UNIVERSITÀ POLITECNICA DELLE MARCHE

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea Magistrale

Progettazione e sviluppo di un sistema di gestione di interventi su segnalazioni di errori e malfunzionamenti

Design and development of a management system to take action on error and malfunction alerts

Relatore
Chiar.mo Prof. Aldo Franco Dragoni

Laureando
Luca Esposto

a.a. 2019/20

Alla mia famiglia e ai miei amici,
sostegno in questo lungo viaggio.

Se la conoscenza può creare dei problemi,
non è tramite l'ignoranza che possiamo risolverli.
Isaac Asimov

Se qualcosa può andare storto, lo farà
(nel momento peggiore possibile)
Legge di Murphy

Niente è facile come sembra
Primo corollario alla legge di Murphy

Tutto richiede più tempo di quanto si pensi
Secondo corollario alla legge di Murphy

Ringraziamenti

Il primo ringraziamento va sicuramente ad Alessandro, dalla cui proposta è nato tutto lo sviluppo di questo progetto; con lui ringrazio anche tutti gli altri membri della Tecnodata srl che ho avuto il piacere di conoscere, per avermi accolto e messo a mio agio e per il supporto, sempre presente nonostante il periodo "più che difficile" causa emergenza sanitaria. Ringrazio il professor Dragoni per i saggi consigli, non limitati solo allo svolgimento del tirocinio e della tesi. Ringrazio anche tutto il team dell'AIRT-Lab per la disponibilità a risolvere ogni mio dubbio. Infine il mio ultimo ringraziamento, ma anche il più sentito, va sicuramente alla mia famiglia. A Silvia, mia sorella, per la sua presenza incondizionata e perché mi scuote dalla mia pigrizia e mi ricorda col suo esempio che l'impegno è sempre fondamentale, in ogni ambito. A mia madre e mio padre, perché è grazie alla loro fatica che ho il privilegio di poter studiare quello che mi piace senza rinunciare a nulla di importante e soprattutto per l'incondizionata fiducia e pazienza.

Ancona, Dicembre 2020

Luca Esposto

Sommario

Il servizio di assistenza tecnica per gestire e monitorare le infrastrutture IT rappresenta un ambito ancora fortemente caratterizzato dall'intervento umano, dove l'automazione fatica ad inserirsi. In questo contesto, per andare incontro alle esigenze di efficientamento del lavoro, questo progetto si pone come obiettivo la progettazione e lo sviluppo di un assistente digitale in grado di sostituire l'assistente tecnico umano. Le funzionalità progettate a questo scopo riguardano la ricezione e l'elaborazione dei dati riguardanti segnalazioni di errori e malfunzionamenti provenienti da sistemi di monitoring, l'elaborazione di soluzioni e la presentazione all'utente del servizio di un'interfaccia utente user-friendly in grado di guidarlo verso la risoluzione dei problemi segnalati senza la necessità di contattare l'assistenza tecnica umana. A tale scopo il progetto definisce una serie di modelli e fa uso delle più recenti tecnologie nel campo del NLP e dell'interazione uomo-macchina, testuale e vocale.

Technical assistance service to manage and monitor IT infrastructures represents an area still characterized by human intervention, where automation struggles to fit. In this context, in order to meet the needs of work efficiency, this project aims to design and develop a digital assistant capable of replacing the human technical assistant. Functions designed for this purpose concern reception and processing of errors and malfunctions reports data from monitoring systems, work out solutions and making a user-friendly interface available to to service user, able to guide them towards problem solving of reported problems without the need to contact human technical assistance. To this end, the project defines a series of models and makes use of the latest NLP and HCI technologies, including both voice and text interaction.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Capitolo 1

Introduzione

1.1 Terminologia

L'esposizione di un progetto informatico incorre nell'uso di numerosi termini tecnici, spesso in lingua inglese per convenzione, a volte propri di uno specifico ambito di applicazione; tali termini possono presentare ambiguità di interpretazione da un ambito all'altro. Per queste motivazioni e per andare incontro ai lettori occasionali si invita a fare riferimento all'appendice A, dove si tenta di eliminare le ambiguità e si illustrano i riferimenti ai nomi utilizzati.

1.2 Motivazioni

Questo progetto, d'ora in poi denominato TELLnet, nasce da una necessità reale dell'azienda "Tecnodata srl"[1], comune a molte (se non a tutte) le società di servizi informatici (vedi par. 2.2): le soluzioni offerte a terze parti nel campo dei sistemi informativi necessitano di gestione e monitoraggio dell'infrastruttura IT implementata e di assistenza all'insorgere delle inevitabili problematiche ricorrenti ad essa correlate. Tale assistenza si concretizza in genere nell'interazione, tipicamente telefonica, con uno o più assistenti umani esperti (in genere sistemisti) per l'individuazione delle cause e la ricerca e attuazione di soluzioni.
Scopo di TELLnet è offrire un'alternativa **user-friendly** e allo stesso tempo completa e sicura a detta interazione, **sostituendo l'assistente umano con uno digitale**, al fine di ridurre drasticamente il tempo speso dagli addetti all'assistenza (limitandolo agli eventuali casi non risolti dall'assistente digitale).

1.3 Scenario

Allo stato attuale il servizio assistenza tipico dell'azienda (Tecnodata) si svolge come segue:

1. Il software di monitoring, tipicamente Nagios (vedi appendice B, paragrafo B.1.1) rileva un problema

[1] per maggiori informazioni: https://www.tecnodatasrl.com

2. Il software di monitoring notifica tramite email e/o sms ad entrambe le parti l'insorgenza di un errore/malfunzionamento

3. Il comparto IT dell'azienda cliente contatta l'assistenza (o viceversa) per concordare le operazioni di recovery (cosa fare e quando)

4. L'assistenza attua le operazioni nei tempi e modi concordati e ne verifica l'efficacia, altrimenti si torna al punto 2 (fino a risoluzione avvenuta)

La figura 1.1 illustra lo svolgimento sopra descritto.

Figura 1.1: Schema del processo di assistenza

1.4 Obiettivi e sintesi risultati

Gli obiettivi del progetto TELLnet si riassumono in:

1. Raccogliere segnalazioni di problemi (errori e malfunzionamenti) da una o più istanze di software di monitoring, con un modello indipendente dal software di monitoring utilizzato, per maggiore flessibilità

2. Elaborare rimedi ai problemi rilevati (comandi eseguibili)

3. Offrire un'interfaccia web user-friendly dove visualizzare i problemi rilevati e i rispettivi rimedi proposti, dando la possibilità di scegliere ed eseguire i rimedi

4. Offrire la possibilità di interagire con TELLnet anche tramite chatbot (testuale e vocale)

Lo scenario obiettivo è illustrato in figura 1.2.

L'obiettivo 1 è stato raggiunto con un modello comune a tutte le segnalazioni, concretizzato in una tabella del database (vedi par. 4.3.2).

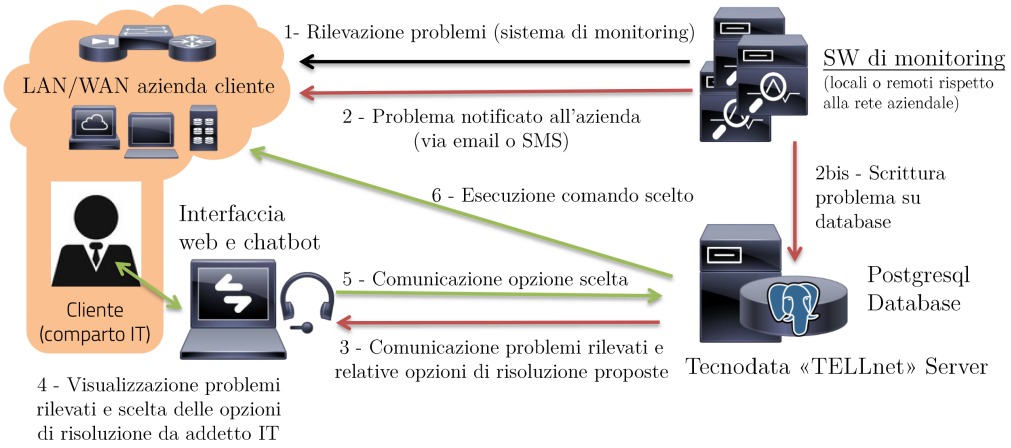

Figura 1.2: Schema del processo di assistenza che si desidera ottenere

L'obiettivo 2 è stato parzialmente raggiunto (vedi par. 4.2.3) con un modello semi-strutturato dei problemi che prevede: un insieme di rimedi parametrizzati inseriti in fase di inizializzazione; l'assegnazione ai nuovi problemi di rimedi conosciuti tramite un calcolo euristico della somiglianza (basato sul modello dei problemi); un algoritmo di istanziamento dei parametri (vedi par. 4.3.3).

L'obiettivo 3 è stato raggiunto con: modello client/server (server Flask) esteso tramite websocket (che permettono di stabilire e mantenere una connessione dati tra browser e server remoto con scambio bidirezionale di messaggi), message queue e dati da visualizzare disponibili rapidamente con Redis (IMDB), esecuzione comandi da remoto tramite chiamate SSH (con libreria paramiko).

L'obiettivo 4 è stato raggiunto con: gestione conversazioni con il servizio Dialogflow (vedi par. 3.1.2), riconoscimento vocale con WebSpeechRecognition (vedi par. 3.1.2), sintesi vocale con Text-to-Speech (vedi par. 3.1.2).

Capitolo 2

Stato dell'arte

2.1 Premessa

Il progetto TELLnet si sviluppa integrando soluzioni appartenenti a più di un ambito tecnologico e disciplinare:

- Gestione e monitoraggio di infrastrutture IT
- Interazione uomo-macchina e ruolo del linguaggio naturale
- Riconoscimento vocale
- Sintesi vocale del testo

Risulta perciò evidente che un'analisi dello stato dell'arte deve essere condotta, almeno in principio, separatamente per ognuno dei punti sopra elencati.

2.2 Monitoraggio infrastrutture IT

Come sostenuto da un articolo[5] recente che analizza gli strumenti attuali per monitorare le reti per rilevare problemi, garantire la disponibilità dei componenti e misurare le risorse utilizzate da tali componenti, le ultime tendenze nell'ambito delle infrastrutture IT, consolidate ormai da circa un decennio, hanno rinnovato l'importanza di un vecchio argomento: il monitoraggio delle infrastrutture IT e delle applicazioni. Ad oggi infatti si può dire che implementare nuove infrastrutture e applicazioni non è mai stato così facile. Virtualizzazione e cloud computing hanno reso questo processo una routine. Il risultato per l'infrastruttura IT delle aziende è stato un aumento nel numero di diversi elementi da gestire. Inoltre, i sistemi odierni di solito sono geograficamente dispersi o utilizzano diversi sistemi operativi, il che complica la loro gestione. Per di più, poiché sempre più aziende si affidano a dei software, l'integrità del sistema IT è fondamentale per assistenza e supporto clienti 24 ore su 24, 7 giorni su 7. I sistemi possono essere applicazioni locali, applicazioni su cloud pubblico o privato, o qualsiasi combinazione di queste.
La prima fase di questo processo è il monitoraggio dell'infrastruttura IT a livello di hardware, servizio e applicazione.

Fra le soluzioni considerate allo stato dell'arte, analizzate nell'articolo suddetto, si trova anche Nagios (vedi appendice B, par. B.1.1), il software utilizzato nei test del progetto TELLnet.
Un punto di vista[8] ancora più recente sostiene che il mondo del monitoraggio e della diagnostica delle infrastrutture IT sia nel bel mezzo di una tempesta di innovazione e cambiamenti netti, causata dai recenti rapidi progressi in molti campi (robotica, realtà immersiva, intelligenza artificiale, IoT) che stanno portando verso un nuovo livello di sofisticazione tecnologica nel mondo fisico. Anche gli ambienti più tradizionali che consistono in un data center aziendale, un po' di utilizzo del cloud e filiali degli uffici remote, non sono immuni a questa pressione ad espandere le capacità della loro infrastruttura.
Gli attori principali in questo campo, pur offrendo soluzioni individuali sui set di strumenti, forniscono tutti:

- Analisi e gestione delle prestazioni: sfruttate dall'apprendimento automatico della telemetria in tempo reale
- Gestione e orchestrazione: che includono semplici interfacce grafiche per il troubleshooting, attività di supporto help-desk, risoluzione dei problemi e reporting
- Automazione del flusso di lavoro: per accelerare la distribuzione e la consegna dei servizi
- Sicurezza e controllo dell'identità: strumenti software di gestione che consentono di controllare i criteri da qualsiasi nodo di rete

Al centro di tutti questi servizi ci sono il monitoraggio e la manutenzione continui della rete mesh necessaria per garantire il massimo tempo di attività e disponibilità di servizi e risorse. A tal fine, oggi ci sono vari aspetti da considerare:

- Analisi delle cause principali: è necessaria una soluzione per l'individuazione dei problemi che sia il più vicino possibile al tempo reale e strettamente legata al processo di gestione delle modifiche, in modo che la ripetizione degli eventi sia notevolmente meno probabile.
- Riparazione automatizzata: quando gli eventi vengono registrati o la telemetria suggerisce che qualcosa sta andando storto, i flussi di lavoro automatizzati possono risolvere i problemi senza l'intervento di un tecnico. Realizzati con cura, sono inestimabili per il corretto funzionamento delle reti.
- Correlazione gerarchica: questi strumenti comprendono la connettività della rete e possono mappare il grafico di causalità appropriato per garantire che vengano indicati gli elementi appropriati.
- Consolidamento degli strumenti: i principali fornitori di sistemi di monitoraggio offrono strumenti integrati che sostituiscono i tipici schermi multipli di un NOC. Il consolidamento di questi in un'unica panoramica, con la possibilità di approfondire un problema quando necessario, è fondamentale per garantire operazioni senza problemi.

- Gestione delle prestazioni: aiuta a capire se le politiche di larghezza di banda sono appropriate ai servizi e anche a identificare ed eliminare i colli di bottiglia per l'efficienza operativa.

- Report sui livelli di servizio: è fondamentale per consentire di quantificare le metriche chiave e altri KPI per garantire la conformità agli SLA.

