Documentazione Tecnica: WhatsApp Bot con Integrazione RAG e AI

1. Panoramica del Sistema

1.1 Introduzione

Questa applicazione implementa un bot WhatsApp che utilizza l'integrazione con AI (Google Gemini) e un sistema RAG (Retrieval-Augmented Generation) per fornire risposte contestuali agli utenti. Il sistema è progettato per gestire conversazioni via WhatsApp, elaborare documenti caricati dagli utenti e utilizzare queste informazioni per arricchire le risposte generate dall'intelligenza artificiale.

1.2 Architettura del Sistema

L'applicazione è strutturata secondo un'architettura modulare con i seguenti componenti principali:

- **Server Express**: Gestisce le richieste HTTP e i webhook di Twilio
- **Twilio API**: Integrazione per l'invio e la ricezione di messaggi WhatsApp
- **Google Gemini AI**: Servizio di intelligenza artificiale per la generazione di risposte
- **Sistema RAG**: Framework per il recupero di informazioni contestuali da documenti indicizzati
- **Qdrant**: Database vettoriale per la memorizzazione e il recupero di embedding dei documenti
- **Socket.IO**: Per la comunicazione in tempo reale con il frontend

1.3 Flusso di Dati Principale

1. L'utente invia un messaggio via WhatsApp al numero configurato di Twilio
2. Il webhook di Twilio inoltra il messaggio al server
3. Il server elabora il messaggio e recupera eventuali documenti rilevanti tramite RAG
4. Il messaggio dell'utente e i documenti pertinenti vengono inviati a Google Gemini
5. La risposta generata viene inviata all'utente via WhatsApp
6. La conversazione viene registrata nel sistema

2. Componenti del Sistema

2.1 Server (server.mjs)

Il modulo `server.mjs` è il componente centrale dell'applicazione che coordina tutti gli altri moduli.

2.1.1 Funzionalità Principali

- Gestione dei webhook di Twilio per la ricezione di messaggi WhatsApp
- Routing delle richieste API per il frontend

- Caricamento e salvataggio delle conversazioni
- Gestione dell'upload di file PDF
- Monitoraggio dell'inattività dell'utente

2.1.2 API Esposte

- `POST /webhook`: Riceve messaggi da Twilio e avvia l'elaborazione della risposta
- `POST /api/upload-pdf`: Endpoint per caricare file PDF da indicizzare
- `GET /api/conversations`: Recupera l'elenco delle conversazioni
- `GET /api/conversations/:phone`: Recupera i messaggi di una specifica conversazione
- `POST /api/send`: Invia un messaggio manuale a un utente
- `GET /api/documents/:phone`: Recupera i documenti indicizzati per un utente
- `GET /api/settings`: Recupera le impostazioni dell'applicazione
- `POST /api/settings`: Aggiorna le impostazioni dell'applicazione

2.1.3 Gestione Conversazioni

Il sistema mantiene le conversazioni in memoria e le salva su disco nel seguente formato:

```javascript
{
  "phone": "whatsapp:+1234567890",
  "messages": [
    {
      "direction": "received",
      "content": "Messaggio dell'utente",
      "timestamp": "2023-05-16T12:30:45.000Z"
    },
    {
      "direction": "sent",
      "content": "Risposta del sistema",
      "timestamp": "2023-05-16T12:31:00.000Z"
    }
  ],
  "lastActivity": "2023-05-16T12:31:00.000Z",
  "inactivityMessageSent": false,
  "closed": false
}
```

2.1.4 Gestione dell'Inattività

Il sistema monitora l'inattività dell'utente e:

- Invia un messaggio automatico dopo 15 minuti di inattività
- Chiude la conversazione dopo ulteriori 15 minuti senza risposta

2.2 Integrazione Gemini AI (gemini.mjs)

Il modulo `gemini.mjs` gestisce l'interazione con Google Gemini AI per la generazione di risposte.

