Capitolo 5: Architettura del Sistema

Questo capitolo illustra l'architettura complessiva del sistema del chatbot AI (sistema automatizzato di conversazione basato su Intelligenza Artificiale) per il supporto tecnico, descrivendo in dettaglio i componenti principali, le loro interazioni e le scelte strutturali che hanno guidato lo sviluppo. L'architettura è stata progettata per garantire modularità, manutenibilità ed estensibilità, consentendo al sistema di evolvere in risposta a nuove esigenze o tecnologie.

5.1 Overview dell'Architettura Complessiva

Il sistema è stato progettato secondo un'architettura a microservizi modulare (approccio che suddivide l'applicazione in componenti indipendenti e specializzati), con componenti aventi basso accoppiamento (minima dipendenza tra i moduli) che comunicano attraverso interfacce ben definite. Come ogni architettura che si rispetti è organizzata in tre livelli principali:

1. **Livello di Presentazione**: Gestisce le interazioni con gli utenti attraverso l'interfaccia della piattaforma proprietaria Konsolex (sistema operativo cloud unificato sviluppato da OnTheCloud), principalmente, e con l'interfaccia di Telegram (applicazione di messaggistica istantanea).
2. **Livello Applicativo**: Contiene la logica di business, l'elaborazione AI (Intelligenza Artificiale) delle richieste e l'orchestrazione delle operazioni sia amministrative che lato client.
3. **Livello Dati**: Gestisce la persistenza delle informazioni e l'accesso ai dati esterni tramite API (interfacce che permettono a diversi software di comunicare tra loro).

La separazione di questi livelli garantisce che modifiche in una parte del sistema non influenzino necessariamente le altre, permettendo evoluzione e manutenzione indipendenti.

Il file `index.ts` fornisce il punto di ingresso principale dell'applicazione, inizializzando i componenti fondamentali e orchestrando l'avvio del sistema.

5.2 Moduli Principali e Loro Interazioni

Il sistema è composto da diversi moduli principali che collaborano per fornire le funzionalità complete. Questa organizzazione modulare facilita la manutenzione, il testing e l'estensione del sistema.

5.2.1 Web Server & REST API

Definito in `web-server.ts`, questo modulo espone API REST (interfacce web standardizzate) per l'interazione con il frontend (interfaccia utente) amministrativo e quello lato client, supportando funzioni di monitoraggio e gestione.

5.2.2 OpenAI Handler

Il modulo OpenAI Handler, definito principalmente in `openai-handlers.ts` e `openai-tool.ts`, si occupa dell'integrazione con l'API OpenAI (interfaccia fornita da OpenAI per accedere ai suoi modelli di intelligenza artificiale), gestendo thread conversazionali (conversazioni organizzate in sequenze), richieste di function calling (meccanismo che permette all'AI di richiamare funzioni specifiche) e interpretazione delle risposte AI.

5.2.3 Endpoint Handler

Questo modulo, implementato in `endpoint.ts`, fornisce un'astrazione per le interazioni con le API esterne della piattaforma Konsolex, permettendo al sistema di eseguire operazioni tecniche come riavvio server, gestione domini e altre funzioni amministrative.

5.2.4 Sistema di Ticketing

Implementato principalmente in `ticketsCashe.ts`, gestisce il ciclo di vita dei ticket di supporto (richieste di assistenza), inclusa creazione, stato e risoluzione.

5.2.5 Telegram Bot

Implementato nei file `bot.ts` e `admin-commmand.ts`, questo modulo gestisce l'interfaccia utente tramite la piattaforma di messaggistica Telegram, processando i comandi e inviando risposte formattate. È il punto di contatto alternativo tra gli utenti e il sistema, che permette agli utenti di utilizzare il sistema da remoto.

5.2.6 Data Repositories

Rappresentati dai file nella directory `repositories`, questi componenti implementano il pattern Repository (modello di progettazione che separa la logica di accesso ai dati dalla logica di business) per fornire un'astrazione sull'accesso ai dati persistenti.