2.3 Interazione uomo-macchina

L'interazione uomo-macchina, o Human-Computer Interaction (HCI), è un campo della scienza che studia la progettazione e l'uso della tecnologia informatica concentrandosi sulle interfacce tra persone e computer e su come progettare, valutare e implementare sistemi informatici interattivi che soddisfino l'utente. L'importanza di questo campo di studio riguarda sia l'esperienza utente che la sicurezza. Gli esseri umani interagiscono con i computer in molti modi diversi, il che significa che avere una buona interfaccia che faciliti tale interazione è fondamentale. Le interfacce uomo-macchina mal progettate possono portare a molti problemi imprevisti. Un classico esempio è l'incidente della fusione nucleare a Three Mile Island, dove le indagini hanno concluso che il design dell'interfaccia uomo-macchina era in parte responsabile del disastro[7].
All'inizio del progetto di design di un'interfaccia utente è bene che siano proprio le caratteristiche degli utenti che la useranno a guidare le scelte progettuali: chi saranno e quali attività svolgeranno, si richiederà loro qualche tipo di esperienza in qualche area? Con che frequenza verrà eseguita quella specifica attività?
Terminato questo lavoro e deciso cosa si vuole ottenere con l'interfaccia, si può iniziare a implementare un prototipo dell'interfaccia e testarlo con utenti reali il prima possibile. Nell'implementazione è consigliabile seguire alcune buone pratiche:

- Rendere leggibili gli elementi dell'interfaccia: se i caratteri o gli oggetti visualizzati non possono essere percepibili, non possono essere utilizzati in modo efficace.

- La ridondanza è importante: se un segnale viene presentato più di una volta, è più probabile che venga compreso correttamente. La ridondanza non implica la ripetizione. Un semaforo è un buon esempio di ridondanza, poiché il colore e la posizione sono ridondanti.

- La somiglianza confonde: conviene usare elementi distinguibili. I segnali che sembrano essere simili saranno probabilmente confusi. Funzionalità inutilmente simili dovrebbero essere rimosse e funzionalità dissimili dovrebbero essere evidenziate.

- Utilizzo di più risorse: un utente può elaborare più facilmente le informazioni su risorse diverse. Ad esempio, le informazioni visive e uditive possono essere presentate simultaneamente piuttosto che presentare tutte le informazioni visive o uditive.

- Sostituire la memoria con informazioni visive: un utente non dovrebbe aver bisogno di conservare informazioni importanti esclusivamente nella memoria di lavoro o recuperarle dalla memoria a lungo termine. Un menu, un elenco di controllo o un altro display possono aiutare l'utente facilitando l'uso della memoria.

- Non reinventare la ruota: le vecchie abitudini da altre interfacce si trasferiranno facilmente per supportare l'elaborazione di nuove se sono progettate in modo coerente. Un progetto deve accettare questo fatto e utilizzare gli standard già utilizzati in interfacce simili.

2.3.1 Interazione in linguaggio naturale

Parlando di HCI un articolo recente[3] si spinge a dire che una potenziale rivoluzione sta accadendo davanti ai nostri occhi. Per decenni, ricercatori e professionisti dell'interazione uomo-computer (HCI) hanno migliorato le loro capacità nella progettazione di interfacce utente grafiche. Ora le cose possono prendere una svolta inaspettata, verso interfacce utente in linguaggio naturale, in cui l'interazione con i sistemi digitali avviene non tramite scorrimento, scorrimento laterale o clic sui pulsanti, ma piuttosto tramite stringhe di testo in linguaggio naturale. Ciò è particolarmente visibile nei recenti sviluppi nei chatbot, ovvero agenti macchina che fungono da interfacce utente in linguaggio naturale per fornitori di dati e servizi[2], tipicamente nel contesto delle applicazioni di messaggistica. Se i giganti della tecnologia come Google, Facebook e Microsoft hanno ragione, presto l'interazione digitale si sposterà da siti Web e app con interfacce utente grafiche a piattaforme di messaggistica come Messenger e Allo. Se (o meglio quando) ciò accadrà, si apriranno enormi sfide e opportunità nel campo dell'HCI.
Per i fornitori di servizi commerciali o senza scopo di lucro, le interfacce utente in linguaggio naturale sono sul punto di diventare un'interfaccia attraente attraverso la quale interagire con i clienti. Aziende come Domino's Pizza e Taco Bell stanno provando i chatbot come agenti di prenotazione nelle applicazioni di messaggistica. I chatbot medici, come Cardea, forniscono consigli sulla salute e elenchi di medici. I governi stanno esplorando i chatbot come mezzo per facilitare il voto alle elezioni. In un futuro non troppo lontano, i chatbot potrebbero essere l'interfaccia utente preferita per molte delle attività che siamo abituati a svolgere attraverso una pagina web o un'applicazione dedicata. Questa popolarità porta a una domanda importante: quali implicazioni avranno i rapidi sviluppi e l'adozione dei chatbot sul modo in cui ci avviciniamo alla progettazione dei sistemi interattivi in HCI? Stando ai vertici delle maggiori società informatiche globali si tratta di un punto di svolta. Il CEO di Microsoft Satya Nadella ha paragonato la prevista transizione ai chatbot e alle interfacce utente in linguaggio naturale a rivoluzioni precedenti come l'introduzione dell'interfaccia utente grafica, il Web e l'Internet mobile. Il CEO di Facebook Mark Zuckerberg ha affermato che i chatbot sono una soluzione alla sfida del sovraccarico delle app. Se questa visione delle interfacce conversazionali porta con sé qualche verità, importanti cambiamenti sono in serbo per il campo dell'HCI.
Si potrebbe obiettare che le interfacce utente in linguaggio naturale non sono una novità nel campo dell'HCI. In effetti, i ricercatori HCI li hanno già studiati, ad esempio, nel contesto di sistemi multimodali, sistemi di risposta vocale interattivi, controllo vocale nel contesto dell'accessibilità e sistemi di conversazione[1]. Tuttavia, la maggior parte della ricerca e della pratica sull'usabilità riguarda probabilmente le interfacce utente grafiche e, in una certa misura, la progettazione dell'hardware. Come campo, si sono trascorsi gli ultimi due decenni a perfezionare il modo di progettare per l'interazione con pagine Web o app, attingendo a meccanismi di interazione sempre più ricchi per supportare l'usabilità e l'esperienza dell'utente.

Una transizione ai chatbot e alle interfacce utente in linguaggio naturale ha molte implicazioni, alcune particolarmente degne di nota:

- Conversazioni come oggetto di design: nella futura era dei chatbot e delle interfacce utente in linguaggio naturale, progettare per l'usabilità implicherà suggerire all'utente cosa può aspettarsi dal servizio e l'interpretazione adeguata della sua risposta. Vedere le conversazioni come l'oggetto del design rappresenta chiaramente una sfida nel campo dell'HCI, un passaggio dal vedere il design come un compito esplicativo, cioè un compito di spiegare all'utente quali contenuti e caratteristiche sono disponibili e quali passi intraprendere per raggiungere l'obiettivo desiderato, a un compito interpretativo, cioè un compito di capire di cosa ha bisogno l'utente e come può essere servito al meglio. Allo stesso tempo, questa sfida può rafforzare la nostra comprensione di come adattare dinamicamente l'interfaccia utente man mano che il dialogo con l'utente evolve.

- Necessità di passare dalla progettazione di interfacce utente alla progettazione di servizi: la ricerca e la pratica HCI hanno ormai analizzato a fondo come progettare interfacce utente specifiche. L'attenzione come campo è stata rivolta all'oggetto del design (il sistema interattivo) piuttosto che agli obiettivi dell'utente. Ciò è ragionevole, date le basi di ingegneria del software e progettazione dei sistemi. Tuttavia, il passaggio alle interfacce utente in linguaggio naturale può implicare la necessità di ripensare questo obiettivo.
  Nell'era futura dei chatbot e delle interfacce utente in linguaggio naturale, i contenuti e i servizi non si differenziano per le loro interfacce utente, ma piuttosto per la loro comodità nell'accesso al contesto dei thread conversazionali. Per ricercatori e professionisti ciò implica la progettazione di interi processi di servizio attraverso punti di contatto conversazionali con l'utente, piuttosto che interfacce utente specifiche. Su questi temi c'è probabilmente qualcosa da imparare dal campo emergente del design dei servizi.

- Necessità di progettare per l'interazione in reti di attori umani e di macchine intelligenti: il design in HCI spesso riguarda un utente, un dispositivo. Ciò è particolarmente vero per quanto riguarda l'interaction design. Qui, l'oggetto di progettazione e valutazione è tipicamente l'interfaccia utente, in quanto sarà percepita e agita da un singolo utente. Questo non vuol dire che non esistano altri approcci al design. Ad esempio, i campi confinanti del game design, del lavoro collaborativo supportato dal computer e della progettazione di sistemi sociotecnici hanno esplorato la progettazione per sistemi multi-agente.

Principali soluzioni chatbot

Come illustrato in un articolo recentissimo [9], tutte le principali società IT hanno sviluppato piattaforme basate su cloud che consentono di creare un chatbot in pochi passaggi e la maggior parte delle volte senza conoscenza dei linguaggi di programmazione. Questi servizi si basano su motori di NLU (Natural Language Understanding) che si occupano dell'identificazione di informazioni come entità e intenti dalle frasi fornite come input.
Nella scelta di quale soluzione adottare per il chatbot del progetto TELLnet si è tenuto conto dei risultati (presentati nel suddetto articolo) riguardanti le prestazioni delle principali

piattaforme NLU disponibili sul mercato (IBM Watson, Google Dialogflow, Microsoft Luis, Facebook Wit.ai) per quanto riguarda la lingua italiana. Questi risultati hanno portato alla scelta del servizio DialogFlow (vedi par. 3.1.2) offerto dalla Google Cloud Platform.

2.4 Riconoscimento vocale

Il riconoscimento vocale, noto anche come riconoscimento vocale automatico (ASR, Automatic Speech Recognition) o Speech-to-Text, è una capacità che consente a un programma di elaborare il discorso umano in un formato scritto. Sebbene sia comunemente confuso con la Voice Recognition (in italiano ancor di più dato che la traduzione è comunque riconoscimento vocale), il riconoscimento vocale viene in questa sede inteso come quel processo che si concentra sulla traduzione del parlato da un formato verbale a uno di testo scritto, mentre la Voice Recognition cerca solo di identificare la voce di un singolo utente.
Sebbene le soluzioni di riconoscimento offerte dai colossi del settore e basate sul machine learning rappresentino la forma più avanzata e sofisticata disponibile, per il progetto TELLnet si è fatta una scelta diversa adottando il servizio WebSpeechRecognition (vedi par. 3.1.2). Questa scelta dipende in gran parte dalla considerazione del rapporto fra semplicità di implementazione e qualità dei risultati. Almeno in questa fase dello sviluppo del progetto TELLnet, si è preferito non complicare ulteriormente l'architettura del progetto. Infatti per interfacciarsi con i servizi più sofisticati il browser necessiterebbe dell'uso del framework WebRTC per inviare streaming audio. WebRTC a sua volta richiederebbe l'implementazione (nel backend o in un altro server) della gestione di tale streaming audio in input dal browser e in output verso i servizi di riconoscimento. Invece WebSpeechRecognition semplifica notevolmente lo sviluppo perché implementa già la gestione dello streaming verso un servizio di riconoscimento ed offre comunque prestazioni accettabili e trascrizione "real-time", dove per real-time si intende che non è necessario aspettare la fine della frase per vedere i risultati della trascrizione (i risultati parziali mostrati possono subire variazioni finché non vengono confermati, cosa che avviene al più tardi a fine frase).
Non si esclude che futuri sviluppi del progetto TELLnet possano giovarsi di servizi più sofisticati, ma per gli scopi di questa fase di sviluppo questa soluzione si ritiene adeguata.

2.5 Sintesi vocale del testo

La sintesi vocale è la produzione artificiale per simulazione, generata da computer, del linguaggio umano parlato. Un sistema di sintesi vocale del testo (TTS, Text-to-Speech) converte il testo dal linguaggio naturale scritto (in una qualunque lingua normale) in parlato.
Esistono varie aziende che si contedono il ruolo di leader in questo settore, perlopiù le stesse citate precedentemente riguardo ai chatbot e al riconoscimento vocale, dato che i tre ambiti sono strettamente collegati. Non a caso, anche i migliori servizi di sintesi vocale utilizzano il machine learning.
Per dare voce al chatbot del progetto TELLnet si è scelto il servizio Text-toSpeech della Google Cloud Platform (vedi par. 3.1.2). Le motivazioni di questa scelta sono essenzialmente due: il servizio rientra fra i migliori a disposizione ed è integrato nella stessa piattaforma di servizi che offre anche Dialogflow (la GCP), semplificando per coerenza strutturale il

processo di sviluppo e ottimizzando l'utilizzo delle risorse computazionali perché ottenute dalla sottoscrizione alla medesima piattaforma di servizi.

Capitolo 3

Strumenti utilizzati

3.1 Client

Le tecnologie utilizzate nello sviluppo del FrontEnd sul lato client di TELLnet permettono all'utente di interagire tramite una classica interfaccia web (mouse e tastiera) e/o tramite un chatbot testuale e vocale.

3.1.1 Interfaccia web

L'interfaccia web, costituita da una pagina di login e una pagina principale, è sviluppata con i classici linguaggi per i siti web:

- Javascript (vedi appendice B, paragrafo B.2.1)
- HTML (vedi appendice B, paragrafo B.2.2)
- CSS (vedi appendice B, paragrafo B.2.3)

In particolare, l'aspetto dell'interfaccia è basato su Bootstrap 4.5.0, un Framework CSS pensato per sviluppare siti web responsive[1], personalizzato poi con ulteriori regole CSS definite nei file signin.css (per la pagina di login) e home.css (per la pagina principale). Alcune animazioni e funzionalità delle pagine, come la finestra di dialogo per la visualizzazione dei rimedi di un problema, utilizzano la libreria jQuery 3.5.1 e i relativi CSS (jQueryUI 1.12.1). L'aggiornamento asincrono dei dati mostrati nella pagina principale, così come altri messaggi scambiati con il server, è reso possibile dall'utilizzo delle WebSockets API, che permettono di stabilire e mantenere una connessione dati tra browser e server remoto con scambio bidirezionale di messaggi.

3.1.2 Chatbot

Nella pagina principale del FrontEnd si può interagire anche con un chatbot, basato su tre tecnologie:

[1] Che si adattano a seconda delle dimensioni dello schermo

- WebSpeechRecognition per il riconoscimento vocale
- Google Cloud Text-to-Speech per la sintesi vocale
- Dialogflow ES per la gestione delle conversazioni

WebSpeechRecognition

Parte delle Web Speech API[2], WebSpeechRecognition permette di trascrivere il parlato mostrando in diretta la trascrizione e distinguendo fra trascrizione temporanea e definitiva; si tratta di una funzionalità gratuita pensata appositamente per i browser, compatibile con centinaia di lingue (numero in costante aumento).

