Capitolo 6: Componenti Principali

Questo capitolo fornisce un'analisi dettagliata dei componenti principali che costituiscono il sistema chatbot AI (sistema automatizzato di conversazione basato su Intelligenza Artificiale), descrivendo il loro ruolo specifico, la loro implementazione e le interazioni che stabiliscono con gli altri componenti. Ogni modulo è stato progettato per cercare di mantenere separazione delle responsabilità, manutenibilità e estensibilità.

6.1 Web Server & REST API

Il componente Web Server, implementato nel file `web-server.ts`, costituisce il cuore dell'interfaccia programmatica del sistema, esponendo API REST (architettura per la comunicazione tra sistemi web) che consentono la comunicazione tra il frontend (parte visibile all'utente) e il backend (parte che elabora i dati). Questo modulo utilizza Express.js come framework web e implementa un'architettura di endpoint organizzati in modo funzionale per gestire diversi tipi di richieste.

6.1.1 Architettura Generale

Il file `web-server.ts` configura un'applicazione Express con middleware essenziali come CORS (Cross-Origin Resource Sharing, meccanismo che consente a risorse web su un dominio di essere richieste da un altro dominio), parser JSON (per elaborare payload in formato JSON) e un middleware di logging per tracciare le richieste in ingresso. Il server espone un insieme di endpoint organizzati in tre categorie principali, definite nelle costanti `API` e `ADMIN_API` nel file `constants.ts`.

6.1.2 Endpoint per Utenti Finali

Il web server fornisce diversi endpoint destinati agli utenti finali, progettati per gestire l'interazione con il chatbot:

- **`API.SEND_MESSAGE`**: Endpoint cruciale che gestisce l'invio di messaggi degli utenti e la generazione di risposte. Questa funzione implementa una logica articolata che:
    - Valida i parametri della richiesta (userId, tg_id, testo)
    - Verifica e gestisce l'esistenza di ticket aperti
    - Salva il messaggio dell'utente nel database
    - Genera risposte AI o inoltra messaggi agli amministratori in base al contesto
    - Notifica la piattaforma proprietaria Konsolex degli aggiornamenti
- **`API.MESSAGES`**: Recupera la cronologia dei messaggi per un determinato utente, permettendo la visualizzazione dello storico delle conversazioni.
- **`API.GET_MESSAGES_USER_GPT_BY_USER_ID`**: Fornisce i messaggi filtrati (solo utente e GPT) per un utente specifico, escludendo i messaggi di ticket e amministrativi.
- **`API.GET_TICKETS_MESSAGES_USER`**: Recupera i messaggi relativi ai ticket di supporto per un utente specifico.
- **`API.CHECK_USER_OPEN_TICKETS`**: Verifica rapidamente se un utente ha ticket di supporto aperti, utile per la gestione dello stato dell'interfaccia.

6.1.3 Endpoint Amministrativi

Endpoint specifici per gli amministratori, che consentono la gestione del sistema di supporto:

- `ADMIN_API.ADMIN_REPLY_TICKET`: Permette agli amministratori di rispondere ai ticket degli utenti. Questa funzione:
  - Verifica l'esistenza dell'utente e del ticket associato
  - Salva la risposta dell'amministratore nel database
  - Inoltra la risposta all'utente via Telegram
  - Aggiorna gli stati di risposta e notifica
  - Sincronizza lo stato con l'interfaccia della piattaforma proprietaria Konsolex
- `ADMIN_API.GET_ALL_USERS` e `ADMIN_API.ALL_USERS_DATA`: Forniscono informazioni sugli utenti registrati nel sistema, con diversi livelli di dettaglio. La seconda funzione include dati completi come messaggi e stato dei ticket.
- `ADMIN_API.GET_MESSAGES_USER_GPT`: Recupera i messaggi non relativi a ticket per tutti gli utenti, ordinati per data di ultima interazione.
- `ADMIN_API.ADMIN_TICKET_LIST`: Fornisce la lista di tutti i ticket aperti con i relativi messaggi, essenziale per la dashboard amministrativa.
- `ADMIN_API.CLOSE_TICKET`: Permette agli amministratori di chiudere un ticket di supporto quando la problematica è stata risolta.

6.1.4 Integrazione con Konsolex

Il web server implementa meccanismi di integrazione con la piattaforma proprietaria Konsolex:

- **Sincronizzazione stato messaggi**: Attraverso chiamate a `KONSOLEX_ENDPOINT.MESSAGES_UPDATE`, il sistema notifica la piattaforma Konsolex quando ci sono nuovi messaggi o aggiornamenti, garantendo coerenza tra le interfacce.
- **Verifica utenti**: L'endpoint `API.GET_TELEGRAM_ID` supporta la mappatura tra ID Telegram e account Konsolex.
- **Autenticazione tramite userId**: Tutti gli endpoint richiedono un `userId` valido come forma di autenticazione, che viene utilizzato anche per le richieste verso l'API Konsolex.

