato 4.26, è responsabile della sintesi vocale asincrona dei messaggi ricevuti. Se il buffer dei messaggi ricevuti non è vuoto, ossia se ci sono messaggi da leggere, e la funzione di sintesi vocale è abilitata, interrompe eventuali operazioni di riconoscimento vocale in corso, preleva il primo messaggio dal buffer, avvia la lettura vocale utilizzando il modulo `TextToSpeech` con le impostazioni specificate, attende il completamento della lettura, quindi procede in modo ricorsivo fino a quando ci sono altri messaggi da leggere nel buffer. Infine, avvia nuovamente il riconoscimento vocale. È necessario interrompere il riconoscimento vocale durante la

lettura dei messaggi al fine di evitare che le risposte ricevute attualmente in fase di lettura vengano erroneamente interpretate come nuovi messaggi da trascrivere.

```
1  async speakText() {
2      if (this.receivedBuffer.length > 0 && this.speech == true) {
3          SpeechRecognition.stop();
4          const msg = this.receivedBuffer.shift();
5          this.isReading = true;
6          await TextToSpeech.speak({
7              text: msg,
8              lang: "it-IT",
9              rate: 1.5,
10             });
11             this.isReading = false;
12             this.speakText();
13         }
14         this.startRecognition();
15     }
```

Listato 4.26: Metodo che gestisce il text-to-speech

4.5 Manuale utente

Al termine dell'illustrazione del codice sviluppato, si forniscono alcune linee guida per garantire un utilizzo efficace del chatbot e minimizzare potenziali errori.

Inizialmente, è cruciale comprendere che molte azioni non sono accessibili fino a quando non vengono fornite le informazioni necessarie per l'esecuzione. Pertanto, è obbligatorio specificare l'unità operativa in cui si opera prima di selezionare un paziente particolare e, analogamente, è necessario identificare il paziente prima di richiedere o inviare informazioni relative ad esso.

In aggiunta, sia nel caso di messaggi vocali che di testo, è fondamentale esprimersi con chiarezza e evitare l'uso di termini ambigui. Nel caso di messaggi vocali, si consiglia particolarmente di prestare attenzione alla pronuncia corretta delle parole per agevolare

la fase di conversione da speech-to-text. L'utilizzo di cuffie con microfono integrato è fortemente consigliato, sia per migliorare la qualità dell'audio che per semplificare l'interazione complessiva con il chatbot.

Sebbene il chatbot sia in grado di classificare anche frasi con variazioni, gli esempi forniti nella Tabella 4.1 costituiscono utili suggerimenti per agevolare la conversazione e prevenire la necessità di ripetere comandi poco chiari. Di seguito, invece, viene riportato un esempio pratico di una possibile conversazione con il chatbot nella quale si effettuano operazioni di lettura e scrittura con i dati di un paziente.

Utente: Ciao!

Assistente: Ehi! Ben trovato. Come ti posso aiutare?

U: Seleziona unità operativa

A: In che unità operativa ti trovi?

U: Psichiatria

A: Unità operativa Psichiatria salvata correttamente!

A: Resto in attesa della prossima richiesta!

U: Seleziona assistito

A: Qual è il cognome del paziente?

U: De2200113

A: Qual è il nome del paziente?

U: Mo200113

A: Cognome e nome di De2200113 Mo200113 sono stati salvati correttamente!

A: Resto in attesa della prossima richiesta!

U: Quali sono i parametri del paziente?

A: Desideri conoscere l'elenco dei parametri vitali

A: Paziente: De2200113 Mo200113

A: Unità operativa: Psichiatria

A: Confermi?

U: Sì

A: PESO; 78; Kg; Fuori scala N; 25/08/2023 18:41;

A: SODIO; 150; mmol/L; Fuori scala S; 25/08/2023 18:40;

A: TC; 38; °C; Fuori scala S; 25/08/2023 12:45;
A: PESO; 90; Kg; Fuori scala N; 25/08/2023 12:37;
A: SODIO; 728; mmol/L; Fuori scala S; 25/08/2023 11:55;
A: SODIO; 464; mmol/L; Fuori scala S; 24/08/2023 17:23;
A: SODIO; 464; mmol/L; Fuori scala S; 24/08/2023 16:43;
A: SODIO; 127; mmol/L; Fuori scala S; 08/08/2023 10:36;
A: SODIO; 546; mmol/L; Fuori scala S; 08/08/2023 10:20;
A: SODIO; 786; mmol/L; Fuori scala S; 08/08/2023 10:19;
U: Quali sono le allergie dell'assistito?
A: Desideri conoscere le allergie dell'assistito
A: Paziente: De2200113 Mo200113
A: Unità operativa: Psichiatria
A: Confermi?
U: Sì
A: Fattore Farmaci; Descrizione PARACETAMOLO/CODEINA
FOSFATO EMIIDRATO; Genere Anamnestica;
U: salva nuovo parametro vitale
A: Quale parametro vitale vuoi inserire?
U: Temperatura
A: Qual è il valore del parametro?
U: 37.6
A: Valore conforme alle soglie.
A: Il parametro Temperatura è stato salvato correttamente
con valore 37.6!
A: Resto in attesa della prossima richiesta!
U: Seleziona nuovo assistito
A: Qual è il cognome del paziente?
U: COS96572
A: Qual è il nome del paziente?
U: Vi96572
A: Cognome e nome di COS96572 Vi96572 sono stati salvati

correttamente!

A: Resto in attesa della prossima richiesta!

U: Salva nuova nota

A: Qual è la nota che vuoi registrare?

U: Esami nella norma, nessun riscontro negativo.

A: La nota "Esami nella norma, nessun riscontro negativo."
è stato salvata correttamente!

A: Resto in attesa della prossima richiesta!