Google Cloud Text-to-Speech

Il servizio Google Cloud Text-to-Speech fa parte della Google Cloud Platform, piattaforma di servizi professionali offerti da Google. Si tratta di un servizio di sintesi vocale basato su machine learning. Il servizio richiede abbonamento, ma mette a disposizione un periodo di prova gratuita.
La scelta di utilizzare un servizio di sintesi vocale basato sul machine learning permette di ottenere risultati notevolmente superiori ad altre soluzioni, con voci meno metalliche, allo scopo di migliorare nettamente l'esperienza utente.

Dialogflow ES

Il servizio Dialogflow ES fa anch'esso parte della Google Cloud Platform. Si tratta di una piattaforma di comprensione del linguaggio naturale che semplifica la progettazione e l'integrazione di un'interfaccia utente conversazionale. Dialogflow può analizzare sia input testuali che audio.
Le interfacce tradizionali per computer richiedono un input strutturato e prevedibile per funzionare correttamente, il che rende l'uso di queste interfacce innaturale e talvolta difficile. Se gli utenti finali non riescono a comprendere facilmente questo input strutturato, hanno difficoltà a capire cosa fare. Questo è ancora più vero se l'interazione avviene con una conversazione: non si può pensare di elencare in modo esaustivo tutte le possibili varianti equivalenti di una frase del linguaggio naturale, men che meno farlo per ogni possibile frase alla quale si vuole poter rispondere efficacemente.
Per superare questi limiti Dialogflow utilizza un modello basato su:

- **Agent**: gestisce le conversazioni con gli utenti finali. È un modulo di comprensione del linguaggio naturale che comprende le sfumature del linguaggio umano
- **Intents**: ogni Intent classifica l'intenzione di un utente finale per un turno di conversazione. Per ogni Agent si definiscono molti Intent per gestire una conversazione completa. Quando un utente scrive o dice qualcosa, Dialogflow abbina l'espressione dell'utente al miglior Intent nel proprio Agent. La scelta della corrispondenza di un

[2]Documentazione: https://wicg.github.io/speech-api/

Intent è una classificazione basata sul machine learning, a partire da frasi di training definite per addestrare l'Agent. Grazie a questo modello le frasi scelte per riconoscere una richiesta (o comunque un input, anche un evento) dell'utente non hanno bisogno di essere eccessivamente esplicite. Negli Intent si possono definire anche dei parametri: quando un Intent viene scelto in fase di esecuzione, Dialogflow fornisce i valori estratti dall'espressione dell'utente finale come parametri; ogni parametro ha un tipo, chiamato tipo di entità, che determina esattamente come vengono estratti i dati. A differenza dell'input non elaborato dell'utente, i parametri sono dati strutturati che possono essere facilmente utilizzati per eseguire una logica o generare risposte. Per ogni Intent si possono definire varie risposte (testuali, vocali, strutturate o anche generate da codice esterno richiamato, detto webhook).

- **Entità**: ogni parametro degli Intent ha un tipo, denominato tipo di entità, che determina esattamente come vengono estratti i dati da un'espressione dell'utente. Dialogflow fornisce entità di sistema predefinite che possono corrispondere a molti tipi comuni di dati. Ad esempio, esistono entità di sistema per la corrispondenza di date, orari, colori, indirizzi e-mail e così via. Permette inoltre di creare entità personalizzate per abbinare i dati.

- **Contesti**: i contesti del flusso di dialogo sono simili al contesto del linguaggio naturale. Permettono di definire i flussi delle conversazioni ed estrapolare dati da una parte precedente della conversazione per comprendere frasi in cui quegli stessi dati sono sottintesi. Quando un Intent viene scelto, tutti i contesti di output configurati per quell'Intent diventano attivi. Mentre tutti i contesti sono attivi, è più probabile che Dialogflow scelga Intent configurati con i contesti di input che corrispondono ai contesti attualmente attivi.

3.2 Application Server

Le tecnologie utilizzate nel BackEnd hanno lo scopo di mettere a disposizione un server che gestiste la comunicazione fra FrontEnd, logica applicativa (TELLnet Core) e basi di dati.

3.2.1 Flask

Flask è un framework per applicazioni web WSGI leggero. Progettato per adattarsi ai piccoli progetti ma con la possibilità di scalare fino ad applicazioni complesse. Attualmente è uno dei framework di applicazioni web Python più popolari. Non pone particolari vincoli di progettazione, lasciando ampio margine di personalizzazione attraverso strumenti, librerie ed estensioni a disposizione. Nel progetto TELLnet sono utilizzate anche le librerie Flask-Login (per le operazioni di autenticazione), Flask-RESTful (per offrire webhook come servizio REST, vedi par. 3.2.3) e FlaskSocketIO per estendere il server con la gestione dei socket per scambio di messaggi.

3.2.2 Paramiko

Nella logica applicativa (TELLnet Core) è necessario utilizzare il protocollo SSH per eseguire comandi da remoto (e.g. per applicare i rimedi scelti, che altro non sono se non comandi che eseguono script); a tale scopo si utilizza la libreria paramiko 2.7.1 del linguaggio Python dal momento che in questo linguaggio è scritto il codice del BackEnd, sia per le funzionalità server (con Flask, come suddetto) che per la logica applicativa.

3.2.3 Webhook

Un Webhook nella programmazione web è un metodo per aumentare o alterare il comportamento di una pagina web, o di un'applicazione web, con chiamate di ritorno personalizzate. Viene qui utilizzato come servizio agganciato a Dialogflow per estendere ed elaborare a runtime alcune risposte del chatbot, attivando questo servizio negli Intent per i quali si desidera maggiore flessibilità di risposta. Ad esempio, per dare il benvenuto il chatbot utilizza un webhook che controlla l'orario corrente per adattare il saluto.

3.3 Database Server

TELLnet fa uso di due distinti DBMS con caratteristiche differenti per gestire l'archiviazione in memoria dei dati (Redis e PostgreSQL). Nell'implementazione attuale sono entrambi installati sullo stesso elaboratore dell'Application Server ma possono essere dislocati diversamente secondo necessità. Tuttavia è consigliabile mantenere almeno Redis dove si trova viste le motivazioni del suo utilizzo (vedi par. 3.3.2).

3.3.1 PostgreSQL

PostgreSQL è un sistema di gestione di database relazionale open source che utilizza ed estende il linguaggio SQL combinato con molte funzionalità che archiviano e ridimensionano in modo sicuro i carichi di lavoro dei dati più complicati.
PostgreSQL si è guadagnato una solida reputazione per la sua comprovata architettura, affidabilità, integrità dei dati, robusto set di funzionalità, estensibilità e la dedizione della comunità open source dietro il software per fornire costantemente soluzioni performanti e innovative. Viene utilizzato nel progetto per mantenere in memoria di massa tutte le informazioni stabili relative alla logica applicativa (TELLnet Core), come informazioni sui problemi conosciuti e su tutta la struttura dei rimedi relativi conosciuti, sulle segnalazioni ricevute, sullo storico delle segnalazioni risolte.
Per interagire con il db PostgreSQL si fa uso della libreria Python psycopg2.

3.3.2 Redis

Redis è un archivio in-memory per strutture dati, utilizzato come database (IMDB), cache e broker di messaggi. Supporta numerose strutture dati: stringhe, hash, liste, insiemi, bitmap e molte altre. Ha una struttura chiave-valore ed è in grado di garantire la proprietà di durabilità. Redis ha reso popolare l'idea di preferire database in-memory rispetto ai database tradizionali perché in grado di sfruttare i vantaggi della memoria principale come

la velocità e i vantaggi della memoria secondaria come la persistenza dei dati. Infatti, i dati vengono sempre modificati e letti dalla memoria primaria ma, allo stesso tempo, possono essere anche memorizzati sulla memoria secondaria in un formato non leggibile e utilizzabile soltanto per ricostruire i dati nel momento del riavvio del sistema. Redis è un prodotto open source.
Per interagire con Redis si fa uso della libreria Python redis 3.5.1.

Capitolo 4

Progettazione e sviluppo BackEnd

4.1 Funzionalità

Il BackEnd del progetto TELLnet ne racchiude il cuore della logica applicativa (TELLnet Core). Il suo scopo è di automatizzare il processo di assistenza e recovery nella gestione dei sistemi informativi attraverso:

- Ricezione ed elaborazione delle segnalazioni provenienti da uno o più software di monitoring, tenendo traccia degli aggiornamenti
- Invio al FrontEnd dello stato aggiornato dei problemi segnalati e dei relativi rimedi elaborati da proporre
- Esecuzione dei comandi, relativi ai rimedi scelti, tramite chiamate remote su protocollo SSH
- Verifica dell'esito dei rimedi tentati e notifica al FrontEnd
- Archiviazione dello storico dei problemi rilevati con informazioni sulla loro evoluzione, dei rimedi tentati e degli esiti

4.2 Scelte progettuali

Per implementare le funzionalità descritte e raggiungere così gli obiettivi del progetto si sono rese necessarie alcune scelte progettuali che hanno poi influenzato l'architettura e il funzionamento del BackEnd. Le scelte principali riguardano:

- **Modello delle segnalazioni**: quale struttura dati per le segnalazioni è accettata da TELLnet
- **Modello dei problemi**: quale struttura dati per descrivere i problemi monitorati è accettata da TELLnet

- **Metodo di elaborazione dei rimedi**: quale forma di elaborazione dati utilizzare per assegnare rimedi ai problemi

4.2.1 Perché un modello delle segnalazioni

La scelta di definire un modello per le segnalazioni nasce da due necessità:

- Strutturare i dati delle segnalazioni dandogli una forma comune per poter fare un'analisi automatica agevole e veloce; la struttura (descritta nel par. 4.3.3) comprende anche un campo facoltativo e semistrutturato (in formato json) per consentire l'arricchimento informativo delle segnalazioni, per consentire e gestire l'eterogeneità dei problemi monitorati allo scopo di non restringere l'insieme dei problemi gestibili dal sistema.

- Offrire un modello generico che sia adottabile da qualunque sistema di monitoring sufficientemente flessibile (cioè che permetta di configurare come strutturare i dati delle segnalazioni emesse) per non limitare la scelta di questa importante componente. Nel testare il progetto TELLnet è stato utilizzato Nagios (vedi B.1.1) come sistema di monitoring, un software flessibile e altamente configurabile[5].

4.2.2 Perché un modello dei problemi

La scelta di definire un modello per i problemi nasce dalla necessità di automatizzare (per ora parzialmente) il processo di estensione dell'insieme dei problemi monitorati, rispetto a quelli definiti fin dall'avvio e dall'inizializzazione di TELLnet, mitigando così lo sforzo descrittivo di configurazione che si presenta quando si riscontra la necessità di estendere le funzionalità a nuovi problemi. Senza questa soluzione sarebbe necessario inserire manualmente i dati necessari per gestire ogni nuovo problema che si desideri monitorare e gestire, anche quelli concettualmente equivalenti a problemi conosciuti ma che differiscono per parametri contestuali (e.g. richiedono la cancellazione di un file, ma i nome dei file o dei path nel filesystem sono differenti). Tale sforzo resta consistente nella definizione dell'insieme di partenza dei problemi monitorati e relativi rimedi, durante il quale è importante inserire la maggior parte (idealmente tutte) delle classi di problemi che si desidera far gestire a TELLnet. Il modello definito (descritto nel par. 4.3.3) è semistrutturato allo scopo di gestire con flessibilità l'inevitabile eterogeneità dei problemi che si possono voler monitorare in un'infrastruttura IT. Questa scelta ha permesso di sviluppare un algoritmo euristico (descritto nel par. 4.3.4) in grado di calcolare la somiglianza fra due problemi, sulla base della loro struttura e dei valori delle loro caratteristiche, consentendo così di poter elaborare rimedi per problemi non ancora conosciuti da TELLnet a partire dai rimedi del problema conosciuto scelto come più somigliante. Resta da considerare che le reali potenzialità di questo metodo scelto dipendono dallo sforzo descrittivo iniziale suddetto; infatti le probabilità di elaborare automaticamente rimedi corretti per un nuovo problema sono decisamente maggiori se questo appartiene ad una classe di problemi già presente fra le conoscenze di TELLnet.

4.2.3 Come elaborare i rimedi

La scelta del metodo per elaborare i rimedi (sequenze di uno o più comandi eseguibili) per i problemi monitorati costituisce forse la sfida più grande in tutta la progettazione di TELLnet. Ad oggi le competenze e la conoscenza (il know-how) necessarie per definire le azioni concrete da mettere in atto nella risoluzione dei problemi generici che si possono presentare in un'infrastruttura IT sono detenute da tecnici umani, in genere sistemisti. Questo progetto si pone fra gli obiettivi (il secondo nel par. 1.4) quello di riversare almeno parzialmente questo know-how in un software.
Il numero di possibili scenari problematici è incalcolabile se si considera le innumerevoli sfumature nei dettagli che vengono gestiti dagli operatori umani grazie alla capacità umana di categorizzare in classi di problemi e allo stesso tempo saper "improvvisare" sui dettagli, cioè reagire a dettagli non considerati a priori grazie a deduzioni dalle proprie conoscenze o persino alla capacità di ricercare nuove conoscenze necessarie a risolvere il problema. Questo significa che scegliere di enumerare tutti i problemi e relativi rimedi che TELLnet dovrebbe saper risolvere non è una strada percorribile.
Allo stesso tempo, l'idea di creare un modello logico, con una qualche forma di intelligenza artificiale, in grado di dedurre i rimedi da attuare a partire dalle informazioni sui problemi segnalati, si scontra non solo con la complessità di uno scenario eterogeneo e dipendente dal contesto, ma anche con le caratteristiche fondamentali dei comandi eseguibili che costituiscono in concreto i rimedi da attuare.
I comandi eseguibili sono in genere esecuzioni di script che operano a basso livello, che implementano operazioni considerabili in gran parte come atomiche dal punto di vista dei ragionamenti riguardanti l'interazione uomo-macchina per ottenere un qualche risultato (e.g. cancellare o spostare un file). Sono infatti utilizzati quotidianamente dai sistemisti per amministrare le infrastrutture IT.
I comandi sono composti da un nome per invocarli, da un set di opzioni fra cui scegliere per variarne in qualche misura il funzionamento e i risultati e da parametri di input. Un esempio di comando può essere: **rm -f /dir/prova.txt**. Questo comando cancella (il nome rm sta per remove) il file prova.txt che si trova nel percorso /dir (parametri) senza chiedere conferma all'utente (opzione -f).
Tuttavia la sintassi dei comandi, cioè quanti e quali opzioni e parametri accettano e in che ordine e forma, dipende dalla definizione dei relativi script che li implementano; non esiste una standardizzazione, seppure molti comandi comunemente usati presentano molte somiglianze. Per di più alcuni comandi sono dipendenti dal sistema operativo per cui sono stati scritti, per cui si possono avere comandi omonimi su sistemi diversi che sono in realtà diversi e comandi che hanno nomi diversi ma stesso scopo che funzionano su sistemi diversi, spesso anche con differenze nella sintassi e/o nei risultati. Sono perciò almeno due le caratteristiche dei comandi che si scontrano con un'approccio logico-deduttivo o con altre forme di intelligenza artificiale (e.g. machine learning) per la loro generazione:

- Hanno una sintassi non standardizzata
- Necessitano di essere scritti in modo esatto, cambiare un dettaglio ne cambia il funzionamento o li rende non eseguibili

Per tutte le motivazioni suddette la scelta finale del metodo per elaborare rimedi è

ricaduta su un approccio con sintassi parametrizzata. Per comando con sintassi parametrizzata, o comando parametrizzato, si intende un comando definito con parametri che fungono da segnaposto per i valori effettivi con cui verranno sostituiti prima di essere eseguiti (vedi par. 4.3.4). Questa scelta permette di ottenere maggiore flessibilità e riuso delle sintassi conosciute rispetto all'uso diretto di comandi eseguibili (che nel progetto TELLnet sono chiamati comandi istanziati). Non si tratta di una soluzione perfetta perché mantiene ancora una certa rigidità, tuttavia allo stato attuale la si è considerata un compromesso necessario per le motivazioni sopra espresse. Per questo motivo nel par. 1.4 questo obiettivo è stato definito come parzialmente raggiunto.
Questo modello permette di definire classi di rimedi (istanziati) che condividono la stessa sintassi parametrizzata, riducendo notevolmente il volume di informazioni da inserire in fase di inizializzazione di TELLnet.