2.2.1 Funzionalità

- Inizializzazione del client Gemini con parametri di configurazione appropriati
- Gestione dell'integrazione con il sistema RAG
- Generazione di risposte AI con controlli di sicurezza e timeout

2.2.2 Configurazione del Modello

Il sistema utilizza il modello `gemini-2.0-flash` con le seguenti configurazioni:

- Temperatura: 0.7
- Top-P: 0.9
- Top-K: 40
- Limite di token di output: 800

2.2.3 Integrazione RAG

Quando un utente invia un messaggio, il sistema:

1. Genera embedding per la query dell'utente
2. Cerca documenti pertinenti nel database vettoriale
3. Estrae contenuti rilevanti e li aggiunge al prompt per Gemini
4. Genera una risposta contestualizzata

3. Sistema RAG (Retrieval-Augmented Generation)

3.1 Componenti RAG

- `pharsingfile.mjs`: Gestisce l'estrazione di testo e la generazione di embedding
- `qdrant.mjs`: Interfaccia con il database vettoriale Qdrant

3.2 Processo di Upload Documenti

1. L'utente carica un file PDF tramite l'API
2. Il file viene salvato temporaneamente e processato
3. Il testo viene estratto e suddiviso in chunk
4. Per ogni chunk viene generato un embedding

5. I chunk e i relativi embedding vengono memorizzati in Qdrant

3.3 Recupero Documenti

1. Il sistema genera embedding per la query dell'utente
2. Esegue una ricerca di similarità vettoriale in Qdrant
3. Recupera i documenti più rilevanti per l'utente specifico
4. Integra questi documenti nel prompt per Gemini

4. Integrazione Twilio

4.1 Configurazione Sandbox WhatsApp

L'applicazione utilizza il numero di Sandbox WhatsApp di Twilio (+14155238886) per le comunicazioni. Gli utenti devono:

1. Inviare un messaggio di attivazione a questo numero
2. Seguire le istruzioni per attivare la connessione WhatsApp

4.2 Gestione Messaggi

- `sendTwilioMessage()`: Funzione che gestisce l'invio di messaggi tramite Twilio
- Gestisce il troncamento di messaggi lunghi (limite di 1500 caratteri)
- Implementa una robusta gestione degli errori per vari scenari di fallimento

5. Dati e Persistenza

5.1 Struttura Dati

- `/data/conversations.json`: Memorizza le conversazioni
- `/data/settings.json`: Contiene le impostazioni dell'applicazione
- `/temp/`: Directory per i file temporanei durante l'upload

5.2 Impostazioni Applicazione

```javascript
{
  "responseMode": "auto", // Modalità di risposta: "auto" o "manual"
  "defaultResponse": "Grazie per il tuo messaggio! Un operatore ti risponderà a breve."
}
```

6. Requisiti e Dipendenze

6.1 Requisiti di Sistema

- Node.js (versione consigliata: 14.x o successiva)
- Connessione internet per l'accesso alle API esterne

6.2 Dipendenze Principali

- Express: Framework web per il server HTTP
- Twilio: SDK per l'integrazione con WhatsApp
- @google/generative-ai: SDK per l'integrazione con Google Gemini
- Socket.io: Per la comunicazione in tempo reale
- Multer: Per la gestione dell'upload dei file
- Dotenv: Per la gestione delle variabili d'ambiente

6.3 Variabili d'Ambiente

Il sistema richiede la configurazione delle seguenti variabili d'ambiente nel file `.env`:

```
# Twilio
TWILIO_ACCOUNT_SID=your_twilio_sid
TWILIO_AUTH_TOKEN=your_twilio_token
TWILIO_PHONE_NUMBER=whatsapp:+14155238886

# Google Gemini
GEMINI_API_KEY=your_gemini_api_key

# Qdrant
QDRANT_URL=your_qdrant_url
QDRANT_API_KEY=your_qdrant_api_key

# Server
PORT=3000
```

7. Installazione e Configurazione

7.1 Installazione

1. Clonare il repository
2. Installare le dipendenze con `npm install`
3. Configurare le variabili d'ambiente nel file `.env`
4. Avviare il server con `npm start`