5.2.7 Interazioni tra Moduli

I moduli comunicano attraverso interfacce ben definite, riducendo l'accoppiamento e aumentando la testabilità. Ad esempio, quando un messaggio arriva tramite l'interfaccia della piattaforma Konsolex:

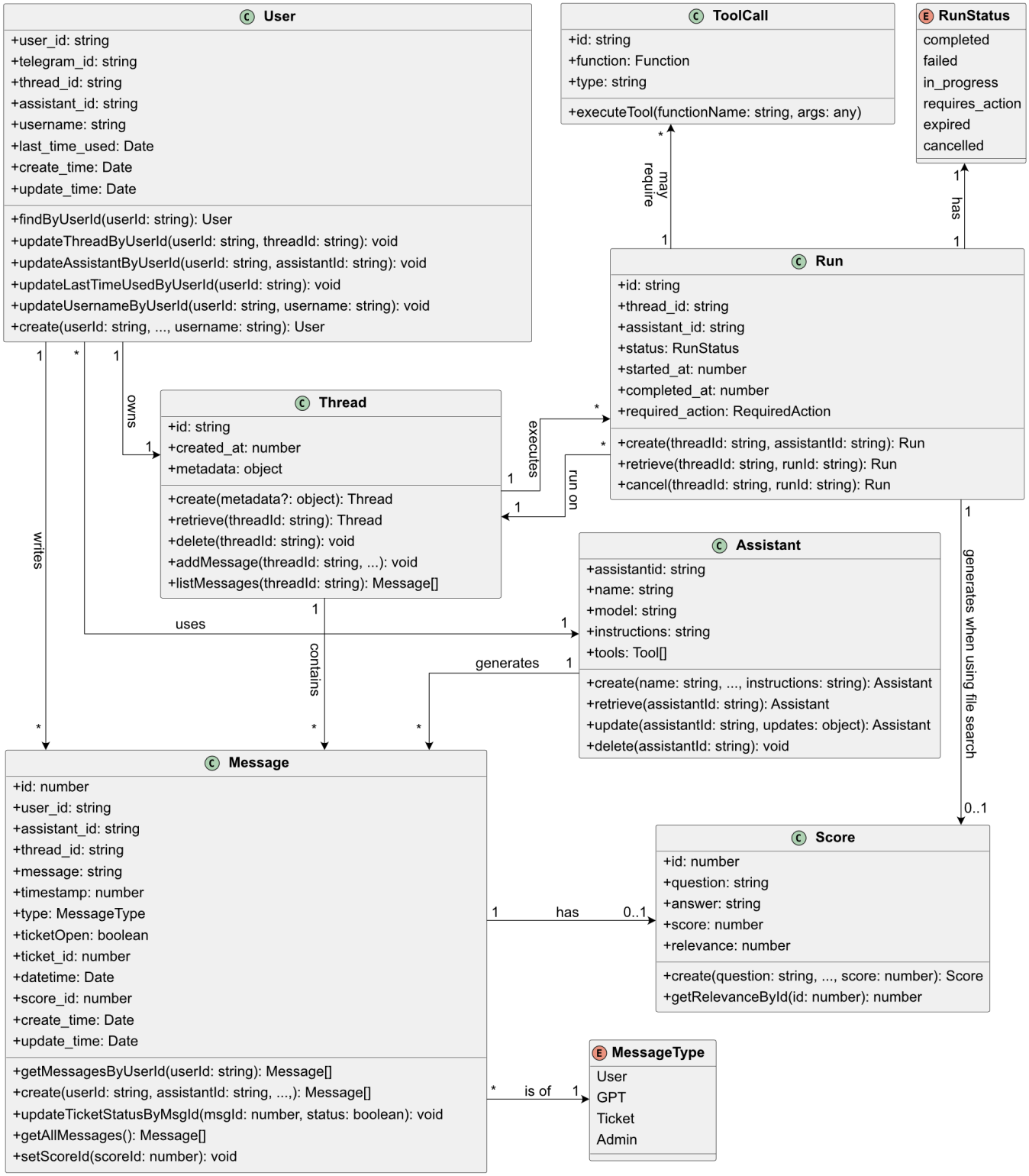
1. Il Bot riceve il messaggio
2. La funzione `storeMessageAndCreateReply` in `aux-functions.ts` gestisce la persistenza e l'elaborazione
3. OpenAI Handler genera una risposta utilizzando l'API GPT-4
4. Se necessario, vengono chiamate funzioni specifiche attraverso l'Endpoint Handler
5. La risposta viene formattata e inviata all'utente