4.3 Architettura

L'architettura generale del progetto, scelta e sviluppata per implementare le funzionalità suddette, è mostrata in figura 4.1 e riguardo al BackEnd è sintetizzabile con:

- Tellnet Core (elaborazione logica applicativa)
- Redis (message queue e mantenimento dati aggiornati per FrontEnd)
- Flask (Server Web, Socket Manager e Webhook)

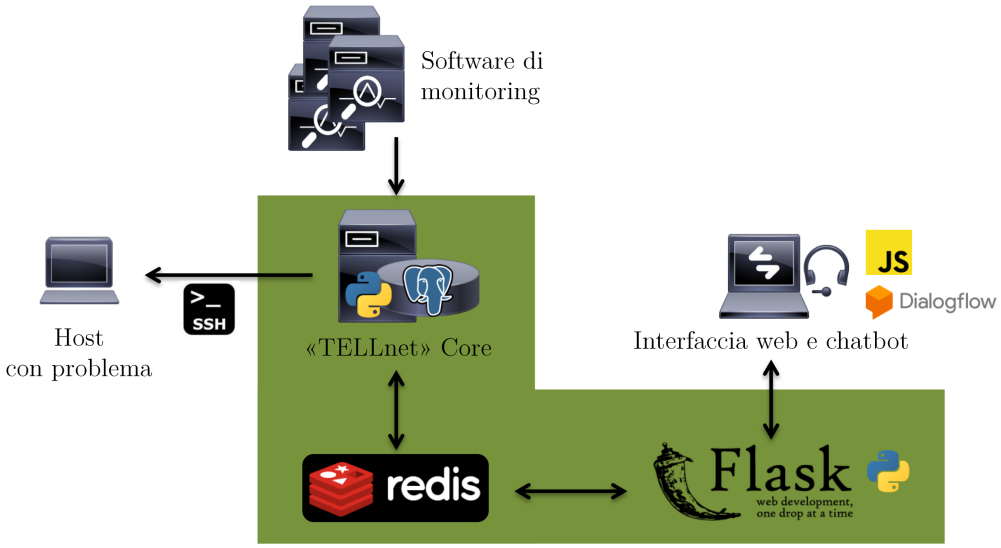

Figura 4.1: Schema dell'architettura di TELLnet (in verde il BackEnd)

4.3.1 TELLnet Core

Il cuore della logica applicativa risiede in TELLnet Core, un insieme di script Python che svolgono compiti specifici:

- **tellnet.py**: inizializza il programma utilizzando i dati di configurazione letti dal file config.ini, poi (ciclicamente) richiama il metodo check_up dello script check.py, da cui ottiene ogni volta i dati aggiornati sullo stato delle segnalazioni dei problemi, riorganizza tali dati in formato json, li salva in Redis aggiornando i dati precedenti e li invia al FrontEnd con un messaggio tramite socket (vedi listati 4.1 e 4.2).

```
1  def start():
2      # INITIALIZING CONFIGURATION
3      wip = []
4      params_wl = config_wl()
5      ext_sio = SocketIO(message_queue='redis://')
6      redisClient = redis.StrictRedis()
7      networks = []
8      with Database() as db:
9          networks = db.get_known_networks()
10     # START CHECK
11     starttime = time.time()
12     while True:
13         with Database() as db:
14             wip = check_up(wip, db, ext_sio, redisClient)
15         interval = params_wl.get('check_up_interval', 60)
```

Listato 4.1: Script Python di avvio: tellnet.py, parte iniziale

```
1              infos = {"date": datetime.datetime.now(), "wip":
                   wip_infos[key]}
2              infos_json = dumps(infos, default=str)
3              redisClient.hset("Wip", key, infos_json)
4              ext_sio.emit('update_wip', infos_json,
                   namespace="/private", room=key)
5          time.sleep(interval - ((time.time() - starttime) %
               interval))
6
7
8  if __name__ == '__main__':
9      start()
```

Listato 4.2: Script Python di avvio: tellnet.py, parte finale

- **check.py**: legge dal database nuove segnalazioni (non ancora lette) e aggiorna la struttura dati dello stato corrente, che descrive le segnalazioni vecchie e nuove di problemi ancora presenti. Per ogni nuovo problema segnalato avvia in un nuovo thread lo script action.py (passando i dati del problema) per avviare la ricerca di rimedi. Per problemi conosciuti non risolti dai rimedi tentati avvia di nuovo action.py con l'informazione sui rimedi già tentati. Per le segnalazioni di risoluzione di problemi

avvia in un nuovo thread lo script store.py per archiviare le informazioni relative e invia una notifica sull'esito al FrontEnd con un messaggio tramite socket. Infine restituisce a tellnet.py (che l'ha avviato) lo stato corrente aggiornato.

- **action.py**: elabora rimedi per un problema dato, a partire dai dati conosciuti che legge dal database e li ordina per probabilità di successo (contenuta fra i dati nel database, dovuta ad esperienza passata); tiene conto anche di eventuali rimedi già tentati precedentemente per la stessa occorrenza del problema. Se il problema segnalato e i relativi rimedi non sono presenti nel database, recupera con una chiamata SSH le informazioni sul problema tramite richiesta del file checks.yaml all'host su cui si è presentato (vedi par. 4.3.3); quindi salva tali informazioni ed avvia il metodo problem_similarity dello script compare.py e dai risultati ricevuti sceglie il problema più simile fra quelli con rimedi compatibili e ne eredita i rimedi, sostituendo i valori dei parametri con quelli del nuovo problema (vedi par. 4.3.4). Un rimedio è compatibile con un problema se tale problema presenta nella propria struttura (vedi par. 4.3.3) tutti i parametri presenti nel comando eseguibile relativo al rimedio.

- **store.py**: utilizza metodi dello script postgres.py per archiviare i dati relativi alle segnalazioni risolte (timestamp di comparsa e risoluzione, rimedi tentati con relativi esiti e cronologia degli stati attraversati)

- **compare.py**: implementa un'euristica che calcola il grado di somiglianza di un problema dato con tutti quelli conosciuti sulla base della struttura e sui valori del problema (vedi par. 4.3.3) e dell'host relativo.

- **postgres.py**: implementa i metodi per interagire con le tabelle del database PostgreSQL, per eseguire le operazioni CRUD, gestire transazioni ed eventuali eccezioni (errori a runtime) e convertire le strutture dati da quelle del database a quelle utilizzate negli altri script Python e viceversa.

- **connections.py**: implementa i metodi che necessitano di effettuare chiamate con protocollo SSH, cioè execute per eseguire un comando su un host remoto e get_file per richiedere un file da un host remoto.

- **command.py**: script eseguito in background (creando un thread) dal server Flask alla ricezione di un messaggio dal FrontEnd (tramite socket) che richiede l'esecuzione di un rimedio dato. Utilizza postgres.py per recuperare i dettagli del rimedio scelto e connections.py per effettuare la chiamata SSH per eseguire il comando.

- **df_webhook.py**: script eseguito in background (creando un thread) dal server Flask; implementa i servizi webhook che ampliano le potenzialità del chatbot e la personalizzazione delle risposte. Il metodo set_context (vedi listato 4.3) permette di attivare un contesto in una conversazione attiva col chatbot (in Dialogflow); in questo modo, all'arrivo sul FrontEnd di una nuova segnalazione di un problema, viene gestito l'evento e impostato il contesto istanziando il parametro che permette di identificare quale problema fra quelli elencati è l'oggetto della conversazione se l'utente effettua una richiesta dandolo per sottinteso (situazione descritta più nel dettaglio nel capitolo 5, par. 5.4.1).

```
1  def set_context(project_id, session_id, language_code, text,
       output_context, lifespan_count=5, parameters=None):
2      session_client = init_session_client()
3      session = session_client.session_path(project_id, session_id)
4
5      if text and output_context:
6          context_path = "projects/" + project_id +
               "/agent/sessions/" + session_id + "/contexts/" +
               output_context
7          text_input = dialogflow.types.TextInput(text=text,
               language_code=language_code)
8          query_input = dialogflow.types.QueryInput(text=text_input)
9          response = None
10         if parameters:
11             params = dialogflow.types.struct_pb2.Struct()
12             for key in parameters:
13                 params[key] = parameters[key]
14             context = dialogflow.types.Context(name=context_path,
                   lifespan_count=lifespan_count, parameters=params)
15             query_params =
                   dialogflow.types.QueryParameters(contexts=[context])
16             response =
                   session_client.detect_intent(session=session,
                   query_input=query_input, query_params=query_params)
17         else:
18             response =
                   session_client.detect_intent(session=session,
                   query_input=query_input)
```

Listato 4.3: Metodo che attiva un contesto su dialogflow, dallo script df_webhook.py

Questa organizzazione del codice ha anche lo scopo di rendere intuitivo il compito delle varie parti di TELLnet Core, per agevolare manutenzione e sviluppi ulteriori.

4.3.2 Modello segnalazioni

Allo scopo di raccogliere le segnalazioni dei problemi (errori e malfunzionamenti) a prescindere da quali e quanti software di monitoring si desideri utilizzare per la rilevazione, si è sviluppato un modello da rispettare per tali segnalazioni. Rispettare tale modello è l'unico requisito per permettere a TELLnet Core di elaborare tali informazioni; i dettagli su come queste siano state ottenute è irrilevante per l'elaborazione di TELLnet.

Struttura Dati

I dati sulle segnalazioni vanno inseriti nella tabella unsolved_problems del database PostgreSQL e devono rispettare la struttura mostrata in tabella 4.1. Un esempio di dati nella tabella unsolved_problems è illustrato in figura 4.2.

Campo	Tipo di dato	Obbligatorio	Concetto
check_name	stringa	SI	identifica il problema
host_name	stringa (indirizzo IPv4)	SI	identifica l'host su cui si è verificato il problema
network	stringa	SI	identifica la rete (o "sistema") di cui fa parte l'host
check_interval	numero intero	SI	intervallo (in secondi) fra due controlli del sistema di monitoring
timestamp	timestamp (YYYY-MM-DD HH:MM:SS)	SI	istante in cui è stato effettuato il controllo
status	stringa	SI	entità del problema
values	json	NO	descrive caratteristiche variabili del problema

Tabella 4.1: Modello per le segnalazioni

check_name	host_name	network	check_interval	timestamp	status	values
controllo_nagios.ctrl	172.18.70.10	tn01ng	60	2020-12-03 21:57:53	Critical	{"exists": true}
controllo_nagios.ctrl	172.18.70.10	tn01ng	60	2020-12-03 21:58:53	Critical	{"exists": true}
controllo_nagios.ctrl	172.18.70.10	tn01ng	60	2020-12-03 21:59:53	Critical	{"exists": true}

Figura 4.2: Tabella del database con dati di segnalazioni

Regole

Oltre a rispettare la struttura dati mostrata in tabella 4.1, il modello richiede di seguire tre regole nell'invio delle segnalazioni:

- Continuare i controlli anche dopo la prima segnalazione di un problema, inviando nuove segnalazioni periodiche fintanto che il problema permane (l'intervallo fra due segnalazioni deve corrispondere a quello indicato nel campo *check_interval* delle segnalazioni stesse).
- Inviare una segnalazione "speciale" la **prima volta** che si verifica la scomparsa di un problema precedentemente segnalato, indicando con *OK* lo stato della segnalazione.
- Nel campo *status* la stringa *OK* rappresenta un valore "speciale", **riservato** alle segnalazioni di scomparsa del problema; valori consigliati per le segnalazioni di errori sono *CRITICAL* e *WARNING*, ma è possibile utilizzarne anche altri.

4.3.3 Modello dei problemi

Allo scopo di parametrizzare i problemi per poter riutilizzare gli stessi rimedi parametrizzati, istanziando per ciascuno i propri valori, la struttura dei problemi da monitorare viene definita da un modello semi-strutturato.