6.1.5 Gestione Errori e Resilienza

Il web server implementa una gestione degli errori strutturata, con pattern comuni in tutti gli endpoint:

- Validazione dei parametri obbligatori con feedback espliciti all'utente
- Gestione delle eccezioni con try/catch e logging dettagliato
- Risposte HTTP appropriate (400 per richieste errate, 404 per risorse non trovate, 500 per errori interni)
- Recupero resiliente in caso di errori durante le operazioni di database o chiamate API esterne

Questa architettura API ben organizzata fornisce un'interfaccia coerente e flessibile per tutte le componenti del sistema, facilitando l'interazione sia per gli utenti finali che per gli amministratori, e garantendo una solida integrazione con la piattaforma Konsolex esistente.

6.2 OpenAI Handler

L'OpenAI Handler è il componente centrale responsabile:

- dell'interazione con l'API OpenAI, che permette l'accesso ai modelli di intelligenza artificiale come GPT-4
- della gestione dei thread conversazionali (conversazioni organizzate in sequenze che mantengono il contesto)
- dell'interpretazione delle risposte AI, fungendo da ponte tra l'interfaccia utente e le capacità di intelligenza artificiale.

6.2.1 Gestione Thread e Assistenti

Uno degli aspetti più innovativi dell'implementazione è l'utilizzo dell'API Assistants di OpenAI, che consente di mantenere thread conversazionali persistenti per ogni utente. Il file `openai-handlers.ts` implementa la funzione `createReply` che gestisce l'aggiunta di messaggi utente ai thread, l'esecuzione di "run" dell'assistente con parametri ottimizzati, e l'elaborazione delle risposte generate.

6.2.2 Function Calling

Una delle funzionalità più potenti implementate è la capacità dell'AI di richiamare funzioni specifiche per eseguire operazioni concrete. La funzione `handleToolCalls` nel file `openai-handlers.ts` gestisce questo meccanismo (detto "function calling"), elaborando le richieste di esecuzione di funzioni dall'AI, invocando le funzioni appropriate tramite `executeTool` e restituendo i risultati all'AI per l'integrazione nella risposta.

6.2.3 Ottimizzazione delle Risposte

Per garantire la massima qualità e coerenza delle risposte, l'OpenAI Handler implementa varie strategie di ottimizzazione:

- **Parametrizzazione del modello**: La funzione `createReply` utilizza valori ottimizzati per parametri come `temperature` (parametro che controlla la casualità delle risposte, impostato a 0.1) e `top_p` (parametro che influenza la distribuzione di probabilità delle parole generate, impostato a 1) per favorire risposte deterministiche e tecnicamente accurate.
- **Gestione dei timeout**: La funzione `waitForResponse` implementa meccanismi per gestire risposte che richiedono troppo tempo.
- **Formatting delle risposte**: La funzione `formatGPTResponse` in `aux-functions.ts` applica una formattazione delle risposte per migliorare la leggibilità e l'usabilità.

6.2.4 Resilienza e Gestione Errori

Il sistema è progettato per essere resiliente a errori e interruzioni dell'API, con meccanismi di gestione degli errori nella funzione `createReply` che includono:

- **Retry automatici**: In caso di errori temporanei o di rete
- **Creazione di nuovi thread**: La gestione del caso "TIMEOUT" include la creazione di un nuovo thread quando quello esistente diventa inutilizzabile
- **Fallback a risposte predefinite**: Messaggi di errore user-friendly quando l'AI non è in grado di generare una risposta appropriata

6.3 Telegram Bot

Il Telegram Bot rappresenta una scelta alternativa all'utilizzo della piattaforma proprietaria Konsolex, consentendo agli utenti di porre domande al chatbot anche quando si è da remoto. Attualmente i clienti lo utilizzano principalmente per rispondere ai messaggi degli amministratori e non tanto per fare domande alla AI.

6.3.1 Struttura e Inizializzazione

Il bot era già precedentemente implementato utilizzando la libreria Telegraf (framework che semplifica lo sviluppo di bot per Telegram), che fornisce un'API type-safe per interagire con la piattaforma Telegram. La funzione `setupBot` nel file `bot.ts` è responsabile della configurazione del bot, registrando i vari handler per comandi, messaggi di testo e callback da pulsanti interattivi. La mia aggiunta consisteva nel cambiare la logica del flusso dei messaggi che attualmente non vanno più da user ad admin, ma passano attraverso OpenAI, e solo se un ticket è aperto si ha la possibilità di parlare con un amministratore.