Questa separazione di responsabilità segue il principio Single Responsibility (principio secondo cui ogni modulo dovrebbe avere una sola responsabilità) e facilita l'evoluzione indipendente dei componenti.

5.3 Diagrammi Architetturali

5.3.1 Class Diagram

Il diagramma delle classi (rappresentazione grafica delle classi e delle loro relazioni in un sistema orientato agli oggetti) illustra le principali entità del sistema e le loro relazioni, evidenziando la struttura statica dell'architettura software.



Le principali classi del sistema includono:

- **User**: Rappresenta un utente del sistema con attributi come `user_id`, `telegram_id`, e `thread_id`. Offre funzionalità per la gestione degli utenti, inclusi metodi per trovare, aggiornare e creare profili utente.
- **Message**: Modella i messaggi scambiati nella piattaforma, differenziati per tipologia (utente, GPT, ticket, admin). Include attributi come `message`, `timestamp`, e `ticketOpen`. Consente l'accesso ai messaggi di un utente e la gestione dello stato dei ticket.

- **Assistant**: Gestisce un assistente virtuale associato agli utenti. Nello stato attuale sono presenti solo due assistenti: uno per l'ambiente di test e uno per la gestione dei client.
- **Score**: Memorizza metriche di valutazione delle risposte AI, con attributi come `question`, `answer`, e `relevance`. Permette di calcolare gli score delle risposte dell'AI in base alle domanda del client.
- **Thread**: Gestisce i thread conversazionali di OpenAI, con attributi come `id`, `user_id`, e `expired`. Offre metodi per la creazione, l'aggiunta di messaggi e il reset del thread. Usato per tenere l'attenzione dell'assistente attiva.
- **Run**: Rappresenta un'esecuzione di un assistant su un thread, con attributi come `status`, `tool_calls`, e `required_action`. Implementa la logica per avviare l'esecuzione, recuperare risposte e gestire le chiamate agli strumenti.
- **ToolCall**: Richiama le richieste di esecuzione di funzioni specifiche da parte dell'AI, con attributi come `function` e `type`. Fornisce il metodo `executeTool` per l'esecuzione di operazioni concrete.

Il sistema utilizza anche enumerazioni come **MessageType** (User, GPT, Ticket, Admin) e **RunStatus** (completed, failed, in_progress, requires_action, expired, cancelled) per gestire gli stati e le tipologie di entità.