Intento	Descrizione	Esempio attivazione
Saluto	Messaggio di saluto verso l'assistente	"Buongiorno!"
Conferma	Messaggio di conferma	"Sì"
Negazione	Messaggio di negazione	"No"
Aiuto	Richiesta di aiuto sul funzionamento dell'assistente	"Aiutami"
Richiesta U.O.	Selezione di una nuova unità operativa	"Seleziona unità operativa"
Richiesta paziente	Selezione di un paziente	"Seleziona assistito"
Registrazione parametro vitale	Salvataggio di un nuovo parametro vitale	"Salva nuovo parametro vitale"
Registrazione nota	Salvataggio di una nuova nota	"Salva nuova nota nel diario"
Elenco U.O.	Richiesta dell'elenco di tutte le Unità Operative disponibili	"Dimmi l'elenco delle unità operative"
Elenco avvisi terapia	Elenco degli avvisi di mancate somministrazioni terapie	"Quali sono gli avvisi terapia?"
Elenco ricoverati	Richiesta dell'elenco dei ricoverati	"Dimmi l'elenco dei ricoverati"
Parametri vitali assistito	Lista dei parametri vitale dell'assistito selezionato	"Voglio conoscere i parametri del paziente"
Terapie assistito	Tutte le terapie dell'assistito	"Terapie dell'assistito"
Somministrazioni assistito	Somministrazioni da effettuare all'assistito selezionato	"Quali farmaci devo somministrare al paziente?"
Allergie assistito	Lista delle allergie	"A cosa è allergico il paziente?"
Diario assistito	Ultima nota registrata nel diario dell'assistito	"Qual è l'ultima nota sul paziente?"
Informazioni assistito	Informazioni sull'assistito	"Dimmi le informazioni del ricoverato"

Tabella 4.1: Frasi di esempio per l'attivazione delle funzioni

Capitolo 5

Discussione e analisi dei risultati

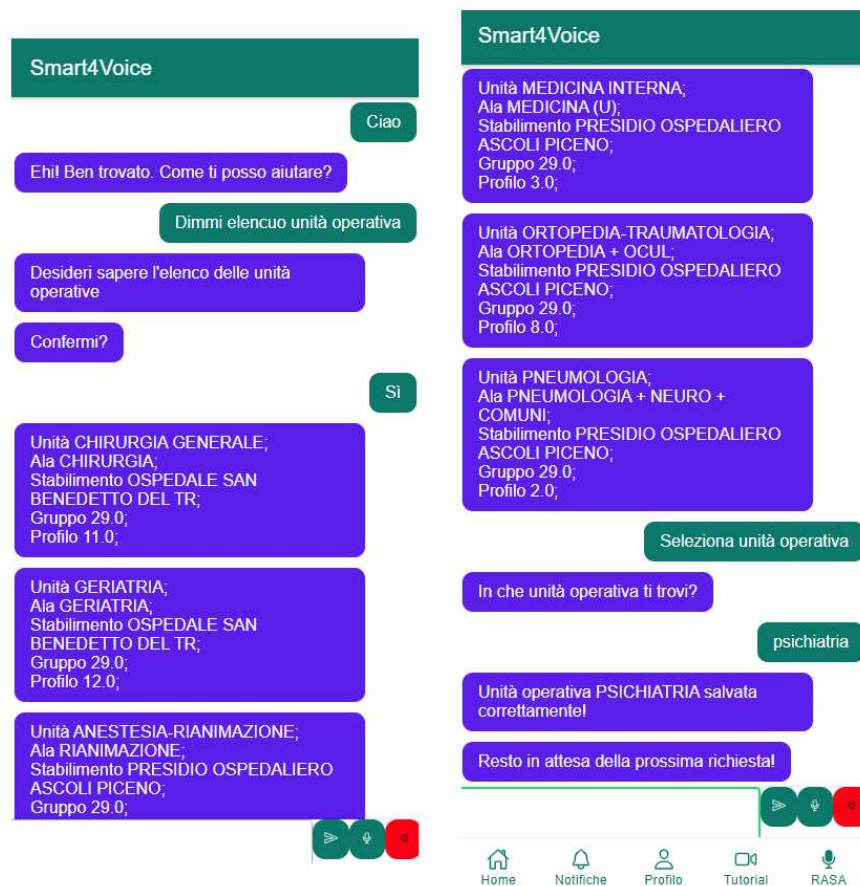
Nel quinto capitolo si riassumeranno le funzionalità del chatbot mostrando alcuni esempi di conversazione nell'applicazione. Si proseguirà analizzando i risultati ottenuti e valutando le performance del chatbot e della trascrizione vocale. Infine, si presenteranno alcune alternative già disponibili sul mercato per valutarne punti di similitudine e differenze al fine di fornire una prospettiva completa sullo stato attuale delle soluzioni disponibili.

5.1 Riepilogo delle funzionalità

Per riepilogare le funzionalità del chatbot e fornire esempi tangibili di conversazioni, vengono presentati screenshot della schermata implementata nell'applicazione.

Nella Figura 5.1(a) si delinea un potenziale inizio di conversazione, in cui, in seguito a un saluto, si chiede l'elenco delle Unità Operative disponibili. La conversazione prosegue con la Figura 5.1(b) in cui viene illustrata la concreta selezione di un'Unità Operativa, e poi, ancora, nella Figura 5.2(a), la scelta di un paziente all'interno di tale unità. È importante notare che, nel caso in cui si tenti di selezionare un paziente o un'Unità Operativa non disponibile, verrà visualizzato un messaggio di errore, invitando l'utente a ripetere il comando, come nel caso della Figura 5.2(b).

La Figura 5.3(a) illustra la procedura di richiesta e attivazione esplicita di un'azione personalizzata. In particolare, l'esempio presentato riguarda la richiesta delle terapie da somministrare all'utente. Tuttavia, il meccanismo di richiesta è analogo per altre funzionalità, come l'acquisizione delle informazioni sulle allergie, dei dettagli personali



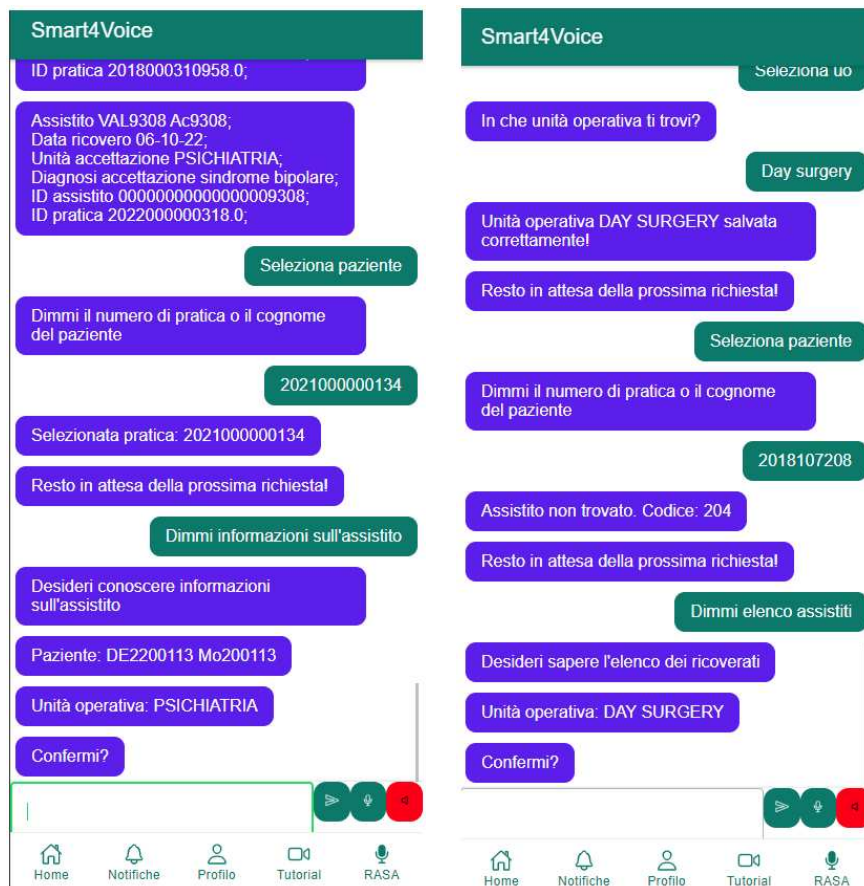
(a) *Lista delle Unità Operative* (b) *Selezione di un'Unità Operativa*