7.2 Verifica della Configurazione

All'avvio, il sistema esegue una serie di controlli sulle dipendenze e configurazioni:

- Verifica la presenza di tutte le variabili d'ambiente necessarie
- Testa la connessione con Twilio
- Controlla la configurazione di WhatsApp
- Verifica l'accesso all'API di Google Gemini
- Controlla l'accesso a Qdrant

8. Risoluzione dei Problemi

8.1 Problemi Comuni

Problema	Possibile Causa	Soluzione
Errore 63007	Configurazione WhatsApp mancante	Verificare che il numero sia registrato nel Sandbox WhatsApp di Twilio
Errore 21617	Messaggio troppo lungo	I messaggi sono automaticamente troncati a 1500 caratteri
Errore 21211	Numero di telefono invalido	Verificare il formato del numero di telefono
Errore 20003	Autenticazione fallita	Verificare TWILIO_ACCOUNT_SID e TWILIO_AUTH_TOKEN
RAG fallisce	Problemi con Qdrant	Verificare la connessione a Qdrant e le credenziali

8.2 Log e Monitoraggio

Il sistema implementa un logging estensivo per facilitare il debugging:

- Log dettagliati all'avvio per verificare la configurazione
- Log per ogni messaggio ricevuto e inviato
- Log specifici per le interazioni con RAG e Gemini
- Gestione degli errori con stack trace completi

9. Considerazioni sulla Sicurezza

9.1 Protezioni Implementate

- Limitazione delle dimensioni dei file (10MB per upload)
- Filtri per tipi di file (solo PDF)
- Impostazioni di sicurezza di Gemini per bloccare contenuti potenzialmente dannosi
- Timeout per le richieste API esterne

9.2 Miglioramenti Consigliati

- Implementare autenticazione per le API del frontend

- Cifrare i dati sensibili nelle conversazioni salvate
- Implementare rate limiting per prevenire abusi

10. Estensibilità e Sviluppi Futuri

10.1 Potenziali Miglioramenti

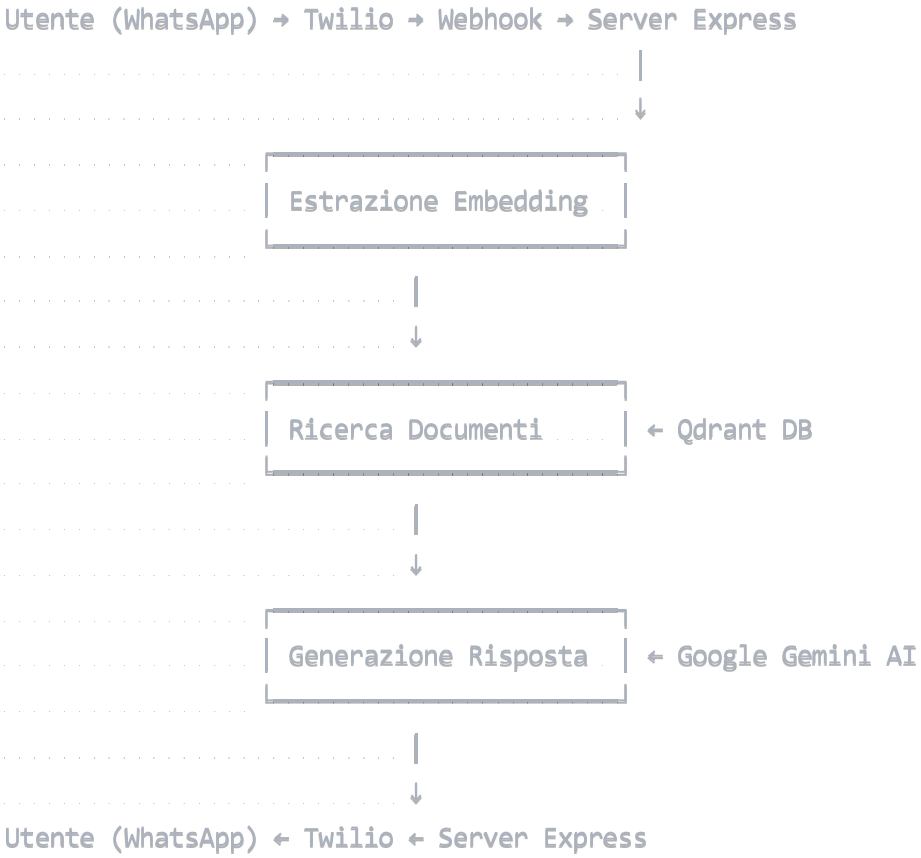
- Supporto per ulteriori formati di file oltre PDF
- Integrazione con più modelli AI oltre a Gemini
- Implementazione di un sistema di analisi dei sentimenti
- Supporto multi-lingua migliorato
- Integrazione con sistemi CRM