6.3.2 Gestione delle Conversazioni

Il bot gestisce il flusso conversazionale attraverso una serie di handler specializzati:

1. **Messaggi di testo**: L'handler `bot.on("text")` elabora i messaggi testuali degli utenti, utilizzando la funzione `storeMessageAndCreateReply` di `aux-functions.ts` per salvare i messaggi e generare risposte.

2. **Comandi**: Handler come `bot.start()` gestiscono comandi specifici come `/start` per iniziare una nuova conversazione o `/help` per ottenere assistenza.

3. **Interazioni con pulsanti**: L'handler `bot.on("callback_query")` gestisce le interazioni con pulsanti inline utilizzati per funzionalità specifiche.

6.4 Endpoint Handler per Konsolex

L'Endpoint Handler costituisce il punto di integrazione centrale con la piattaforma proprietaria Konsolex, consentendo al chatbot di eseguire operazioni concrete sulla piattaforma in risposta alle richieste degli utenti.

6.4.1 Architettura dell'Integrazione

Implementato nel file `endpoint.ts`, questo componente fornisce un layer di astrazione per le interazioni con le diverse API della piattaforma Konsolex definite nelle costanti `KONSOLEX_ENDPOINT` nel file `constants.ts`.

6.4.2 Operazioni Server

L'Endpoint Handler implementa funzioni come `restartServer` e `getServerList` nel file `endpoint.ts` per eseguire operazioni sui server gestiti dalla piattaforma proprietaria Konsolex.

6.4.3 Gestione Domini e DNS

L'handler fornisce funzioni come `getDomainList` e `checkDomainAvailability` per la gestione di domini e configurazioni DNS, interfacciandosi con gli endpoint Konsolex corrispondenti e gestendo le risposte in formato appropriato.

6.4.4 Container e Database

Per operazioni su container (tecnologia di virtualizzazione che impacchetta applicazionie) e database, l'handler offre funzionalità come `restartMysqlOrPostfix` e `restartContainer`, che permettono il riavvio di servizi specifici o container interi attraverso le relative API Konsolex.

6.4.5 Autenticazione e Verifica Utenti

L'Endpoint Handler include funzioni come `checkUserIdExists` e `setTelegramId` per la verifica degli utenti e l'integrazione con il sistema di autenticazione della piattaforma Konsolex, utilizzando gli endpoint di autenticazione della piattaforma.

6.5 Sistema di Ticketing

Il sistema di ticketing rappresenta un componente fondamentale per la gestione delle richieste di supporto che richiedono intervento umano, implementando un flusso completo dalla creazione alla risoluzione dei ticket.

6.5.1 Architettura del Sistema

Implementato principalmente nei file `ticketsCashe.ts`, il sistema di ticketing adotta un approccio basato su cache in memoria dei ticket aperti e con persistenza su database dei messaggi appartenenti alla categoria ticket, utilizzando strutture dati come `Ticket` e `TicketStatus` per rappresentare i ticket e il loro stato.

6.5.2 Creazione e Gestione Ticket

La creazione di ticket avviene principalmente attraverso la funzione `sendMessageToAdmin` in `endpoint.ts`, che prepara un ticket strutturato con tutte le informazioni necessarie, aggiorna lo stato del messaggio originale con `updateTicketStatusByMsgId` e crea o aggiorna il ticket con `getOrCreateTicket`.

6.5.3 Interfaccia per Operatori

Il sistema implementa un'interfaccia specializzata per gli operatori, tramite API REST nel file `web-server.ts` che include endpoint come `ADMIN_API.ADMIN_REPLY_TICKET` per rispondere ai ticket e `ADMIN_API.CLOSE_TICKET` per chiuderli, oltre a interfacce Telegram con bottoni interattivi per operazioni rapide.

6.5.4 Notifiche e Aggiornamenti

Il sistema implementa meccanismi di notifica tramite la funzione `sendAdminReplyToUser` in `aux-functions.ts` per mantenere utenti e operatori aggiornati sullo stato dei ticket.

6.6 Repository Pattern per Persistenza Dati

Il sistema implementa il pattern Repository (modello di progettazione software che isola la logica di accesso ai dati) per fornire un'astrazione dell'accesso ai dati, disaccoppiando la logica di business dalla persistenza e facilitando manutenzione ed evoluzione del database.

6.6.1 Struttura dei Repository

I repository sono implementati nella directory `repositories` con file specializzati per ogni entità principale:

- `user-repository.ts`: Gestione entità User
- `message-repository.ts`: Gestione entità Message
- `assistant-repository.ts`: Gestione entità Assistant
- `score-repository.ts`: Gestione entità Score

Ogni repository fornisce un'interfaccia coerente per le operazioni CRUD (Create, Read, Update, Delete - operazioni fondamentali sui dati). Per esempio, il file `message-repository.ts` implementa funzioni come `create`, `findById`, e `findByUserId` per la gestione delle entità Message, mentre `user-repository.ts` implementa funzioni analoghe per le entità User.