Figura 5.1: Esempi di conversazioni iniziali con il chatbot

dell'assistito, dei parametri vitali, delle annotazioni nel diario clinico e del numero di terapie da somministrare nei reparti.

Analogamente alle selezioni, le richieste di letture potrebbero non generare risultati nel caso in cui non siano disponibili valori da riportare. In tali situazioni, verrà visualizzato un messaggio apposito, come evidenziato nella Figura 5.3(b).

La Figura 5.4(a) illustra il corretto inserimento di nuovi dati attraverso una form, evidenziando il processo di registrazione sul diario clinico. La conversazione segue un modello sostanzialmente analogo sia per la richiesta di registrazione di un nuovo parametro vitale sia per la richieste dell'andamento dei valori di uno specifico esame.



(a) Selezione di un assistito

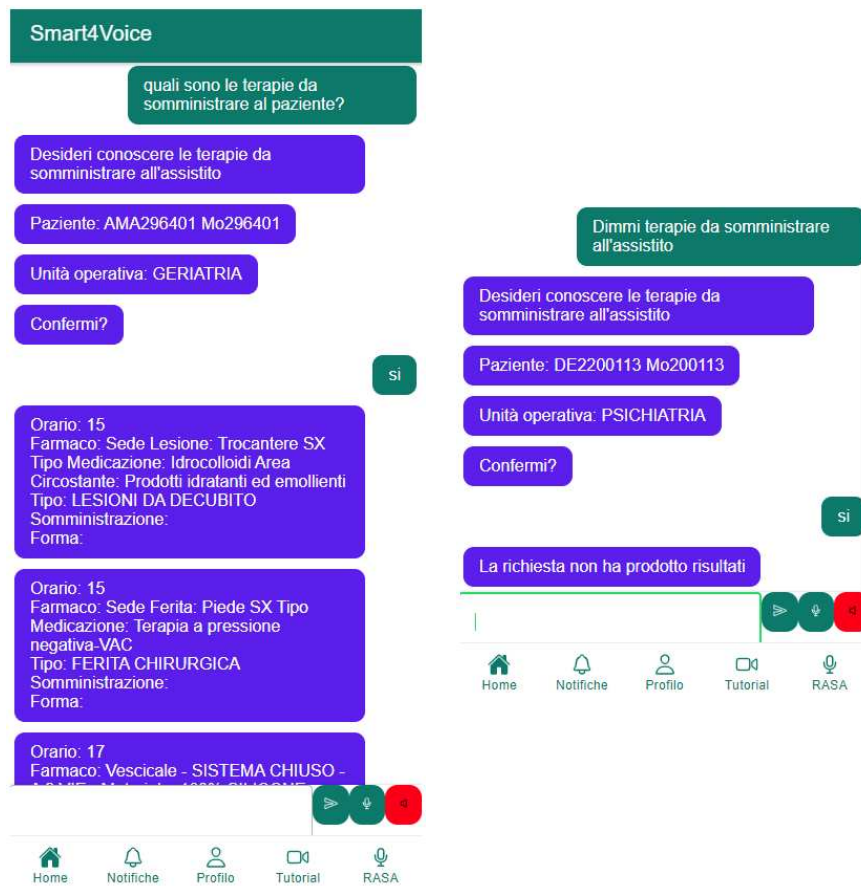
(b) Assistito richiesto non trovato

Figura 5.2: Esempi di conversazioni iniziali con il chatbot

Nel caso in cui si verificano errori di compilazione dei campi, il sistema di validazione identificherà tali errori e richiederà all'utente di ripetere l'inserimento. Un esempio di questa interazione è mostrato nella Figura 5.4(b), dove si evidenzia la gestione di un inserimento di un valore di un parametro vitale al di fuori del range ammissibile.

5.2 Analisi dei risultati

All'interno di questa sezione, ci si dedica all'approfondita valutazione delle performance del modello di chatbot, esaminando con attenzione le capacità di interazione e risposta



(a) *Richiesta delle terapie da somministrare* (b) *Richiesta senza risultati*

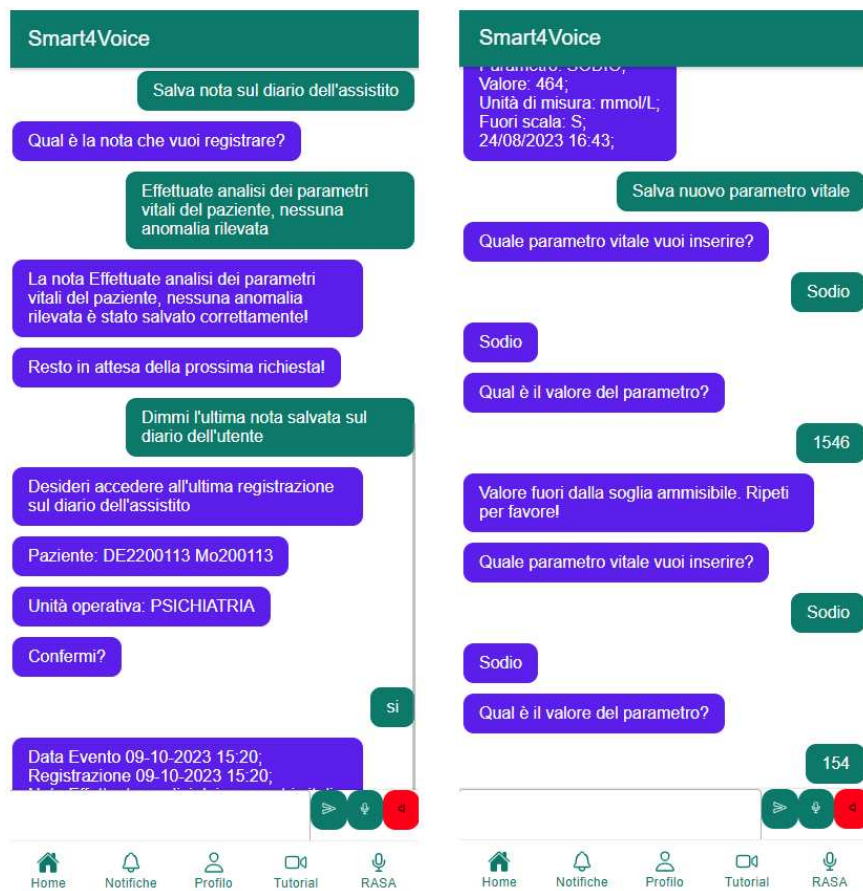
Figura 5.3: Esempi di funzionalità del chatbot

del sistema. Successivamente, si conduce un'analisi sulla trascrizione vocale.