Struttura dati

La struttura utilizzata per la definizione di un problema è molto simile per caratteristiche a quella dei file json; permette infatti una organizzazione nidificata delle informazioni. La sintassi utilizzata è quella dei file con estensione .yaml, più chiara e leggibile rispetto ad un json. YAML viene definito dai suoi creatori come "uno standard di serializzazione dei dati a misura d'uomo per tutti i linguaggi di programmazione"[1]. La scelta di YAML dipende anche dalla sua popolarità di utilizzo per i file di configurazione con gli script Python, il quale offre una comoda libreria (pyYAML) per la sua gestione.
La struttura definita per la descrizione dei problemi in TELLnet è mostrata nel listato 4.4, con l'uso dei commenti (#) anche per mostrare le alternative fra keyword.

```
1 nome_problema:
```

[1]Vedi https://yaml.org/

```
2   # campi obbligatori
3   type: # binary | discrete -> keywords
4   subtype: # exists | open | percentage_thresholds |
        discrete_thresholds | ... -> keywords
5   subject_type: # file | port | ... -> keywords
6   # campi facoltativi
7   file_type: tipo_di_file # scelto fra 7 possibili
        keywords
8   path: /absolute/path
9   filename: nome_file # con eventuale estensione
10  subject_property: proprieta_monitorata
11  thresholds:
12  # una o multiple terne con questo formato:
13  - status: stato
14  - operator: operatore_logico
15  - value: soglia_numerica
```

Listato 4.4: Formato semi-strutturato per caratterizzare i problemi

I campi type, subtype e subject_type sono obbligatori e richiedono valori scelti fra un elenco di keyword; ad essi si aggiungono campi facoltativi, fra cui alcuni già definiti. Il campo file_type richiede come valore uno dei sette tipi di file definiti nei sistemi Linux:

- regular_file
- directory
- character_device_file
- block_device_file
- local_socket_file
- named_pipe
- symbolic_link

Il modello può essere esteso con altri campi oltre quelli facoltativi già descritti, in modo da descrivere altri casi d'uso, purché non si creino ambiguità.
Nel listato 4.5 è mostrato come esempio un file di configurazione .yaml utilizzato nei test di sviluppo. In questo caso nell'host che ospita il file ci sono tre problemi monitorati:

- **controllo_nagios.ctrl**: problema binario di esistenza (esiste/non esiste) riguardo ad un file che è un regular_file, si chiama nagios.ctrl e si trova nel percorso /home/tellnet/sp/
- **controllo_spazio_disco**: problema discreto (riguarda un valore non binario) legato ad una soglia percentuale, riferito ad un file che è una directory, si chiama test e si trova nel percorso /; il problema riguarda nello specifico la memoria libera ed è caratterizzato da due soglie: la prima corrisponde ad uno status Critical ed è verificata

quando la memoria libera è minore di 0.3 (sotto il 30%), la seconda corrisponde ad uno status Warning ed è verificata quando la memoria libera è minore di 0.5 (sotto il 50%) esclusi valori che corrispondono all'altra soglia. Vale a dire che si tratta di un controllo sulla percentuale di memoria libera in un disco che ha per radice la directory /test.

- **controllo_test-file**: problema molto simile a controllo_nagios.ctrl che tuttavia si riferisce al file chiamato test-file

```
1  controllo_nagios.ctrl:
2    type: binary
3    subtype: exists
4    subject_type: file
5    file_type: regular_file
6    path: /home/tellnet/sp/s
7    filename: nagios.ctrl
8  controllo_spazio_disco:
9    type: discrete
10   subtype: percentage_thresholds
11   subject_type: file
12   file_type: directory
13   subject_property: free_memory
14   path: /
15   filename: test
16   thresholds:
17   - status: Critical
18   - operator: <
19   - value: 0.3
20   - status: Warning
21   - operator: <
22   - value: 0.5
23 controllo_test-file:
24   type: binary
25   subtype: exists
26   subject_type: file
27   file_type: regular_file
28   path: /home/tellnet/sp/
29   filename: test-file
```

Listato 4.5: Esempio di file di configurazione con 3 problemi monitorati: controllo_nagios.ctrl, controllo_spazio_disco, controllo_test-file

Utilizzo del modello

In ogni host monitorato deve essere presente un file, con estensione .yaml, che contenga le informazioni sulle caratteristiche di tutti i problemi monitorati nell'host, riportate rispettando la struttura data. Resta possibile aggiungere nuovi controlli sull'host, purché si vada ad aggiornare tale file di configurazione con i dati del nuovo controllo/problema.
Il file di configurazione viene richiesto (tramite chiamata SSH) e letto da TELLnet Core quando riceve la prima segnalazione per un problema che non conosce ancora.
I problemi sono identificati univocamente dalla terna (check_name, host_name, network), perciò su uno stesso host di una rete i problemi monitorati non possono essere identificati con lo stesso nome (check_name).

4.3.4 Elaborazione rimedi

Per i problemi conosciuti, inseriti e gestiti nel sistema fin dall'inizializzazione di TELLnet anche i rimedi devono essere già noti e inseriti. Tuttavia, per agevolare l'inserimento progressivo di nuovi controlli/problemi da gestire, i rimedi sono inseriti con una sintassi parametrizzata, permettendo così il riutilizzo per problemi simili ma con valori specifici diversi.
L'elaborazione di rimedi per problemi "nuovi" parte dalla ricezione della prima segnalazione di tale problema. A questo punto, non trovando corrispondenze nel database con i problemi conosciuti, TELLnet Core richiede il file di configurazione checks.yaml all'host su cui si è verificato il problema e sulla base della struttura del problema che trova lì descritta (vedi par. 4.3.3) avvia un algoritmo euristico che calcola la somiglianza del nuovo problema con quelli conosciuti; dai risultati dell'algoritmo sceglie il problema più simile fra quelli con rimedi compatibili e ne eredita i rimedi, sostituendo i valori dei parametri con quelli del nuovo problema.

Calcolo somiglianza

In sintesi, l'algoritmo di calcolo della somiglianza fra problemi prende in considerazione sia la struttura che i valori che descrivono i problemi, compresi i dati che descrivono l'host su cui il problema è monitorato. Da questi dati, confrontando il problema "nuovo" con ogni singolo problema conosciuto, vengono calcolate due somiglianze:

- **somiglianza strutturale**: riguarda quante feature (o campi) sono comuni ai due problemi e quante altre ne hanno non in comune e viene indicata come "half_sim".
- **somiglianza nei valori**: riguarda quante delle feature (o campi) in comune fra i due problemi hanno anche gli stessi valori e viene indicata come "full_sim".

Le due somiglianze vengono combinate, creando un elenco dei problemi conosciuti ordinato per somiglianza al "nuovo". Fra questi viene scelto il problema con valori di somiglianza più alto fra quelli con almeno un rimedio compatibile con il "nuovo" problema.
Un rimedio è compatibile con un problema se tale problema presenta nella propria struttura (vedi par. 4.3.3) tutti i parametri presenti nel comando eseguibile relativo al rimedio. Quindi i rimedi del problema scelto, compatibili col "nuovo, vengono ereditati da quest'ultimo (maggiori dettagli su questo processo sono presentati nel prossimo paragrafo: rimedi

ereditati).
Nel confronto delle feature e dei valori per calcolare "half_sim" e "full_sim" vengono utilizzati anche dei pesi, cioè dei valori numerici che indicano l'importanza di quella specifica feature nell'identificazione della somiglianza; tali pesi sono inizializzati tutti con lo stesso valore, poi vengono modificati a runtime nel momento in cui un rimedio ereditato ha successo o fallisce, influenzando i pesi relativi alle feature responsabili della sua scelta nel momento del calcolo della somiglianza.
La struttura delle tabelle del database coinvolte in questo processo è mostrata in figura 4.3.

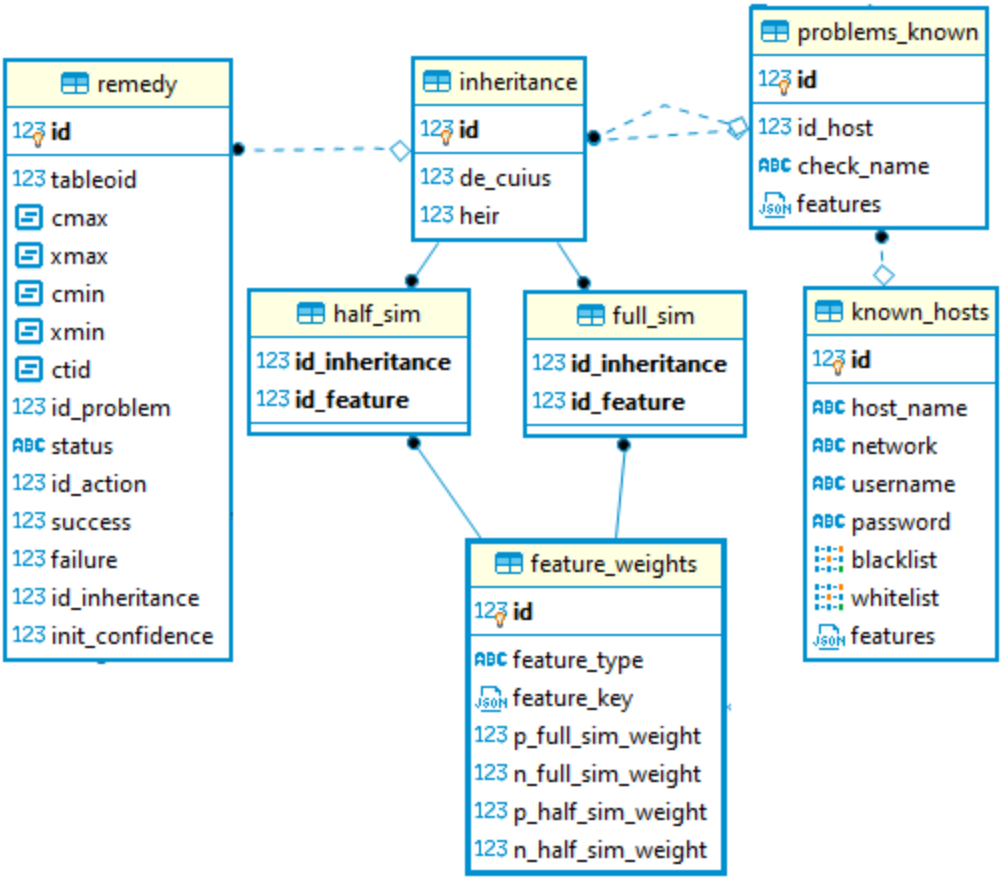

Figura 4.3: Schema E-R della porzione di database coinvolta nel calcolo della somiglianza

Rimedi ereditati

Una volta scelti i rimedi che il nuovo problema deve ereditare, restano da effettuare le operazioni necessarie per creare i nuovi rimedi a partire da quelli scelti, sostituendo le parti parametrizzate. A questo scopo vengono recuperati nel database gli "scheletri" di tali rimedi, ovvero la loro versione con i parametri non istanziati, la "sintassi parametrizzata".

Questa "sintassi parametrizzata" non è già presente completa in un dato nel database, bensì viene generata per giustapposizione e innesti di parti più elementari. In questo modo i comandi più semplici e comuni non vengono ripetuti nel database per ogni comando complesso che li contiene, permettendo il riutilizzo dei dati nonché un legame subito rilevabile fra comandi.
Sia il codice dei comandi eseguibili che la descrizione testuale in linguaggio naturale, che traduce per l'utente le azioni eseguite dal rimedio, contengono i parametri da istanziare identificati dal simbolo $ all'inizio e alla fine; fra questi simboli si trova il nome della feature da associare, il cui valore viene recuperato dai dati del nuovo problema.
Nelle tabelle 4.2 e 4.3 è illustrato l'effetto di questo processo ereditario con un esempio relativo ai problemi controllo_nagios.ctrl e controllo_test-file (già illustrati nel paragrafo 4.3.3) in cui il secondo eredita dal primo; nella tabella 4.2 è mostrato l'effetto sulla sintassi dei comandi eseguibili, nella tabella 4.3 quello sul testo che rappresenta la descrizione testuale in linguaggio naturale degli stessi comandi.

Sintassi (generata con innesti)	**Comando per controllo_nagios.ctrl**	**Comando per controllo_test-file**
mv $PATH$$FILENAME$ $PATH$$FILENAME_DEST$	mv /home/tellnet/sp/nagios.ctrl /home/tellnet/sp/nagios.ctrl_ORI	mv /home/tellnet/sp/test-file /home/tellnet/sp/test-file_ORI
rm -f $PATH$$FILENAME$	rm -f /home/tellnet/sp/nagios.ctrl	rm -f /home/tellnet/sp/test-file

Tabella 4.2: Esempio di processo ereditario dei rimedi (sintassi comandi)

Sintassi (generata con innesti)	**Rimedio per controllo_nagios.ctrl**	**Rimedio per controllo_test-file**
Rinomina $FILENAME$ in $FILENAME_DEST$	Rinomina nagios.ctrl in nagios.ctrl_ORI	Rinomina test-file in test-file_ORI
Cancella $FILENAME$ da $PATH$ senza bisogno di conferma	Cancella nagios.ctrl da /home/tellnet/sp/ senza bisogno di conferma	Cancella test-file da /home/tellnet/sp/ senza bisogno di conferma

Tabella 4.3: Esempio di processo ereditario dei rimedi (descrizione del comando in linguaggio naturale)

La struttura delle tabelle del database coinvolte in questo processo è mostrata in figura 4.4.