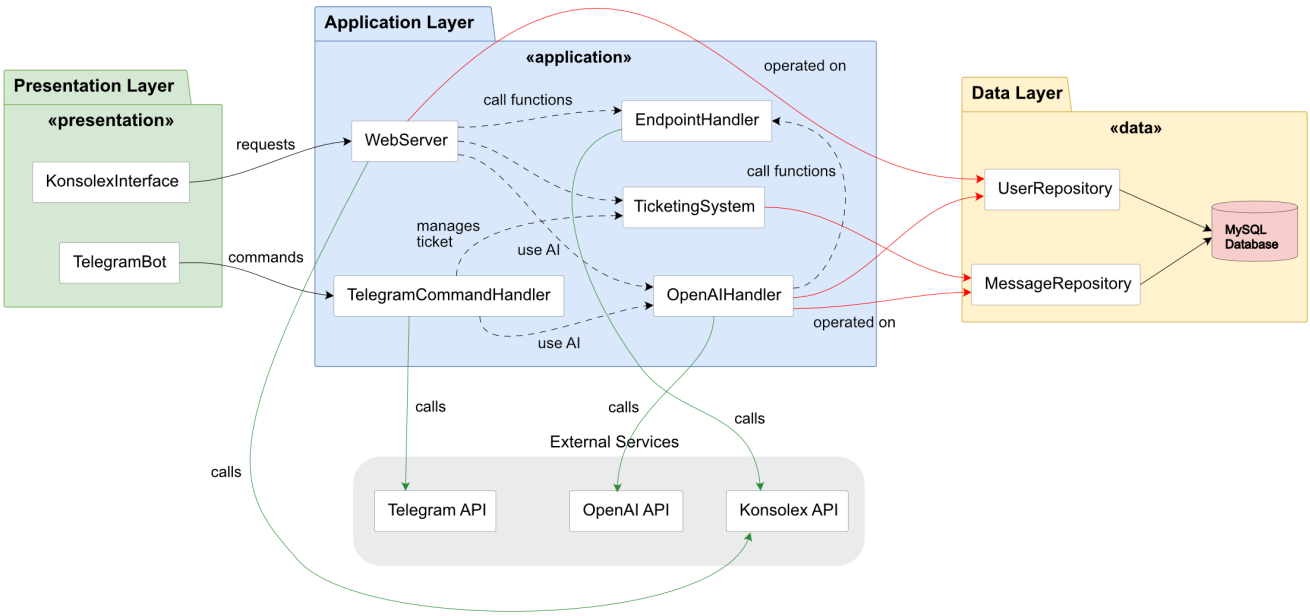
Le relazioni chiave tra queste classi riflettono la struttura logica del dominio:

- Un utente possiede un thread e può scrivere molti messaggi
- Un thread contiene molti messaggi ed esegue molte run
- Un assistant processa molte run e genera molti messaggi
- Una run può richiedere molti tool call e generare uno score quando utilizza la ricerca nei file
- Un messaggio può avere opzionalmente uno score di rilevanza
- I messaggi di ticket possono essere correlati tra loro

Questa organizzazione orientata agli oggetti riflette le entità principali del sistema OpenAI (thread, run, assistant) integrate con le entità specifiche dell'applicazione (user, message, score), creando un'architettura coesa che supporta il flusso conversazionale e l'esecuzione di operazioni tecniche.

5.3.2 Component Diagram

Il diagramma dei componenti (rappresentazione grafica che mostra l'organizzazione e le dipendenze tra i componenti software) visualizza i blocchi funzionali del sistema e le loro dipendenze, fornendo una rappresentazione chiara delle interazioni tra i vari moduli dell'architettura.



Il diagramma è organizzato in tre livelli distinti, ciascuno rappresentato con un codice colore specifico per una migliore leggibilità:

- **Livello di Presentazione** (verde): Comprende i componenti TelegramBot e KonsolexInterface che forniscono i punti di ingresso per l'interazione degli utenti con il sistema.
- **Livello Applicativo** (blu): Include i componenti principali che implementano la logica di business, come OpenAIHandler, EndpointHandler, TicketingSystem, WebServer, AuxiliaryFunctions e TelegramCommandHandler.
- **Livello Dati** (giallo): Contiene i repository (UserRepository, MessageRepository, ScoreRepository) che gestiscono l'accesso al database MySQL sottostante.

Le interazioni tra i componenti sono rappresentate con diversi stili di frecce per evidenziare la natura delle relazioni:

- **Linee continue**: Connessioni dal livello di presentazione al livello applicativo, rappresentando il flusso primario delle richieste utente.
- **Linee tratteggiate**: Interazioni tra i componenti del livello applicativo, mostrando la collaborazione interna per l'elaborazione delle richieste.
- **Linee rosse**: Connessioni dal livello applicativo al livello dati, indicando le operazioni di accesso ai dati.
- **Linee verdi**: Connessioni ai servizi esterni (OpenAI API, Konsolex API, Telegram API), mostrando l'integrazione con sistemi di terze parti.