L'obiettivo è fornire un quadro completo delle prestazioni globali del chatbot, esaminando sia l'efficacia dell'interazione testuale che la precisione nella trascrizione vocale, contribuendo, così, a una valutazione integrata e dettagliata dell'esperienza utente.

5.2.1 Valutazioni delle performance del chatbot

Nel corso dei test del modello Rasa, è stata condotta una valutazione accurata utilizzando la cross-validation con diverse configurazioni di fold, ovvero di ripetizioni. Questa



(a) *Salvataggio di una nuova sul diario* (b) *Validazione fallita per valore inammissibile*

Figura 5.4: Esempi di funzionalità del chatbot

procedura è stata eseguita con fold impostati a 5, 10 e 15, consentendo una panoramica dettagliata delle performance del modello in varie condizioni. Per ottenere questi risultati è stato utilizzato il comando `rasa test nlu` messo a disposizione dal framework.

Risultati con Cross-Validation e fold impostato a 5

L'intent evaluation viene eseguita considerando 264 esempi. I risultati ottenuti sono i seguenti:

- Train Accuracy: 99.6% (deviazione standard 0.2%)

- Train F1-score: 99.8% (deviazione standard 0.1%)
- Train Precision: 100% (deviazione standard 0.0%)
- Test Accuracy: 75.8% (deviazione standard 3.4%)
- Test F1-score: 73.3% (deviazione standard 3.8%)
- Test Precision: 76.3% (deviazione standard 5.4%)

I risultati indicano che il modello raggiunge una precisione estremamente alta durante il training, con una capacità quasi perfetta di identificare correttamente gli intenti specifici. Tuttavia, durante il test, si osserva una diminuzione della precisione, che potrebbe suggerire una certa complessità nell'adattamento a dati non visti.

Risultati con Cross-Validation e fold impostato a 10

In questo caso, gli esempi presi in considerazione sono 176. I risultati ottenuti sono i seguenti:

- Train Accuracy: 99.4% (deviazione standard 0.2%)
- Train F1-score: 99.7% (deviazione standard 0.1%)
- Train Precision: 100% (deviazione standard 0.0%)
- Test Accuracy: 82.9% (deviazione standard 7.2%)
- Test F1-score: 81.4% (deviazione standard 7.8%)
- Test Precision: 83.0% (deviazione standard 9.3%)

Anche in questa configurazione, il modello mantiene una precisione eccezionale durante il training, dimostrando la sua affidabilità nell'apprendimento dai dati. La performance nel test mostra una migliore generalizzazione rispetto alla configurazione precedente, suggerendo una maggiore capacità del modello di adattarsi a dati non osservati.

Particolare attenzione è rivolta a questa specifica configurazione, arricchendo l'analisi con informazioni supplementari quali la matrice di confusione degli intenti e un report

più dettagliato, al fine di ottenere una comprensione più approfondita delle dinamiche e delle prestazioni del sistema in esame.

La Figura 5.5 mostra la *confusion matrix* degli intent; come preannunciato dalle metriche ottenute, i risultati si possono considerare buoni, ma è comunque interessante evidenziare alcuni comportamenti del modello.