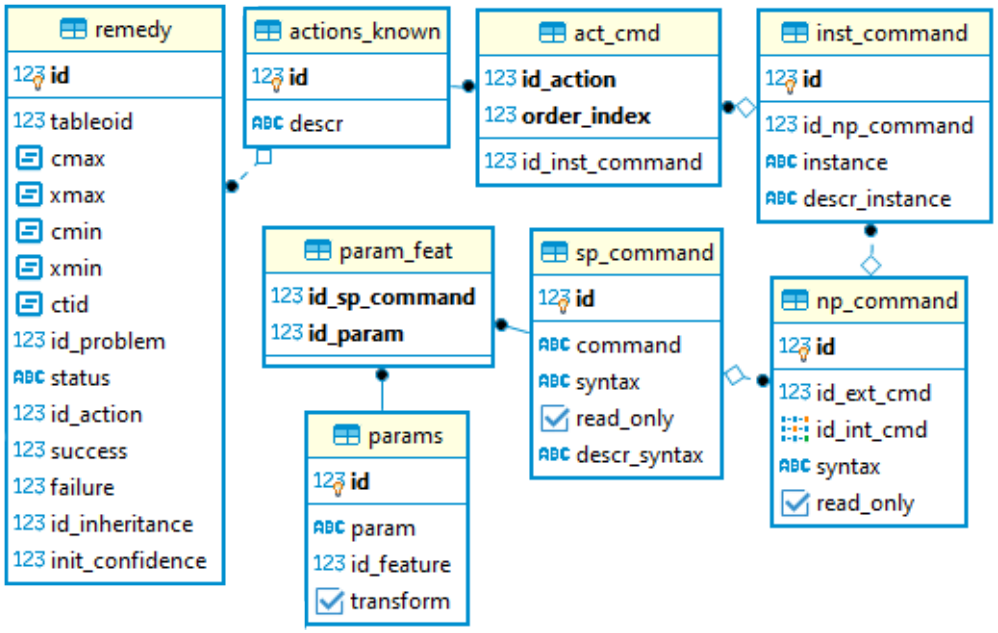

Figura 4.4: Schema E-R della porzione di database coinvolta nel calcolo dei rimedi ereditati

4.3.5 Redis

La funzione di Redis (IMDB descritto al par. 3.3.2) nell'architettura del BackEnd di TELLnet consiste sostanzialmente in due compiti:

- Fornire un sistema di gestione delle code di messaggi comune a tutti a tutte le parti che comunicano tramite Socket: TELLnet Core, server Flask e FrontEnd.
- Conservare in memoria principale le informazioni utili anche al FrontEnd allo scopo di renderle disponibili appena l'utente si autentica ed entra nella pagina principale: dati aggiornati sui problemi correnti e sulle azioni attive (rimedi proposti per le segnalazioni)

4.3.6 Flask

Nell'architettura del BackEnd di TELLnet il ruolo del server, tramite fra il FrontEnd e la logica applicativa (TELLnet Core), è svolto da Flask (descritto nel par. 3.2.1). Come web server, Flask rende accessibili dalla rete Internet le risorse che compongono l'interfaccia web.
Nella cartella templates si trovano le due pagine html:

- **login.html**: pagina per l'autenticazione
- **index.html**: pagina principale, con elenco dei problemi rilevati dal server e chatbot

Nella cartella static si trovano i file CSS (con le regole di stile per la formattazione e l'aspetto, che si sommano e talvolta sostituiscono quelle di default di Bootstrap, vedi par. 3.1.1) e i file Javascript con la logica di funzionamento del FrontEnd (animazioni, aggiornamenti a runtime e dialogo con il server tramite websocket):

- **signin.css**: regole di stile per la pagina di autenticazione (login.html)
- **home.css**: regole di stile per la pagina principale (index.html)
- **script.js**: contiene la logica del FrontEnd che gestisce le comunicazioni con il server tramite WebSocket, l'aggiornamento delle informazioni sui problemi segnalati (compresa la modifica dei dati nella tabella mostrata all'utente e nell'elenco delle notifiche), le richieste dell'utente con input di varia natura, dal classico mouse e tastiera all'ascolto tramite microfono, con riconoscimento vocale e trascrizione nella chat del chatbot, per interagire con esso vocalmente; gestisce anche le animazioni del chatbot, la riproduzione delle sintesi vocali per dare voce al chatbot e la richiesta al server di attivazione del contesto appropriato per il chatbot all'arrivo delle nuove segnalazioni.

L'aspetto del FrontEnd, risultante dai file sopra citati, e maggiori caratteristiche dell'esperienza utente sono descritti nel capitolo 5.

Oltre a rendere disponibili le risorse suddette, Flask svolge anche il ruolo di Socket server, permettendo la comunicazione con messaggi tramite Socket fra TELLnet Core e FrontEnd; offre servizi RESTful necessari al FrontEnd: servizio di autenticazione ed esecuzione dei webhook, sia su richiesta dell'Agent da Dialogflow (vedi par. 3.1.2) sia su richiesta del FrontEnd per interagire con le API di Dialogflow (e.g. attivare un contesto). Infine Flask, su richiesta del FrontEnd tramite Socket, interagisce con le API Google del servizio Text-to-Speech (vedi par. 3.1.2) per ottenere il risultato della sintesi vocale e inviarla al FrontEnd per dare voce al chatbot.

Capitolo 5

Progettazione e sviluppo FrontEnd

5.1 Obiettivi e scelte progettuali

Il FrontEnd del progetto TELLnet ne rappresenta la parte visibile a tutti, l'interfaccia che permette agli utenti di accedere alle sue funzionalità. Le sue caratteristiche hanno lo scopo di offrire un'esperienza utente soddisfacente con risultati il più possibile simili, se non migliori, rispetto alle richieste di assistenza gestite da operatori umani, automatizzando le operazioni di confronto necessarie per identificare nel dettaglio problemi e relative cause e per concordare la strategia risolutiva. A tale scopo sono state fatte scelte progettuali con l'obiettivo di dotare TELLnet di un'interfaccia pensata per essere user-friendly e allo stesso tempo completa e sicura, in grado di sostituire l'operatore umano per l'assistenza mettendo a disposizione dell'utente un insieme di metodi di comunicazione configurabili interoperativi che guidano l'utente verso la risoluzione del problema.
Le scelte progettuali principali riguardo al FrontEnd sono costituite da:

- Interfaccia web classica con interazione mouse e tastiera
- Chatbot testuale
- Riconoscimento vocale per dettare le proprie richieste al chatbot senza interagire con mouse e tastiera
- Sintesi vocale per ascoltare le risposte e le comunicazioni del chatbot senza leggere dalla sua tendina di conversazione (che riporta comunque quanto detto per permettere di leggere per intero la conversazione)

La scelta di affiancare un'interfaccia web classica e un chatbot deriva dalla volontà di offrire una esperienza utente semplice e allo stesso tempo capace di prendere in carico un complesso insieme di richieste dell'utente. Questa scelta, insieme alla scelta di offrire all'utente le opzioni del riconoscimento vocale e della sintesi vocale, segue anche una delle buone pratiche dell'HCI citate nel par. 2.3 (utilizzo di più risorse).
L'interfaccia web classica implementa un'altra buona pratica citata (non reinventare la

ruota) offrendo all'utente una forma d'interazione a cui è abituato.
Il chatbot permette anche di avere un metodo d'interazione la cui interfaccia non si complica all'aumentare delle funzionalità offerte, creando le basi per uno sviluppo scalabile nelle funzioni senza perderne in immediatezza e semplicità grafica.
La scelta di rendere solo opzionali le funzioni di riconoscimento vocale e sintesi vocale va nella direzione di un design che dia centralità all'utente e alle sue preferenze.
Al fine di garantire un'altra buona pratica (rendere leggibili gli elementi dell'interfaccia) l'interazione con il chatbot può essere ulteriormente sviluppata per interagire con l'interfaccia web classica laddove l'utente non capisca come utilizzarla o guidando l'utente a scoprire come e cosa può fare (al momento il chatbot può aprire la finestra di dialogo che mostra i rimedi).
La chat testuale del chatbot segue anche un'altra buona pratica (sostituire la memoria con informazioni visive) permettendo all'utente di consultare tutta la conversazione avvenuta anche quando interagisce vocalmente con il chatbot.
L'adeguatezza del chatbot come metodo d'interazione aggiuntivo per il progetto TELLnet è giustificata anche dall'utente atteso (in genere tecnico IT dell'azienda cliente) e dalla dimensione ridotta del dominio delle conversazioni utili che un'applicazione con le finalità di TELLnet deve saper gestire efficacemente: questo permette di concentrare gli sforzi sulle richieste più utili a questa specifica esperienza utente senza dare all'utente la sensazione di incapacità di dialogo su alcuni temi, dal momento che l'utente troverà (auspicabilmente) naturale che il chatbot lo aiuti invece a capire cosa può fare e lo inviti educatamente a fare richieste sensate per il contesto applicativo. Il chatbot di TELLnet non è "general purpose" e non vuole far intendere di esserlo.

5.2 Funzionalità

I metodi di comunicazione elencati nel par. 5.1, utilizzabili anche alternandoli o più di uno alla volta, permettono di usufruire delle funzionalità offerte da TELLnet tramite:

- Visualizzazione aggiornata degli eventuali problemi rilevati sui sistemi monitorati, mostrata nella pagina principale in una tabella aggiornata in tempo reale senza necessità di ricaricare la pagina. La riga posta immediatamente sopra la tabella mostra l'istante dell'ultimo aggiornamento ricevuto. La tabella, illustrata in figura 5.1, mostra il nome del problema, l'host su cui si è verificato, l'intervallo in secondi fra due controlli su questo problema, data e orario in cui è stato segnalato per la prima volta, data e orario dell'ultima segnalazione e stato attuale.

- Visualizzazione e scelta delle opzioni di risoluzione proposte (rimedi/comandi), ordinate dalla «migliore» (in base agli esiti del passato) e segnalate in rosso se già tentate precedentemente per questa occorrenza del problema (vedi figura 5.2). Tali informazioni vengono mostrate cliccando sulla riga di un problema o richiedendo al chatbot di aprirla (con varie possibili frasi, vedi par. 5.4). Una volta scelta l'opzione desiderata e confermata la scelta le opzioni per quel problema divengono non più disponibili fino alla verifica del loro esito; viene invece mostrato un messaggio che avvisa l'utente che il tentativo di risoluzione è in atto (vedi figura 5.3)

Figura 5.1: Esempio di problemi rilevati visualizzati nella pagina principale del FrontEnd

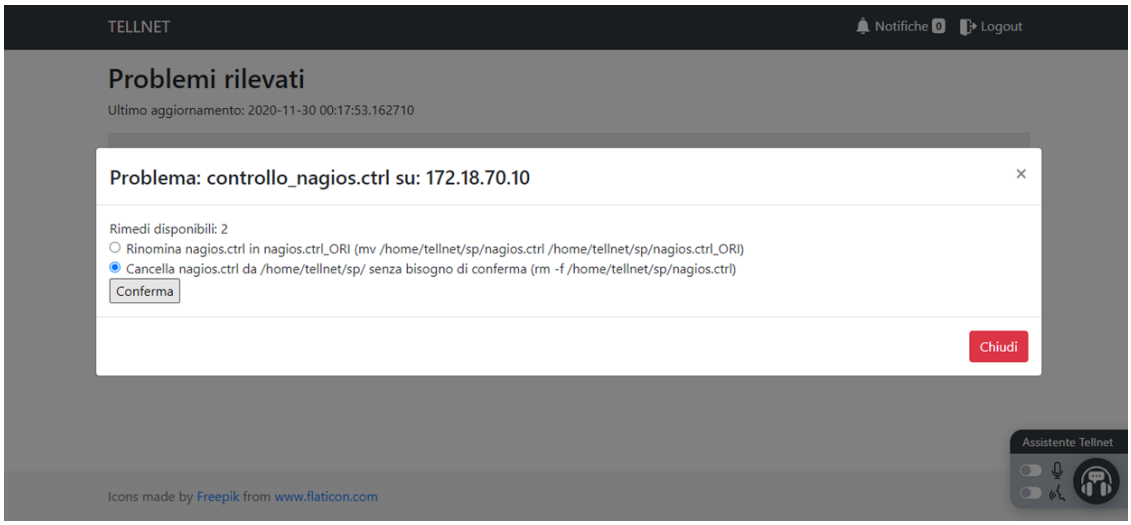

Figura 5.2: Finestra di dialogo per scelta opzioni di risoluzione

- Ricezione e visualizzazione di notifiche di fine tentativo di risoluzione contenenti informazioni sull'esito, con la possibilità di inviare feedback al server sulla correttezza dell'esito (e.g. l'esito può risultare risolutivo mentre il problema è di nuovo segnalato, magari perché il problema è scomparso per brevissimo tempo poi tornato subito a verificarsi). Un esempio di notifica è illustrato in figura 5.4.

5.3 Conversazioni col chatbot implementate

Le conversazioni che il chatbot è in grado di sostenere dipendono dagli Intent definiti nell'Agent (vedi par. 3.1.2), che a questo punto dello sviluppo è in grado di:

- Descrivere la situazione corrente sui problemi rilevati (se ci sono, quanti e quali)

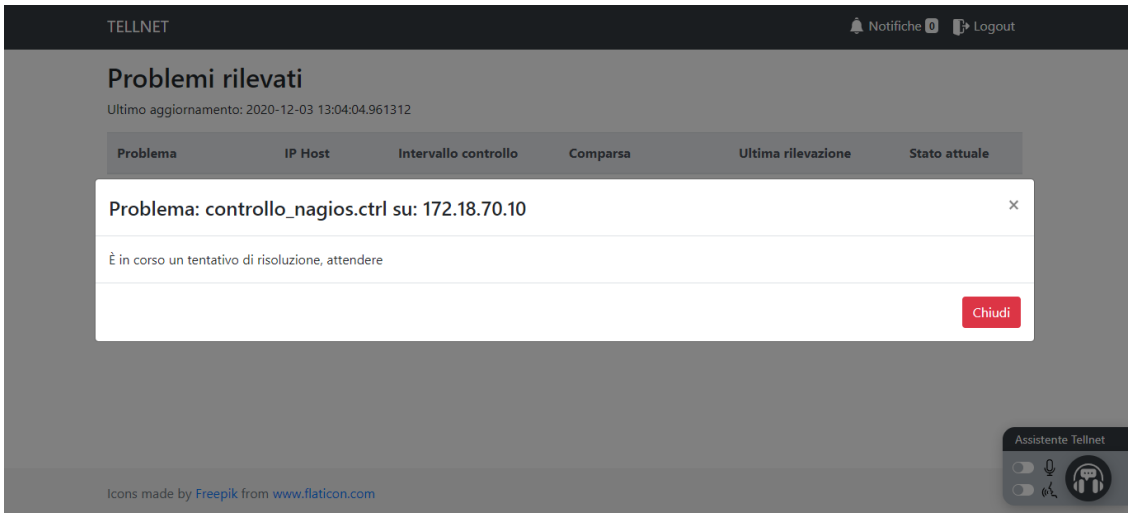

Figura 5.3: Messaggio mostrato per problemi con un tentativo di risoluzione in atto