Il diagramma evidenzia diversi pattern di interazione chiave:

1. La **KonsolexInterface** e il **TelegramBot** instradano i comandi attraverso il **WebServer** e **TelegramCommandHandler**, che utilizzano a loro volta l'**OpenAIHandler** per le risposte intelligenti.
2. L'**OpenAIHandler** rappresenta il nucleo dell'elaborazione AI, comunicando con l'API OpenAI per generazioni di risposte, utilizzando le **AuxiliaryFunctions** per operazioni di supporto e accedendo ai

repository per la persistenza dei dati.

3. Il **WebServer** fornisce un'interfaccia REST per le comunicazioni frontend, orchestrando le interazioni tra i vari componenti applicativi e i repository dati.

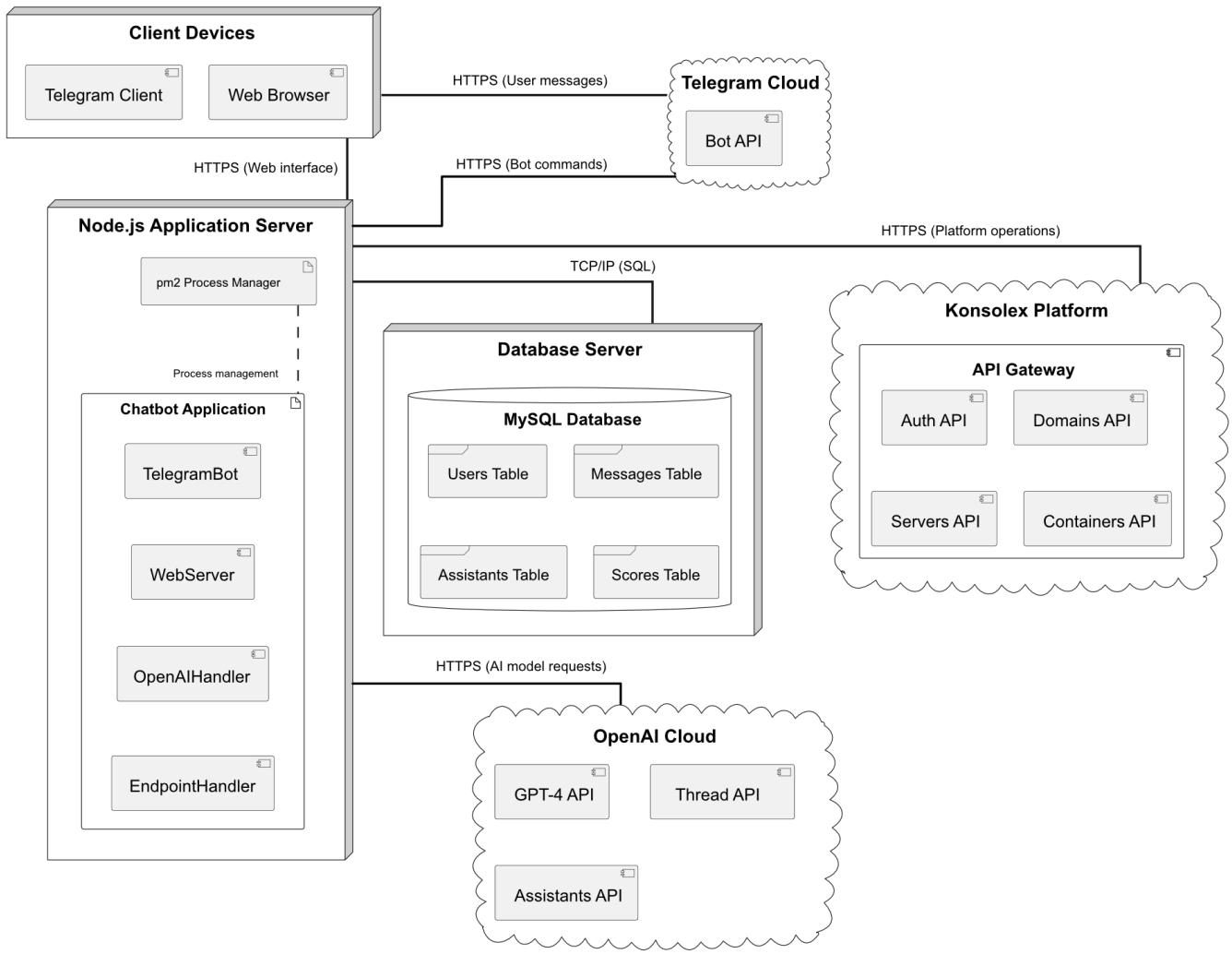
4. L'**EndpointHandler** funge da adattatore per le API Konsolex, consentendo l'esecuzione di operazioni tecniche come riavvii di server o gestione domini.