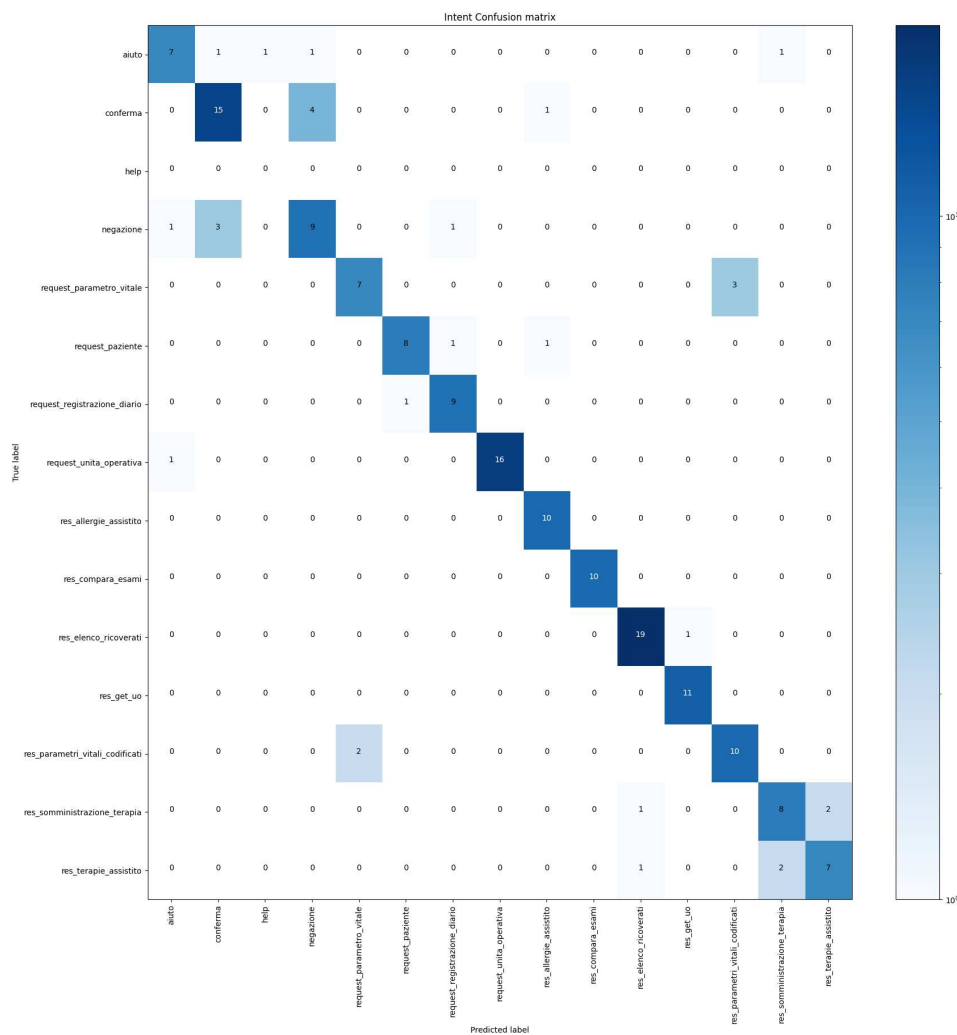


Figura 5.5: Confusion Matrix degli Intent

Innanzitutto, è rilevante considerare che una parte degli errori di interpretazione deriva da richieste semanticamente simili ad altre, creando confusione nel riconoscimento dell'intento dell'utente. Questo fenomeno è intrinseco alla complessità del linguaggio naturale e può comportare situazioni in cui il chatbot interpreta erroneamente l'intento dell'utente, nonostante il meccanismo di conferma. Ad esempio, il chatbot potrebbe confondere le richieste di lettura dei parametri vitali con quelle di aggiunta di un nuovo parametro vitale. Similmente, le richieste di quali terapie somministrare al paziente in un dato momento potrebbero essere semanticamente simili alle richieste relative alle terapie a cui è stato sottoposto da quando è ricoverato o anche al numero di terapie da somministrare nei diversi reparti. Questo fenomeno è osservabile anche dal report degli intent, parzialmente riportato nel Codice 5.1.

```
1 [...]
2   "negazione": {
3     "precision": 0.6428571428571429,
4     "recall": 0.6428571428571429,
5     "f1-score": 0.6428571428571429,
6     "support": 14,
7     "confused_with": {
8       "conferma": 3,
9       "request_registrazione_diario": 1
10    }
11  },
12  "res_terapie_assistito": {
13    "precision": 0.7777777777777778,
14    "recall": 0.7,
15    "f1-score": 0.7368421052631577,
16    "support": 10,
17    "confused_with": {
18      "res_somministrazione_terapia": 2,
19      "res_elenco_ricoverati": 1
20    }
21  },
```

```
22 "request_parametro_vitale": {
23   "precision": 0.7777777777777778,
24   "recall": 0.7,
25   "f1-score": 0.7368421052631577,
26   "support": 10,
27   "confused_with": {
28     "res_parametri_vitali_codificati": 3
29   }
30 },
31 "res_parametri_vitali_codificati": {
32   "precision": 0.7692307692307693,
33   "recall": 0.8333333333333334,
34   "f1-score": 0.8,
35   "support": 12,
36   "confused_with": {
37     "request_parametro_vitale": 2
38   }
39 },
40 "conferma": {
41   "precision": 0.7894736842105263,
42   "recall": 0.75,
43   "f1-score": 0.7692307692307692,
44   "support": 20,
45   "confused_with": {
46     "negazione": 4,
47     "res_allergie_assistito": 1
48   }
49 },
50 [...]
```

Listato 5.1: Report degli intent

Alcuni errori di discernimento tra *'conferma'* e *'negazione'*, sebbene possano sembrare semanticamente distanti, sono più chiaramente compresi attraverso la consultazione

dell'elenco degli intent errati indicato nel Codice 5.2. Da questo emerge che l'uso di espressioni come "è vero" e "non è vero", oppure "assolutamente sì" e "assolutamente no", o ancora "è corretto" e "non è corretto", costituiscono sfide interpretative per il modello. Questi casi rendono difficile per il modello cogliere con precisione le sfumature nelle frasi, rischiando di non captare in modo accurato l'intento comunicativo. Si raccomanda, pertanto, l'utilizzo di conferme o negazioni esplicite come "sì" o "no", al fine di migliorare la chiarezza e la comprensione delle risposte da parte del chatbot.

```
1  [...]
2  {
3      "text": "e' corretto",
4      "intent": "conferma",
5      "intent_prediction": {
6          "name": "negazione",
7          "confidence": 0.8591197729110718
8      }
9  },
10 {
11     "text": "assolutamente no",
12     "intent": "negazione",
13     "intent_prediction": {
14         "name": "conferma",
15         "confidence": 0.9510931372642517
16     }
17 },
18 {
19     "text": "e' vero",
20     "intent": "conferma",
21     "intent_prediction": {
22         "name": "negazione",
23         "confidence": 0.9816632270812988
24     }
25 }
26 [...]
```

Listato 5.2: Alcuni errori di riconoscimento degli intent

La Figura 5.6 mostra l'istogramma di distribuzione della confidenza delle predizioni degli intent. Si può osservare che, quando il chatbot predice correttamente l'intent dell'utente, si osserva che la confidenza associata è mediamente molto elevata, con un picco che raggiunge il 97% di certezza. Questo risultato è coerente con una buona capacità del modello di fornire risposte accurate e sicure in situazioni in cui l'intento è correttamente identificato. Tuttavia, vale la pena notare che, anche in questi casi di corretta previsione, si riscontrano alcuni casi in cui la confidenza scende al di sotto del 90%, o addirittura sotto l'80%, seppure questi casi siano di numero limitato.

Nei casi in cui la predizione del chatbot è errata, la distribuzione delle percentuali di confidenza è più varia. Si osserva una vasta gamma di valori, che variano principalmente tra il 26% e il 100%, con un picco al 97% di confidenza. Questa variazione nella distribuzione della confidenza indica che, in situazioni di errore di predizione, il modello può manifestare una gamma più ampia di livelli di certezza. Risulta evidente che la confidenza può oscillare significativamente nei casi in cui l'intento non è correttamente identificato. Questa analisi della distribuzione della confidenza offre un'importante prospettiva sulla stabilità e la coerenza del modello nelle diverse situazioni di interazione. Da notare la differenza del numero di esempi tra gli intent predetti correttamente ed erroneamente; al livello di confidenza del 97%, che in entrambi i casi corrisponde al picco, si hanno oltre 100 intenti correttamente predetti contro i soli 6 errati.