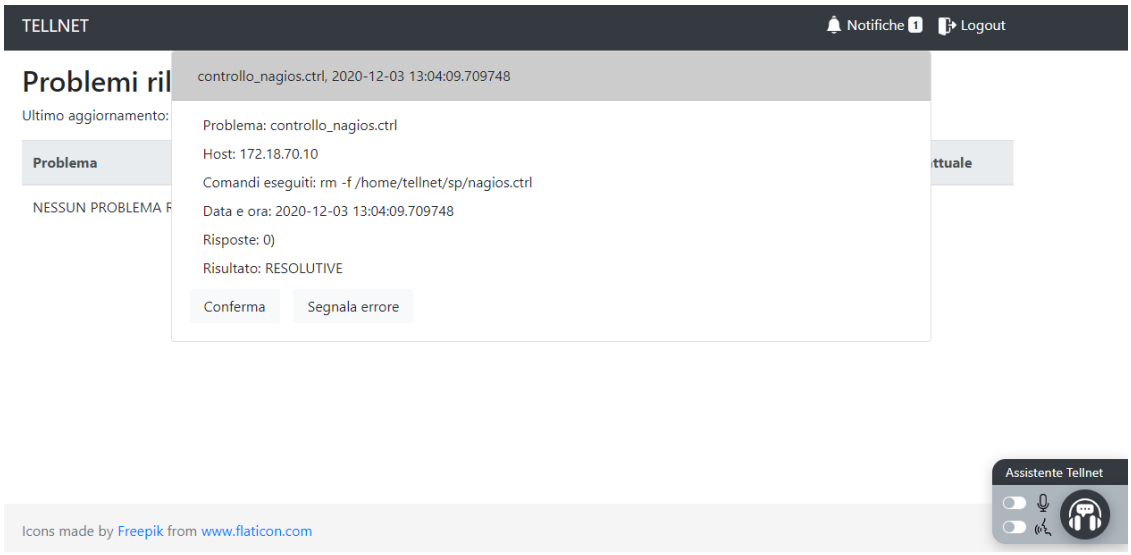

Figura 5.4: Notifica sull'esito di un tentativo di risoluzione

- Fornire maggiori informazioni su un problema scelto (al momento con un numero che riguarda l'ordine nella tabella visualizzata (vedi figura 5.1), ad esempio problema "uno" o "primo")
- Mostrare i rimedi proposti per un problema scelto, aprendo la finestra di dialogo
- Fornire informazioni varie sull'azienda (Tecnodata in questo caso) a seconda di quali vengano chieste (ad es: contatti generici o specifici)

- Gestire lo "SmallTalk", ovvero rispondere a frasi generiche ma ricorrenti nelle conversazioni (e.g. frasi di cortesia)

5.4 Funzionamento chatbot

L'interazione con il chatbot prende la forma di una conversazione nella tendina della chat. Il chatbot accoglie l'utente con un messaggio di saluto, quindi risponde ai messaggi inviati dall'utente oppure comunica messaggi a seguito di eventi: ad esempio quando arriva la segnalazione di un nuovo problema, oltre ad essere inserita nella tabella che le visualizza (vedi figura 5.1) viene anche comunicata dal chatbot (vedi figura 5.5).

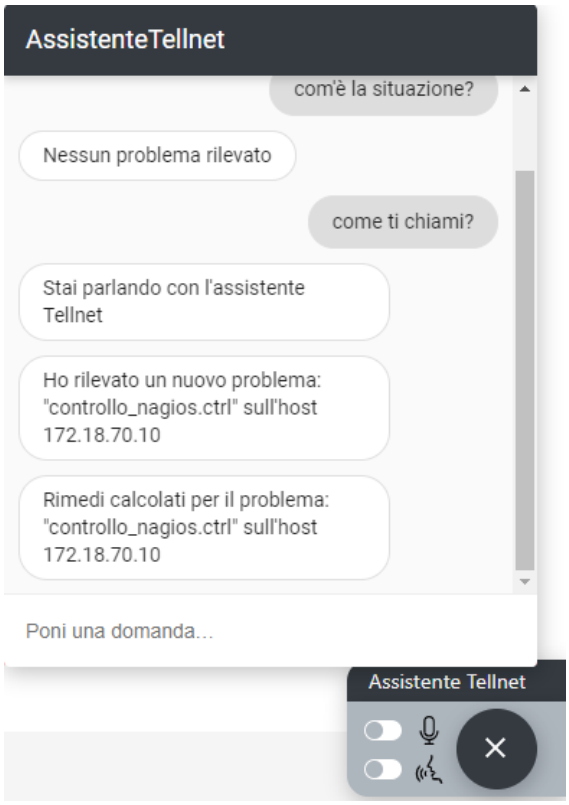

Figura 5.5: Esempio di conversazione con risposte all'utente e comunicazioni di eventi

I messaggi in arrivo quando il chatbot è ridotto a icona sono segnalati da una notifica numerica (vedi figura 5.6) e vengono comunque letti dalla voce del chatbot se è attiva la sintesi vocale.

5.4.1 Informazioni contestuali

Il meccanismo dei contesti di Dialogflow (vedi par. 3.1.2) permette al chatbot di desumere alcune informazioni mancanti (sottintese dall'utente) dal contesto, che può essere attivato

Figura 5.6: Il chatbot ha ricevuto un messaggio non ancora letto

dalle frasi precedenti nella conversazione o da eventi, recuperandole dai parametri istanziati nel contesto attivo. Nella figura 5.7 è illustrato un esempio di conversazione in cui, alla richiesta dell'utente "come posso risolverlo", l'informazione su quale sia il problema a cui si riferisce l'utente è ricavata dal contesto attivato subito prima dalla ricezione della segnalazione di tale problema.

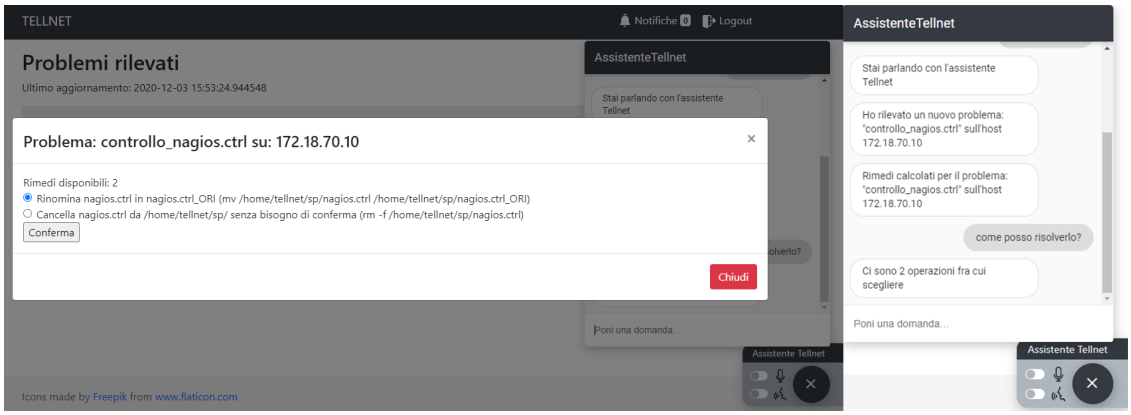

Figura 5.7: Esempio di uso del contesto per dedurre informazioni sottintese (a destra la conversazione, a sinistra la finestra di dialogo aperta in risposta)

5.4.2 Interazione vocale

L'interazione con il chatbot comprende sempre testo scritto, tuttavia l'utente può scegliere di attivare:

- **Riconoscimento vocale**: permette di dettare le proprie richieste al chatbot invece di scriverle con la tastiera
- **Sintesi vocale**: permette di ascoltare risposte e comunicazioni varie del chatbot senza doverle leggere nella chat, dove comunque vengono trascritte

Attivando entrambe le funzionalità suddette si può simulare una conversazione vocale con il chatbot.

Riconoscimento vocale

Il riconoscimento vocale permette di dettare le proprie richieste al chatbot; queste vengono trascritte in tempo reale nel campo di input della chat e inviate al termine della frase.
Se attivato, resta in ascolto anche con il chatbot ridotto a icona. Per evitare che la funzione resti attiva e in ascolto per dimenticanza, anche quando non è più utile, quando non viene rilevato nessuno che parli si attiva un countdown (visibile all'utente) al termine del quale la funzione viene disattivata, terminando l'ascolto del microfono. Nella figura 5.8 è illustrato l'aspetto del chatbot in ascolto.

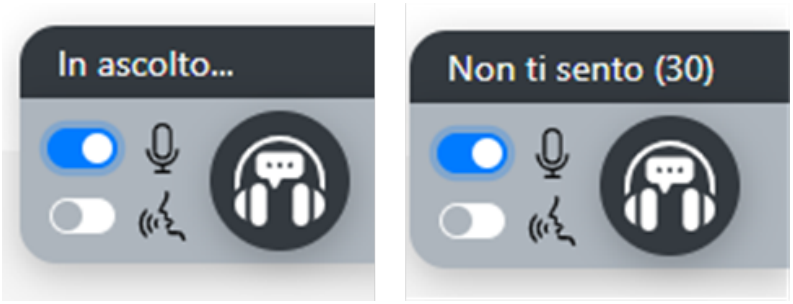

Figura 5.8: Chatbot in ascolto

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Conclusioni

Posto l'obiettivo di fondo di implementare un assistente digitale per offrire assistenza nella gestione e monitoraggio delle infrastrutture IT, si sono individuati quattro obiettivi principali da raggiungere. Di questi, tre si possono considerare raggiunti: infatti pur trattandosi ancora di un progetto in fase di sviluppo, che presenta perciò ampi margini di miglioramento, si sono poste le basi concettuali, architetturali, tecnologiche e implementative per continuare il processo di sviluppo concentrandosi solo sul miglioramento delle funzionalità e dell'esperienza utente.
Per quanto riguarda il primo obiettivo (raccogliere segnalazioni) la scelta progettuale riguardante la definizione del modello delle segnalazioni ha permesso una efficace implementazione della funzionalità senza rinunciare ad un pò di flessibilità.
Il terzo e il quarto obiettivo, strettamente legati (interfaccia web user-friendly e chatbot) sono stati raggiunti seguendo le buone pratiche di design HCI con un'approccio multiparadigma integrato che ha permesso di porre le basi per un'interfaccia utente molto semplice e comprensibile ma non limitata nelle funzionalità, anche in ottica di futuri sviluppi.
Il secondo obiettivo (elaborare rimedi) rappresenta una questione a parte, più complessa e delicata: si sono dovute compiere scelte progettuali non ottimali (in un'ottica di funzionalità ideale) per trovare un compromesso con gli ostacoli implementativi dovuti alla natura del problema e agli attuali sviluppi tecnologici e di standard. Tuttavia l'obiettivo è funzionalmente raggiunto, pur con qualche limite, perciò nel complesso del progetto ci si può ritenere soddisfatti. Per di più la natura modulare del progetto permette lo sviluppo di una componente senza inficiare il resto, per cui in futuro anche i limiti presenti potrebbero essere affrontati senza per questo dover stravolgere il resto del progetto.

6.2 Sviluppi futuri

Trattandosi di un progetto ampio, che chiama in causa molte tecnologie e con molte componenti, presenta molti aspetti da rifinire o talvolta da sviluppare ancora per estenderne le funzionalità. Il chatbot e in generale l'interfaccia possono vedere un notevole incremento

delle funzionalità offerte all'utente, allo scopo di ampliare l'insieme delle possibili richieste gestite. Come già detto, il metodo di elaborazione dei rimedi può essere rivisto in futuro, forti dell'esperienza accumulata dall'utilizzo del sistema e dei prevedibili sviluppi tecnologici. Sempre grazie all'esperienza accumulata, dal momento che TELLnet archivia e storicizza le proprie elaborazioni, si può prevedere di utilizzare tali informazioni per analisi dei dati e tentativi di approccio con machine learning supervisionato, allo scopo di identificare eventuali criticità nel funzionamento del sistema e di raffinare i metodi di elaborazione dei rimedi. Per finire, si può pensare di sviluppare modelli ad-hoc per i problemi monitorati più frequenti e delicati, valutando tecniche di machine learning supervisionato o non supervisionato oppure altre forme di intelligenza artificiale, a seconda della natura dei problemi stessi. Tali modelli potranno essere utilizzati implementando script ad-hoc richiamati come qualunque altro rimedio del sistema, senza dover modificare l'architettura e il funzionamento di TELLnet. La ricerca sull'utilizzo di queste tecnologie in questo campo applicativo è appena agli albori, solo il tempo ci dirà fin dove potrà cambiare le "regole del gioco".

Appendice A

Terminologia

- **API**: Application programming interface, ovvero interfaccia di programmazione delle applicazioni; sono set di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi. Consentono di sviluppare software che comunica con altri prodotti o servizi senza sapere come vengono implementati, semplificando così lo sviluppo e consentendo un netto risparmio di tempo e denaro. Durante la creazione di nuovi strumenti e prodotti o la gestione di quelli esistenti, le API offrono flessibilità, semplificano la progettazione, l'amministrazione e l'utilizzo, e garantiscono opportunità di innovazione
- **BackEnd**: parti di un sito Web o di un programma software che gli utenti non vedono; controparte del frontend, che si riferisce all'interfaccia utente di un programma o di un sito web
- **Comparto IT**: personale addetto al corretto funzionamento dell'infrastruttura informatica aziendale
- **CRUD**: Create, Read, Update, and Delete (crea, leggi, aggiorna ed elimina) sono le quattro funzioni di base che i modelli dovrebbero essere in grado di svolgere
- **Database relazionale**: tipo di database che utilizza una struttura che consente di identificare e accedere ai dati in relazione a un altro dato nel database. Spesso, i dati in un database relazionale sono organizzati in tabelle.
- **DBMS**: Database Management System (sistema di gestione di basi di dati), un sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di database, per questo detto anche "gestore o motore del database"
- **FrontEnd**: la parte di un sistema informatico o di un'applicazione con cui l'utente interagisce direttamente
- **GCP**: Google Cloud Platform, piattaforma di servizi professionali offerti da Google tramite Cloud
- **IMDB**: in-memory database, sistema di gestione di basi di dati che archivia i dati nella memoria principale per ottenere prestazioni più veloci[4][6]