10.2 Architettura per l'Estensione

Il sistema è progettato con una struttura modulare che consente:

- Sostituzione facile dei componenti (es. cambio del provider AI)
- Aggiunta di nuovi endpoint API
- Integrazione con servizi esterni aggiuntivi

Appendice A: Diagramma del Flusso di Dati



Appendice B: Riferimenti API

Questa sezione contiene i dettagli tecnici delle API esposte dal sistema.

B.1 API Webhook

Endpoint: `POST /webhook` **Descrizione**: Riceve notifiche da Twilio quando arriva un nuovo messaggio WhatsApp **Parametri**:

- `Body`: Testo del messaggio
- `From`: Numero di telefono del mittente (formato: `whatsapp:+1234567890`)
- `To`: Numero di telefono del destinatario (formato: `whatsapp:+14155238886`)

B.2 API Upload PDF

Endpoint: `POST /api/upload-pdf` **Descrizione**: Carica e elabora un file PDF per l'indicizzazione nel sistema RAG **Parametri**:

- `pdfFile`: File PDF (multipart/form-data)
- `organizationId`: ID dell'organizzazione/utente a cui associare il documento **Risposta**:

```
json
{
  "success": true,
  "message": "File elaborato con successo",
  "recordsCount": 15,
  "fileName": "documento.pdf"
}
```

B.3 API Conversazioni

Endpoint: `GET /api/conversations` **Descrizione**: Recupera l'elenco di tutte le conversazioni **Risposta**: Array di oggetti conversazione

Endpoint: `GET /api/conversations/:phone` **Descrizione**: Recupera i messaggi di una specifica conversazione **Parametri**:

- `phone`: Numero di telefono dell'utente (formato: `whatsapp:+1234567890`) **Risposta**: Oggetto conversazione con messaggi

B.4 API Invio Messaggi

Endpoint: `POST /api/send` **Descrizione**: Invia un messaggio manuale a un utente **Parametri**:

- `phone`: Numero di telefono del destinatario
- `message`: Testo del messaggio da inviare **Risposta**:

```json
{
  "status": "Messaggio inviato con successo"
}
```

B.5 API Documenti

Endpoint: `GET /api/documents/:phone` **Descrizione**: Recupera i documenti indicizzati per un utente **Parametri**:

- `phone`: Numero di telefono dell'utente **Risposta**:

```json
{
  "success": true,
  "documents": [
    {
      "source": "documento1.pdf",
      "chunks": 10,
      "lastUpdated": "2023-05-16T12:30:45.000Z"
    }
  ],
  "message": "Recuperati 10 chunks da 1 documenti"
}
```

B.6 API Impostazioni

Endpoint: `GET /api/settings` **Descrizione**: Recupera le impostazioni dell'applicazione **Risposta**: Oggetto impostazioni

Endpoint: `POST /api/settings` **Descrizione**: Aggiorna le impostazioni dell'applicazione **Parametri**:

- `responseMode`: Modalità di risposta ("auto" o "manual")
- `defaultResponse`: Testo della risposta predefinita **Risposta**: Oggetto impostazioni aggiornato