Risultati con Cross-Validation e fold impostato a 15

In questo caso è stato possibile utilizzare soltanto 57 esempi; i valori delle metriche ottenuti sono i seguenti:

- Train Accuracy: 100% (deviazione standard 0.0%)
- Train F1-score: 100% (deviazione standard 0.0%)
- Train Precision: 100% (deviazione standard 0.0%)
- Test Accuracy: 96.7% (deviazione standard 8.5%)

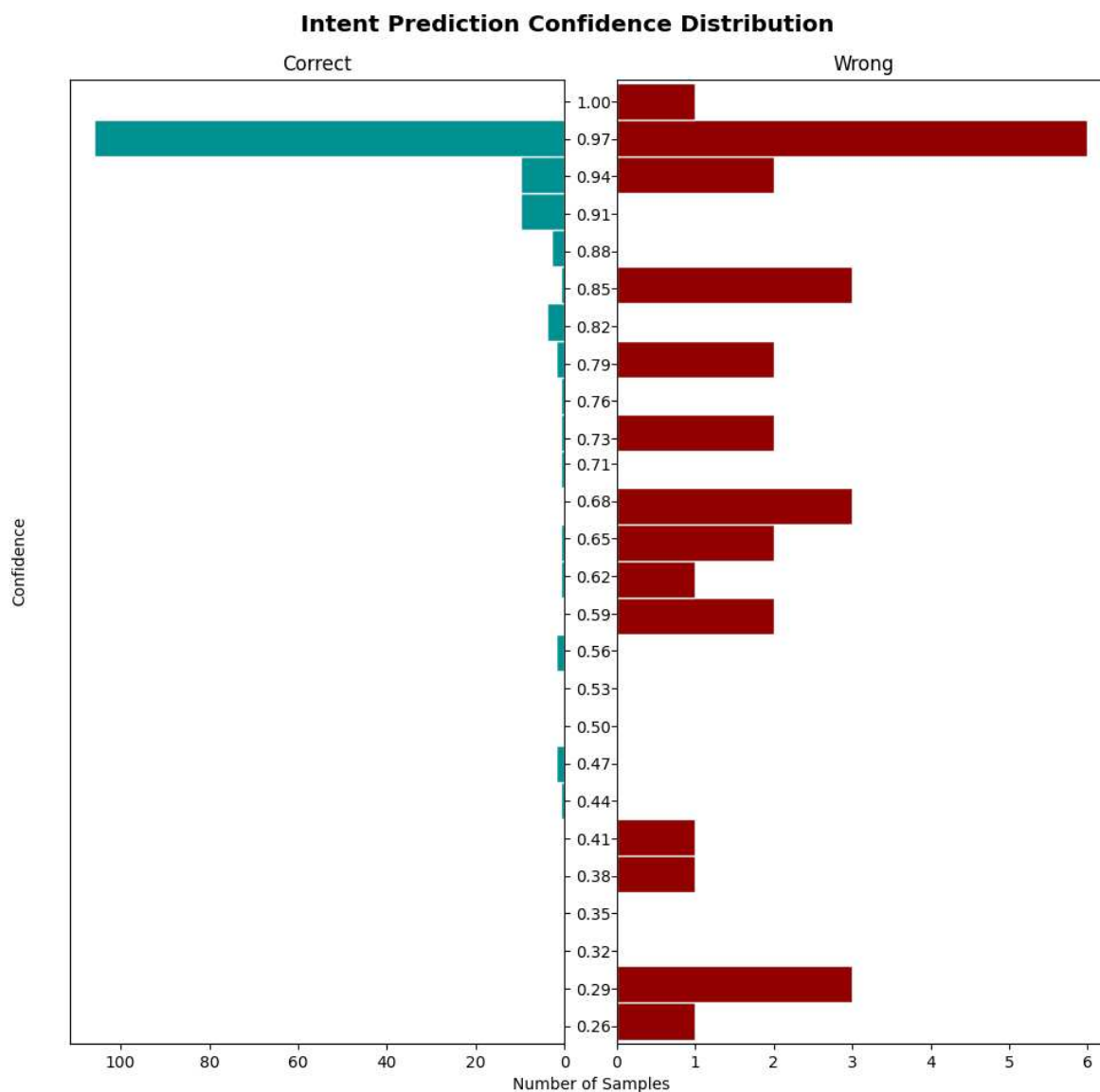


Figura 5.6: Istogramma di distribuzione della confidenza delle predizioni

- Test F1-score: 95.6% (deviazione standard 11.3%)
- Test Precision: 95.0% (deviazione standard 12.7%)

In questa configurazione con 15 fold, il modello raggiunge la perfezione durante il training, dimostrando una sorprendente abilità nell'adattamento a molteplici ripetizio-

ni. La performance di test rimane elevata, indicando una notevole generalizzazione e robustezza.

Alla luce dei risultati ottenuti, si riportano alcune considerazioni:

- *L'alta precisione di training* in tutte le configurazioni suggerisce che il modello acquisisce in modo accurato i pattern presenti nei dati di addestramento.
- La *stabilità delle performance* al variare del numero di fold indica una buona capacità di generalizzazione del modello.
- Il modello mostra *buona adattabilità* anche con un numero limitato di esempi per alcuni intenti durante la cross-validation a 15 fold, evidenziando la sua flessibilità.
- Nonostante le deviazioni standard, i risultati indicano una *bassa varianza* nei punteggi, sottolineando una robusta performance del modello su diverse ripetizioni.

Complessivamente, i risultati confermano che il modello Rasa si rivela efficace nell'identificare intenti, mantenendo notevole precisione e robustezza. La sua abilità di generalizzazione è promettente, suggerendo un'applicabilità affidabile in scenari reali.

È importante sottolineare che l'implementazione del meccanismo di richiesta esplicita di conferma prima dell'esecuzione di un'azione da parte del chatbot si traduce in un notevole aumento della sicurezza e della precisione del sistema. In quasi tutti i casi, questo approccio previene efficacemente l'esecuzione accidentale di azioni indesiderate, fornendo all'utente l'opportunità di confermare o correggere l'intento prima della conclusione dell'operazione.

5.2.2 Valutazioni delle performance della trascrizione vocale

Nell'ambito dell'analisi delle performance della trascrizione vocale è rilevante sottolineare che non è stato sviluppato un modello personalizzato, bensì la scelta è ricaduta su strumenti ampiamente utilizzati sul mercato, quali il WebKit SpeechRecognition per i browser e il modulo SpeechRecognition di Capacitor per l'applicazione mobile. Questa scelta progettuale ha introdotto delle sfide nel processo di valutazione, poiché la mancanza di un modello personalizzato ha complicato la quantificazione di parametri

oggettivi, quali l'accuratezza o l'F1-score. L'attenzione si è quindi concentrata sull'esecuzione di prove sperimentali, coinvolgendo attivamente professionisti medici insieme a utenti comuni.

L'analisi dei risultati, condotta in svariate condizioni, ha rivelato un quadro complessivamente positivo, ma non privo di sfide. In particolare, si sono evidenziate difficoltà nel riconoscimento di termini tecnici medici e nella gestione dei nomi di persone. La comprensione di contesti medici richiede una precisione e una sensibilità particolari, aspetti che saranno oggetto di attenzione per future ottimizzazioni.

Va, altresì, notato che la qualità dell'esperienza di trascrizione è fortemente influenzata da fattori esterni. L'utilizzo di un microfono di alta qualità e l'operatività in ambienti silenziosi emergono come elementi cruciali per garantire una trascrizione accurata. Inoltre, si è riscontrato che cadenze e accenti troppo pronunciati possono compromettere la precisione del riconoscimento vocale; ciò rappresenta una sfida rilevante nella progettazione di sistemi destinati all'ambito medico.