- **IoT**: Internet of Things, la rete di oggetti fisici, "cose", che sono incorporati con sensori, software e altre tecnologie allo scopo di connettere e scambiare dati con altri dispositivi e sistemi su Internet. Questi dispositivi vanno dai normali oggetti domestici a sofisticati strumenti industriali
- **IT**: Information Technology, tecnologia dell'informazione, argomento ampio che riguarda tutti gli aspetti della gestione e dell'elaborazione delle informazioni, soprattutto all'interno di una grande organizzazione o azienda. L'IT non è generalmente utilizzato in riferimento a personal computer o home computing e networking. Sebbene l'IT sia spesso utilizzato per descrivere computer e reti di computer, in realtà include tutti i livelli di tutti i sistemi all'interno di un'organizzazione: dall'hardware fisico ai sistemi operativi, applicazioni, database, archiviazione, server e altro ancora. Anche le tecnologie di telecomunicazione, inclusi Internet e telefoni aziendali, fanno parte dell'infrastruttura IT di un'organizzazione
- **JSON**: JavaScript Object Notation, è un formato leggero per l'archiviazione e il trasporto dei dati; viene spesso utilizzato quando i dati vengono inviati da un server a una pagina Web; è "auto-descrittivo" e facile da capire
- **Keyword**: parola chiave
- **KPI**: Key Performance Indicator, sono gli indicatori (chiave) critici del progresso verso un risultato desiderato. I KPI forniscono un focus per il miglioramento strategico e operativo, creano una base analitica per il processo decisionale e aiutano a focalizzare l'attenzione su ciò che conta di più
- **Linguaggio naturale**: per linguaggio naturale si intende una qualunque lingua propria degli uomini (e.g. inglese, italiano, ...); l'espressione si utilizza per indicare che non si tratta di linguaggio "direttamente" comprensibile ad un calcolatore (e.g. linguaggi di programmazione); questa differenza è dovuta alle ambiguità che i linguaggi naturali possono generare
- **Machine Learning**: Apprendimento automatico; si trattta di un'applicazione dell'intelligenza artificiale che fornisce ai sistemi la capacità di apprendere e migliorare automaticamente dall'esperienza senza essere programmati esplicitamente. L'apprendimento automatico si concentra sullo sviluppo di programmi per computer in grado di accedere ai dati e utilizzarli per apprendere da soli
- **Message queue**: La coda dei messaggi è una forma di comunicazione service-to-service asincrona. I messaggi vengono salvati nella coda finché non vengono elaborati ed eliminati. Ogni messaggio viene elaborato una sola volta, da un singolo consumatore.
- **Modello client/server**: Il modello client-server è una struttura applicativa distribuita che suddivide attività e carichi di lavoro tra i fornitori di una risorsa o un servizio, chiamati server, e i richiedenti del servizio, chiamati client. Spesso client e server comunicano su una rete di computer su hardware separato. Un host server esegue uno o più programmi server, che condividono le proprie risorse con i client. I client, avviano sessioni di comunicazione con i server, che attendono le richieste in arrivo

- **NOC**: Network Operations Center, posizione centralizzata in cui i tecnici IT supportano direttamente gli sforzi del software di monitoraggio e gestione remoto
- **Open source**: il termine open source si riferisce a qualcosa che le persone possono modificare e condividere perché il suo design è pubblicamente accessibile. Il software open source è un software con codice sorgente che chiunque può ispezionare, modificare e migliorare
- **Operazioni di recovery**: Insieme delle operazioni volte a ripristinare uno stato di funzionamento normale
- **REST**: REpresentational State Transfer, è uno stile di architettura per fornire standard tra i sistemi di computer sul Web, rendendo più facile la comunicazione tra i sistemi. I sistemi conformi a REST, spesso chiamati sistemi RESTful, sono caratterizzati dall'assenza di stato e separano i compiti del client e del server. REST richiede che un client effettui una richiesta al server per recuperare o modificare i dati sul server. Una richiesta generalmente consiste in: un HTTP verb (definisce il tipo di operazione da eseguire), un'intestazione(consente al client di passare le informazioni sulla richiesta), un percorso (path) verso una risorsa, un corpo del messaggio opzionale contenente dati
- **RESTful**: sistema conforme all'architettura REST (REpresentational State Transfer)
- **Rimedio**: nel contesto di questo progetto per "rimedio" si intende una sequenza di uno o più comandi eseguibili da un sistema operativo, pensata per rimediare (appunto) ad un problema sorto nel sistema
- **Script**: programma o sequenza di istruzioni che viene interpretata o eseguita da un altro programma piuttosto che dal processore del computer (come lo è un programma compilato); il termine script viene anche utilizzato per indicare un elenco di comandi del sistema operativo pre-memorizzati in un file ed eseguiti in sequenza dall'interprete dei comandi del sistema operativo ogni volta che il nome dell'elenco viene immesso come un singolo comando
- **SLA**: Service Level Agreement, accordo sul livello di servizio che definisce il livello di servizio che ci si aspetta da un fornitore, indicando le metriche in base alle quali il servizio viene misurato, nonché i rimedi o le penalità nel caso in cui i livelli di servizio concordati non vengano raggiunti. È una componente fondamentale di qualsiasi contratto con un fornitore di tecnologia
- **Software di monitoring**: programma che monitora costantemente lo stato dell'hardware e/o del software su cui viene attivato; i singoli programmi possono utilizzare metodi diversi (controlli ciclici, sensori, ...)
- **SQL**: Structured Query Language, linguaggio di programmazione utilizzato per comunicare con i dati memorizzati in un sistema di gestione di database relazionali. La sintassi SQL è simile alla lingua inglese, il che rende relativamente facile scrivere, leggere e interpretare

- **Tecnodata srl**: azienda presso la quale si è svolto il tirocinio durante il quale è nato il progetto Tellnet
- **TELLnet**: nome scelto per il progetto nel suo complesso
- **Thread**: singolo flusso sequenziale di controllo all'interno di un programma; i vantaggi dell'uso dei thread non riguardano un singolo thread sequenziale, bensì l'uso di più thread in esecuzione contemporaneamente e che eseguono attività diverse in un unico programma
- **Timestamp**: sequenza di caratteri o informazioni codificate che identificano quando si è verificato un determinato evento, di solito fornendo data e ora del giorno, a volte accurate fino a una piccola frazione di secondo
- **Troubleshooting**: forma di risoluzione dei problemi, spesso applicata per riparare prodotti o processi guasti su una macchina o un sistema. È una ricerca logica e sistematica dell'origine di un problema per risolverlo e rendere nuovamente operativo il prodotto o processo e un'operazione necessaria per identificare i sintomi. Determinare la causa più probabile è un processo di eliminazione, ovvero l'eliminazione delle potenziali cause di un problema. Infine, la risoluzione dei problemi richiede la conferma che la soluzione ripristini il prodotto o il processo al suo stato di funzionamento
- **User-friendly**: di facile utilizzo per l'utente finale; il più possibile autoesplicativo
- **WebRTC**: Web Real-Time Communications, framework aperto per il Web che abilita le funzionalità di comunicazione in tempo reale (RTC) nel browser
- **WSGI**: Web Server Gateway Interface; è una specifica che descrive come un server web comunica con le applicazioni web e come le applicazioni web possono essere concatenate insieme per elaborare una richiesta. WSGI è uno standard Python descritto in dettaglio nelle specifiche PEP 3333

Appendice B

Linguaggi utilizzati e Software citati

B.1 Software di monitoring

B.1.1 Nagios

Nagios è una potente suite software di gestione e monitoraggio di infrastrutture IT che consente alle organizzazioni di identificare e risolvere i problemi prima che influenzino i processi aziendali critici. Nei test del progetto TELLnet è stato utilizzato Nagios Core (componente principale di Nagios) come software di monitoring per inviare segnalazioni a TELLnet Core. Nagios Core rientra fra gli standard del settore dei software di monitoraggio IT[5]. Il pluripremiato motore Nagios Core è stato lo standard di fatto nel monitoraggio dell'infrastruttura di rete per oltre un decennio, fornendo prestazioni e flessibilità senza precedenti; costituisce la componente fondamentale della suite software Nagios XI.
Nagios Core è un software open source, con una comunità di supporto attiva.
Nagios esegue script bash personalizzabili, fra i quali si illustra nel listato B.1 quello responsabile dell'invio dei dati per la segnalazione del problema controllo_nagios.ctrl a TELLnet, scritto rispettando il modello delle segnalazioni richiesto da TELLnet (vedi 4.3.2).

```
1 #!/bin/bash
2 DATA=$(date "+%d/%m/%y␣%H:%M:%S")
3 NET=tn01ng
4 HOST="172.18.70.10"
5 CONTROLLO=controllo_nagios.ctrl
6 FILE=/home/tellnet/sp/nagios.ctrl
7 if [ -f /home/tellnet/sp/nagios.ctrl ]; then
8     echo "Il␣file␣esiste!"
9     echo $VALUES
```

```
10      /usr/bin/psql -h 172.18.70.11 -p 5432 -U tellnet -d
           tellnetdb -t -c '\x' -c "INSERT␣INTO␣unsolved_problems␣
           VALUES␣('$CONTROLLO',␣'$HOST',␣'$NET',␣60,␣'$DATA',␣
           'Critical',␣'{\"exists\":␣true}')"
11      echo "Critical␣-␣Verificare␣$FILE␣|␣STATUS=CRITICAL"
12      exit 2
13
14  else
15      echo "OK␣-␣Il␣file␣nagios.ctrl␣non␣esiste␣|␣STATUS=OK"
16      if [[ -z "$1" ]]; then
17          echo "parametro␣service-status␣non␣passato␣(o␣vuoto)"
18      else
19          echo "precedente␣service-status:␣$1"
20          if [[ "$1" ==  "OK" ]]; then
21              echo "era␣giÃ ␣OK"
22          else
23              echo "problema␣risolto"
24              /usr/bin/psql -h 172.18.70.11 -p 5432 -U tellnet
                   -d tellnetdb -t -c '\x' -c "INSERT␣INTO␣
                   unsolved_problems␣VALUES␣('$CONTROLLO',␣
                   '$HOST',␣'$NET',␣60,␣'$DATA',␣'OK',␣
                   '{\"exists\":␣false}')"
25          fi
26      fi
27      exit 0
28  fi
29  }
```

Listato B.1: Script bash che segnala il problema controllo_nagios.ctrl

B.2 Linguaggi

B.2.1 Javascript

JavaScript è un linguaggio di programmazione interpretato (anche detto di scripting) con sintassi di alto livello. Consente di implementare funzionalità complesse sulle pagine web, visualizzando aggiornamenti tempestivi dei contenuti, mappe interattive, animazioni grafiche 2D e 3D, e molto altro. Si tratta di una delle tecnologie chiave del WWW (World Wide Web, la rete Internet globale); è un linguaggio multi-paradigma, supporta infatti diversi stili di programmazione (funzionale, imperative e orientato ad eventi).
Nel progetto TELLnet Javascript è utilizzato nello sviluppo del FrontEnd per supportare animazioni, aggiornamenti e comunicazione con il server Flask (dove in effetti risiedono gli script Javascript, nella cartella static, riconoscibili dall'estensione .js).

B.2.2 HTML

HTML (HyperText Markup Language) è il linguaggio di markup utilizzato per strutturare e dare significato ai contenuti web, ad esempio definendo paragrafi, titoli e tabelle di dati o incorporando immagini e video nella pagina. Viene utilizzato insieme ad altri linguaggi: CSS (Cascading Style Sheets) per definire le regole di stile (aspetto) da dare ai contenuti e linguaggi di scripting per la logica applicativa (in genere JavaScript). I browser web richiedono i file HTML al server e li trasformano in una pagina web multimediale comprensibile da un umano.

B.2.3 CSS

CSS (Cascading Style Sheets) è un linguaggio di regole di stile utilizzato per applicare lo stile, cioè l'aspetto, ai contenuti delle pagine di markup come HTML, ad esempio impostando i colori di sfondo e i font (tipi di carattere) e disponendo il contenuto in più colonne. CSS è una delle tecnologie di base del WWW (World Wide Web, la rete Internet globale) insieme al linguaggio di markup HTML e il linguaggio di programmazione JavaScript. CSS è progettato per separare il contenuto informativo del documento dallo stile (layout, colori, font, ecc.). Questa separazione fornisce maggiore flessibilità e controllo per sviluppo e manutenzione. La possibilità di condividere la formattazione tra più pagine web permette di ridurre la complessità e la ripetizione della struttura, incentivando quindi il riutilizzo.

B.2.4 Python

Python è un linguaggio di programmazione di alto livello interpretato (anche detto di scripting), orientato agli oggetti con semantica dinamica. La sintassi semplice e di facile apprendimento di Python enfatizza la leggibilità e quindi riduce i costi di manutenzione del programma. Python supporta numerosi moduli e pacchetti, che incoraggiano la modularità del programma e il riutilizzo del codice. L'interprete Python e la vasta libreria standard sono disponibili in formato sorgente o binario gratuitamente per tutte le principali piattaforme e possono essere distribuiti liberamente.
Nel progetto TELLnet Python è utilizzato nello sviluppo del BackEnd, sia per il server Flask (che è appunto una libreria Python) sia per la componente TELLnet Core, che contiene la logica applicativa. Uno dei motivi per la scelta di Python come linguaggio di sviluppo del BackEnd di TELLnet riguarda la vasta disponibilità di librerie Python per il machine learning, che si prevede (fin dall'inizio) potrebbe diventare una componente importante del progetto negli sviluppi futuri.

Bibliografia

[1] James F. Allen et al. «Toward Conversational Human-Computer Interaction». In: *AI Magazine* 22.4 (dic. 2001), p. 27. DOI: 10.1609/aimag.v22i4.1590. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/1590.

[2] ROBERT DALE. «The return of the chatbots». In: *Natural Language Engineering* 22.5 (2016), pp. 811–817. DOI: 10.1017/S1351324916000243.

[3] Asbjørn Følstad e Petter Bae Brandtzæg. «Chatbots and the New World of HCI». In: *Interactions* 24.4 (giu. 2017), pp. 38–42. ISSN: 1072-5520. DOI: 10.1145/3085558. URL: https://doi.org/10.1145/3085558.

[4] H. Garcia-Molina e K. Salem. «Main memory database systems: an overview». In: *IEEE Transactions on Knowledge and Data Engineering* 4.6 (1992), pp. 509–516.

[5] J. Hernantes, G. Gallardo e N. Serrano. «IT Infrastructure-Monitoring Tools». In: *IEEE Software* 32.4 (2015), pp. 88–93. DOI: 10.1109/MS.2015.96.

[6] Peter Lake e Paul Crowther. «In-Memory Databases». In: *Microsoft Academic* (2013), pp. 183–197. URL: https://academic.microsoft.com/paper/106201864.

[7] P. Mendonca. *Human-Computer Interaction: What Is It, Why Does It Matter & Best Practices*. 12 Mag. 2020. URL: https://exaud.com/human-computer-interaction/.

[8] Optanix. *What is State-of-the-Art in IT Infrastructure Diagnostics?* 20 Mar. 2018. URL: https://www.optanix.com/what-is-state-of-the-art-in-it-infrastructure-diagnostics/.

[9] Matteo Zubani, Serina Ivan e Alfonso Emilio Gerevini. «A Comparison of Natural Language Understanding Services to build a chatbot in Italian». In: *Proceedings of the 4th Workshop on Natural Language for Artificial Intelligence (NL4AI 2020) co-located with the 19th International Conference of the Italian Association for Artificial Intelligence (AIIA 2020)* (nov. 2020). URL: CEUR-WS.org/Vol-2735/paper36.pdf.