5. I **Repository** forniscono un'astrazione uniforme per l'accesso al database, supportando la persistenza e il recupero dei dati per tutti i componenti applicativi.

Questa organizzazione supporta l'inversione del controllo (principio di design che inverte il flusso del controllo tradizionale) e facilita il testing dei singoli moduli, mentre la chiara separazione dei livelli garantisce che modifiche in una parte del sistema abbiano un impatto minimo sulle altre componenti.

5.3.3 Deployment Diagram

Il diagramma di deployment (rappresentazione grafica che mostra la configurazione fisica dei componenti hardware e software) illustra la distribuzione fisica del sistema su diversi nodi hardware e servizi cloud, evidenziando come i componenti architetturali siano allocati nell'infrastruttura operativa.



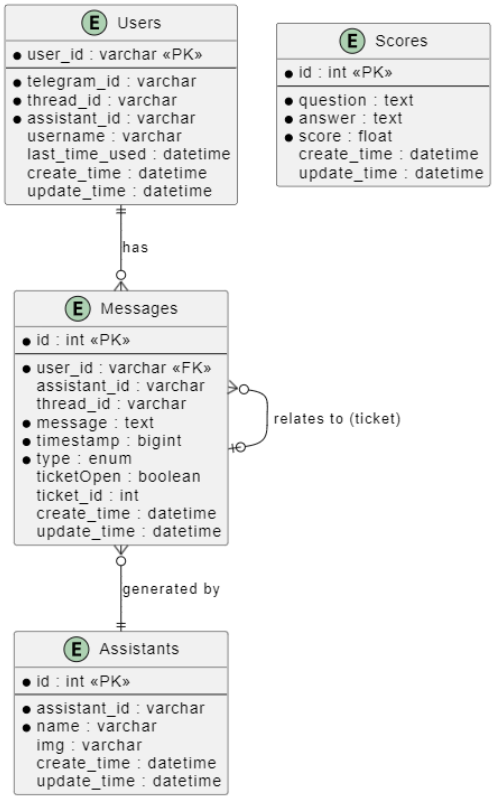
L'architettura di deployment è organizzata attorno a cinque nodi principali:

- **Node.js Application Server** (server applicativo basato su Node.js): Ospita l'applicazione principale, comprendendo i componenti dell'Application Layer come OpenAIHandler, EndpointHandler, WebServer, TelegramCommandHandler e il sistema di ticketing. Questo server gestisce tutta la logica applicativa, le interazioni con le API esterne e il coordinamento dei flussi di comunicazione.
- **MySQL Database Server**: Fornisce la persistenza dei dati attraverso un database relazionale che memorizza informazioni su utenti, messaggi, thread conversazionali e valutazioni delle risposte. I repository implementati nel sistema accedono a questo database tramite interfacce ben definite.
- **Konsolex Platform** (piattaforma proprietaria della società OnTheCloud): Funge da intermediario tra l'applicazione e i servizi esterni di Konsolex, gestendo autenticazione, routing e trasformazione delle richieste. Questo componente è cruciale per l'integrazione con l'ecosistema esistente di OnTheCloud.
- **Telegram API Cloud** (servizi cloud di Telegram): Gestisce tutte le comunicazioni con la piattaforma Telegram, permettendo l'interazione con gli utenti tramite l'interfaccia di messaggistica. Il sistema vi si connette tramite il TelegramBot e il TelegramCommandHandler.
- **OpenAI Cloud Services** (servizi cloud di OpenAI): Fornisce le capacità di intelligenza artificiale attraverso l'API OpenAI, elaborando i messaggi degli utenti, generando risposte contestuali e gestendo thread conversazionali complessi.

La comunicazione tra questi nodi avviene principalmente tramite protocolli HTTP/HTTPS (protocolli di trasferimento dati utilizzati per la comunicazione su internet), con connessioni sicure e autenticate per garantire l'integrità e la riservatezza dei dati scambiati.

5.3.4 Diagramma ER del Database

Il diagramma Entity-Relationship modella la struttura del database che supporta il sistema.



Le principali entità del database includono:

- **Users**: Utenti registrati con riferimenti ai loro thread OpenAI
- **Messages**: Messaggi scambiati nella piattaforma
- **Assistants**: Configurazioni degli assistenti virtuali
- **Scores**: Valutazioni delle risposte AI

Le relazioni tra queste entità riflettono i vincoli di integrità referenziale necessari per mantenere la coerenza dei dati.

5.4 Flussi di Dati e Comandi

5.4.1 Flusso di Conversazione Utente

Il flusso di dati tipico per una conversazione utente segue questo percorso:

1. L'utente invia un messaggio tramite la piattaforma Konsolex
2. Il messaggio viene salvato nel database tramite `user-repository.ts`
3. OpenAIHandler processa il messaggio nel contesto del thread conversazionale esistente
4. Se necessario, vengono eseguite operazioni tramite function calling verso l'Endpoint Handler
5. La risposta generata viene persistita e inviata all'utente

5.4.2 Flusso di Escalation

Quando una richiesta richiede intervento umano:

1. L'AI riconosce la necessità di escalation (trasferimento della richiesta ad un amministratore) o l'utente la richiede esplicitamente
2. Un ticket viene creato tramite `sendMessageToAdmin` in `endpoint.ts`
3. Il ticket viene notificato agli admin tramite Telegram e interfaccia web
4. L'operatore risponde tramite l'interfaccia amministrativa
5. La risposta viene inviata all'utente tramite il canale originale

5.4.3 Flusso di Operazioni Tecniche

Quando è richiesta un'operazione tecnica:

1. L'AI identifica l'operazione necessaria tramite function calling
2. La funzione corrispondente viene invocata nell'Endpoint Handler
3. L'Endpoint Handler interagisce con l'API Konsolex
4. Il risultato viene restituito all'AI per formulare una risposta appropriata
5. L'utente riceve conferma dell'operazione eseguita

5.5 Conclusioni Architetturali

L'architettura del sistema chatbot AI rappresenta un bilanciamento tra principi di design moderno del software e requisiti pratici di un'applicazione di supporto tecnico.

La separazione dei componenti in moduli con responsabilità specifiche facilita non solo la manutenzione corrente ma anche l'evoluzione futura del sistema, permettendo l'adozione di nuove tecnologie o l'estensione delle funzionalità esistenti con minimo impatto sull'architettura complessiva.

L'approccio architetturale adottato riflette la filosofia di "anticipare il cambiamento" tipica dell'ingegneria del software moderna, creando un sistema che non solo risponde ai requisiti attuali ma è pronto ad evolvere con le esigenze future dell'organizzazione.