Non meno importante è sottolineare che un'eventuale imprecisione nella trascrizione non conduce automaticamente a un'azione errata del chatbot. Nella maggior parte dei casi, il sistema è in grado di interpretare correttamente l'intento dell'utente, anche in presenza di eventuali errori lessicali, confermando la resilienza del chatbot di fronte a variazioni nella precisione della trascrizione.

5.3 Confronto con lo stato dell'arte

Il chatbot oggetto di questo lavoro di tesi offre la possibilità di effettuare diverse operazioni, tra cui la lettura e la scrittura di parametri vitali, l'accesso al diario clinico, la visualizzazione dell'andamento degli esami e la consultazione delle terapie da somministrare, tutto mediante l'interazione tramite tastiera o comandi vocali. Al fine di comprendere appieno il suo posizionamento nel contesto degli assistenti virtuali nel settore sanitario, è opportuno confrontarlo con alcune soluzioni di rilievo presenti sul mercato.

Nel panorama attuale, come descritto nella Sezione 1.3.2, l'impiego di chatbot nell'assistenza sanitaria è già consolidato, con esempi come *OneRemission*, focalizzato sull'educazione riguardo al cancro, *Youper*, dedicato alla salute emotiva, e *Florence*, un assistente

virtuale per i pazienti. Tali chatbot sono progettati per fornire informazioni dettagliate sulla salute, offrire diagnosi preliminari e suggerire trattamenti in base ai sintomi dei pazienti. L'utilizzo di chatbot in ambito sanitario presenta numerosi vantaggi, incoraggiando il processo decisionale informato, promuovendo l'esercizio fisico, e supportando terapie specifiche.

In confronto a paradigmi consolidati, emerge una netta differenziazione nel fine e nel target di utenza dell'assistente digitale realizzato. La maggior parte dei chatbot esistenti è concepita per assistere i pazienti o i cittadini in generale, fornendo informazioni utili per la gestione della propria salute o semplificando interazioni comuni con le strutture sanitarie, come la prenotazione di appuntamenti. Diversamente, l'assistente digitale sviluppato mira a essere un ausilio per i professionisti del settore, focalizzandosi sulla facilitazione e l'ottimizzazione delle attività di accesso e scrittura di informazioni relative ai pazienti nei vari reparti.

L'interfaccia vocale, sia in ingresso che in uscita, colloca l'assistente in una comparazione diretta con altri strumenti digitali medici, come i refertatori vocali. In particolare, le funzioni correlate al diario clinico del paziente riflettono, almeno in parte, le caratteristiche di refertazione, ma con alcune distinzioni significative. Mentre i software per la refertazione vocale sono progettati con l'obiettivo di fornire ai medici uno strumento per la compilazione rapida di referti, anche potenzialmente lunghi e complessi, la scrittura sul diario clinico tramite il chatbot sviluppato rappresenta un mezzo per inserire brevi note sullo stato di salute del paziente. Inoltre, è rilevante precisare che le soluzioni di refertazione vocale sul mercato, tra cui Nuance Dragon Medical One, un leader di settore, si basano su modelli di riconoscimento vocale ampiamente addestrati su dizionari medici e non solo; a dimostrazione di ciò basti considerare che Nuance è proprio l'azienda la quale ha fornito il modello di riconoscimento vocale per i primi sviluppi di Apple Siri³⁰. Per queste ragioni, tale obiettivo non costituisce l'essenza della soluzione implementata in questa sede.

In conclusione, il chatbot presenta un focus specifico sulla gestione dei dati clinici, offrendo funzionalità mirate per i professionisti del settore. Sebbene non presenti elementi di totale innovazione rispetto al mercato, l'obiettivo non era quello di introdurre nuove funzionalità, ma di offrire un nuovo strumento per eseguire efficacemente le stesse già

presenti in un'applicazione gestionale per i professionisti. L'introduzione della possibilità di utilizzare esclusivamente la voce per le interazioni mira a semplificare l'esecuzione di tali azioni, specialmente in situazioni in cui l'operatore potrebbe avere le mani impegnate nelle cure dell'assistito.

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi abbiamo illustrato la progettazione e l'implementazione di un chatbot in tecnologia RASA per la gestione dei dati dei pazienti in una struttura sanitaria.

Si è iniziato con uno studio approfondito del mondo dei chatbot, analizzandone la storia e i fondamenti teorici, nonché le diverse tipologie; si sono studiati le attuali applicazioni e lo stato dell'arte, in particolare nel mondo della medicina.

L'attenzione si è poi concentrata sul framework Rasa, scelto come base per lo sviluppo dell'assistente digitale, analizzandone l'architettura e i concetti chiave alla base del suo funzionamento.

Dunque, si è proceduto alla fase di progettazione, dove sono stati raccolti i requisiti fondamentali, sia funzionali che non funzionali, in preparazione allo sviluppo.

L'effettiva implementazione ha avuto inizio con l'addestramento e la configurazione dettagliata del modello. Contemporaneamente, sono state sviluppate le azioni personalizzate per la lettura e la registrazione di dati nella Cartella Clinica Elettronica dei pazienti. In seguito, si sono realizzati gli applicativi web e mobile, destinati a ospitare l'interfaccia d'uso del chatbot.

Al termine dello sviluppo sono stati analizzati i risultati, riportando metriche soddisfacenti in termini di performance del modello del chatbot e un discreto successo anche nelle prove pratiche effettuate con i trascrittori vocali.

Nel confronto con gli strumenti di mercato, sono stati messi in luce gli elementi di novità introdotti dell'assistente realizzato, il quale fornisce al medico un approccio

attualmente inesplorato per accedere ai dati della cartella clinica elettronica.

Al tempo stesso, si evincono anche alcuni prevedibili margini di miglioramento rispetto ad altri strumenti consolidati. In primo luogo, per migliorare ulteriormente l'accessibilità vocale, si propone l'integrazione di modelli di riconoscimento vocale ampiamente addestrati, specificamente orientati al vocabolario medico.

Parallelamente, si possono continuare ad espandere le funzionalità del chatbot, aumentando le possibilità di risposta alle richieste puntuali dei medici, ma anche le capacità; ad esempio si potrebbero integrare modelli di predizione basati sui dati sanitari del paziente al fine di fornire supporto attivo al medico nei momenti di diagnosi. Lo sviluppo e l'ottimizzazione costanti del modello sono fondamentali per renderlo sempre più efficace e robusto nel riconoscere le azioni da svolgere.

In conclusione, questo progetto costituisce un significativo esempio di come le tecnologie emergenti possano rivestire un ruolo fondamentale nel contesto sanitario. L'obiettivo di creare un sistema efficiente, basato sull'Intelligenza Artificiale, che sia facilmente accessibile senza la necessità di effettuare digitazioni da tastiera e con un impatto formativo ridotto può ritenersi raggiunto. La futura diffusione nell'ambito sanitario sarà determinata dagli sviluppi successivi.

Riferimenti

Bibliografia

- [1] Alan M Turing. *Computing machinery and intelligence*. Springer, 2009.
- [2] John R Searle. «Minds, brains, and programs». In: *Behavioral and brain sciences* 3.3 (1980), pp. 417–424.
- [3] Joseph Weizenbaum. «ELIZA—a computer program for the study of natural language communication between man and machine». In: *Communications of the ACM* 9.1 (1966), pp. 36–45.
- [4] Kenneth Mark Colby. «Modeling a paranoid mind». In: *Behavioral and Brain Sciences* 4.4 (1981), pp. 515–534.
- [5] Racter. *The Policeman’s Beard Is Half Constructed*. Warner Books, Inc., 1984.
- [6] Ashish Vaswani et al. «Attention is all you need». In: *Advances in neural information processing systems* 30 (2017).
- [7] Jacob Devlin et al. «Bert: Pre-training of deep bidirectional transformers for language understanding». In: *arXiv preprint arXiv:1810.04805* (2018).
- [8] Alec Radford et al. «Improving language understanding by generative pre-training». In: (2018).
- [9] Eleni Adamopoulou e Lefteris Moussiades. «Chatbots: History, technology, and applications». In: *Machine Learning with Applications* 2 (2020), p. 100006.
- [10] Francesco Colace et al. «Chatbot for e-learning: A case of study». In: *International Journal of Mechanical Engineering and Robotics Research* 7.5 (2018), pp. 528–533.

- [11] Ho Thao Hien et al. «Intelligent assistants in higher-education environments: the FIT-EBot, a chatbot for administrative and learning support». In: *Proceedings of the 9th International Symposium on Information and Communication Technology*. 2018, pp. 69–76.
- [12] Nuria Haristiani. «Artificial Intelligence (AI) chatbot as language learning medium: An inquiry». In: *Journal of Physics: Conference Series*. Vol. 1387. 1. IOP Publishing. 2019, p. 012020.
- [13] Jiyou Jia. «NLML—a Markup Language to Describe the Unlimited English Grammar». In: *arXiv preprint cs/0404018* (2004).
- [14] Tim C Lueth et al. «KANTRA-human-machine interaction for intelligent robots using natural language». In: *Proceedings of 1994 3rd IEEE international workshop on robot and human communication*. IEEE. 1994, pp. 106–111.
- [15] Stanislaio Lauria et al. «Training personal robots using natural language instruction». In: *IEEE Intelligent systems* 16.5 (2001), pp. 38–45.
- [16] Francisco J Chiyah Garcia et al. «Explain yourself: A natural language interface for scrutable autonomous robots». In: *arXiv preprint arXiv:1803.02088* (2018).
- [17] Stanislaio Lauria et al. «Mobile robot programming using natural language». In: *Robotics and Autonomous Systems* 38.3-4 (2002), pp. 171–181.
- [18] Daniela Conti et al. «A comparison of kindergarten storytelling by human and humanoid robot with different social behavior». In: *Proceedings of the companion of the 2017 ACM/IEEE international conference on human-robot interaction*. 2017, pp. 97–98.
- [19] Wing-Chee So et al. «Robot-based intervention may reduce delay in the production of intransitive gestures in Chinese-speaking preschoolers with autism spectrum disorder». In: *Molecular autism* 9.1 (2018), pp. 1–16.
- [20] Paul Vogt et al. «Child-robot interactions for second language tutoring to preschool children». In: *Frontiers in human neuroscience* 11 (2017), p. 73.

- [21] Sayanti Roy et al. «Using human reinforcement learning models to improve robots as teachers». In: *Companion of the 2018 acm/ieee international conference on human-robot interaction*. 2018, pp. 225–226.
- [22] Adam Palanica et al. «Physicians’ perceptions of chatbots in health care: cross-sectional web-based survey». In: *Journal of medical Internet research* 21.4 (2019), e12887.
- [23] Aditya Nrusimha Vaidyam et al. «Chatbots and conversational agents in mental health: a review of the psychiatric landscape». In: *The Canadian Journal of Psychiatry* 64.7 (2019), pp. 456–464.
- [24] M Senthamil Selvil, Mrs S Jansi Rani e Mr C Ranjeeth Kumar. «Speech Recognition in an E-Health Report». In: *Mathematical Statistician and Engineering Applications* 71.3 (2022), pp. 1461–1467.
- [25] Gangmin Li et al. «Building Electronic Health Record using Voice Recognition and Big Data Techniques». In: *2nd Symposium on Health and Education 2019 (SOHE 2019)*. Atlantis Press. 2019, pp. 131–135.
- [26] Vidisha Bhatt, Juan Li e Bikesh Maharjan. «DocPal: a voice-based EHR assistant for health practitioners». In: *2020 IEEE International Conference on E-health Networking, Application & Services (HEALTHCOM)*. IEEE. 2021, pp. 1–6.
- [27] John P Avendano et al. «Interfacing with the electronic health record (EHR): a comparative review of modes of documentation». In: *Cureus* 14.6 (2022).
- [28] Yaa A Kumah-Crystal et al. «Electronic health record interactions through voice: a review». In: *Applied clinical informatics* 9.03 (2018), pp. 541–552.
- [29] Suzanne V Blackley et al. «Speech recognition for clinical documentation from 1990 to 2018: a systematic review». In: *Journal of the american medical informatics association* 26.4 (2019), pp. 324–338.

Sitografia

- [30] Kevin Bostic. *Nuance confirms its voice technology is behind Apple's Siri*. Mag. 2013. URL: <https://appleinsider.com/articles/13/05/30/nuance-confirms-its-technology-is-behind-apples-siri>.
- [31] Fabio Casciabanca. *Chatbot: I senior Li Snobbano ma i giovani li amano di più (Grazie a chatgpt)*. Mag. 2023. URL: <https://www.ninjamarketing.it/2023/05/18/chatbot-numeri-chatgpt/>.
- [32] Bertalan Mesko. *The top 10 healthcare chatbots*. Ago. 2023. URL: <https://medicalfuturist.com/top-10-health-chatbots/>.
- [33] Ott. 2023. URL: <https://rasa.com/>.
- [34] Nov. 2023. URL: <https://rasa.com/docs/rasa/connectors/your-own-website/#chat-widget>.