Technical Design Document

HeySalonDesk HITL (Human-in-the-Loop) System

---

1. Problem Statement

The Challenge

Today, if our AI doesn't know something, it either **hallucinates** or **fails**. That's not good enough.

Traditional AI voice agents face a critical limitation: when they encounter questions outside their knowledge base, they either make up incorrect information (hallucination) or simply fail to provide any response. This creates a poor customer experience and damages trust.

The Solution

We want the AI to behave like a **real agent** — escalate intelligently, learn, and get smarter over time.

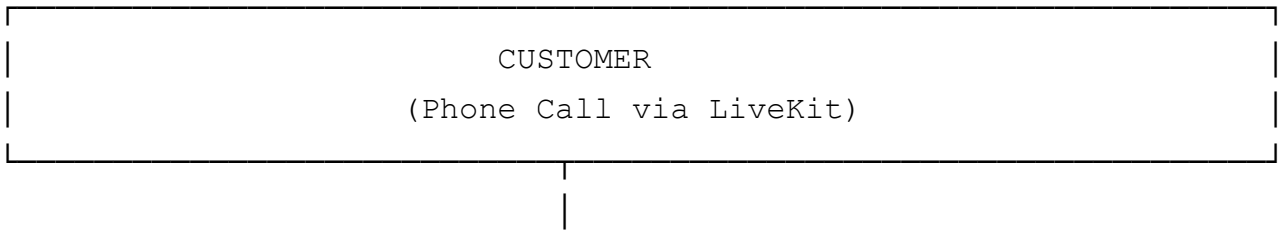
This system implements a **Human-in-the-Loop (HITL)** architecture where:

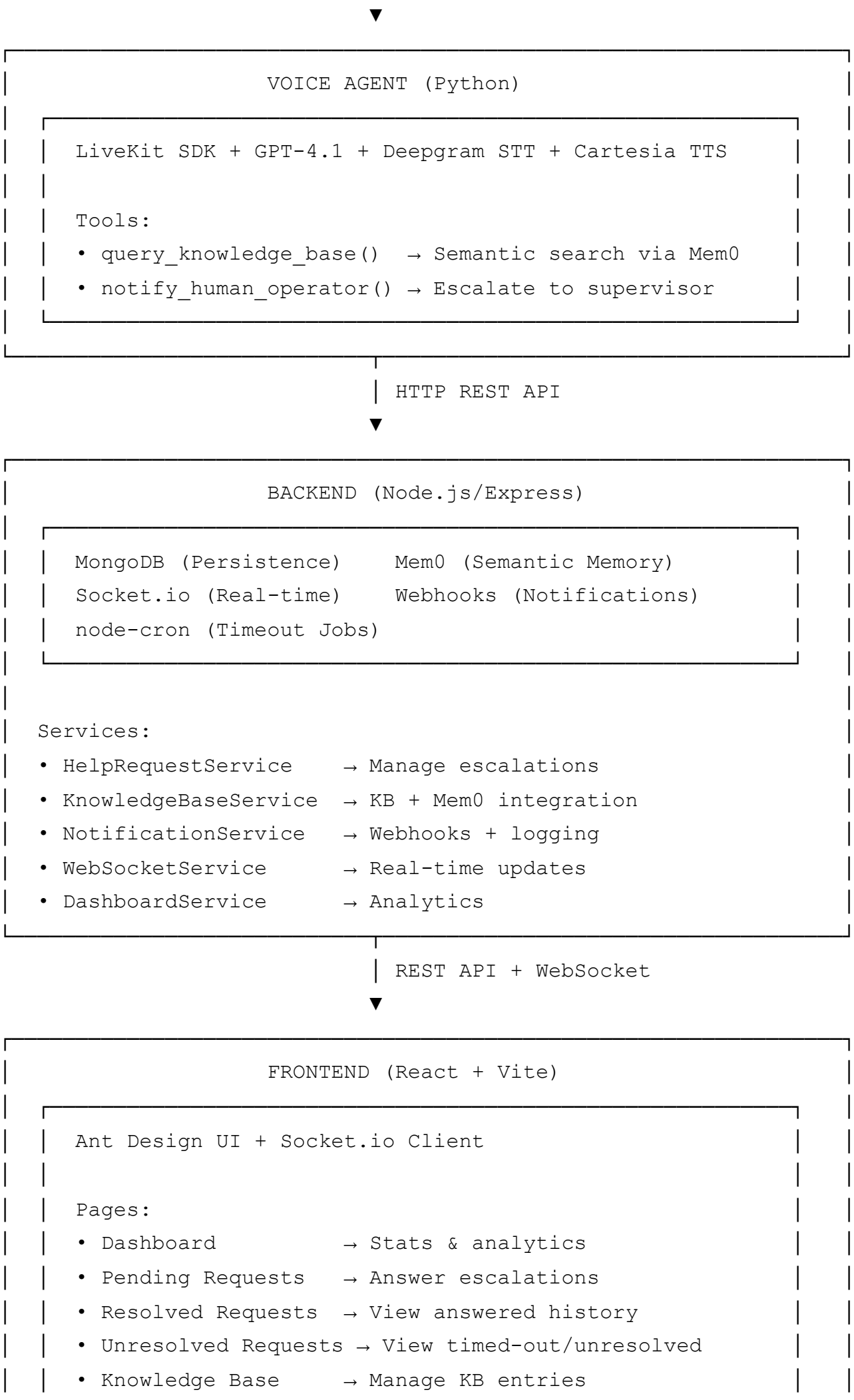
- The AI agent handles known questions autonomously
- Unknown questions are escalated to human supervisors
- Supervisor responses are automatically added to the knowledge base
- The AI learns from each escalation, becoming progressively smarter
- Customers receive accurate answers without hallucination

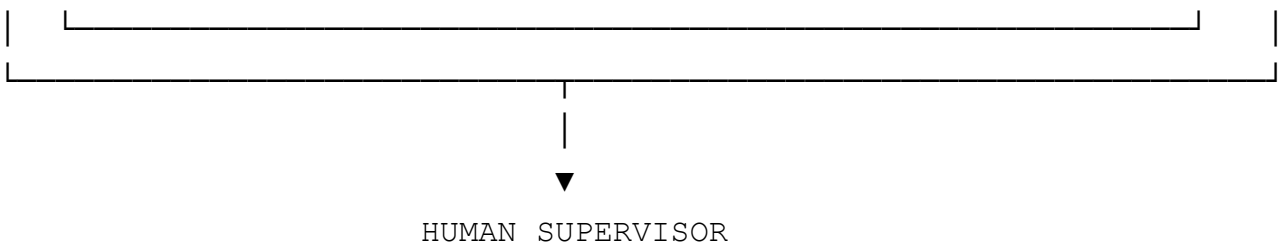
---

2. High-Level Design

System Architecture




VOICE AGENT (Python)
LiveKit SDK + GPT-4.1 + Deepgram STT + Cartesia TTS
Tools:
• query_knowledge_base() → Semantic search via Mem0
• notify_human_operator() → Escalate to supervisor
HTTP REST API
BACKEND (Node.js/Express)
MongoDB (Persistence)
Mem0 (Semantic Memory)
Socket.io (Real-time)
Webhooks (Notifications)
node-cron (Timeout Jobs)
Services:
• HelpRequestService → Manage escalations
• KnowledgeBaseService → KB + Mem0 integration
• NotificationService → Webhooks + logging
• WebSocketService → Real-time updates
• DashboardService → Analytics
REST API + WebSocket
FRONTEND (React + Vite)
Ant Design UI + Socket.io Client
Pages:
• Dashboard → Stats & analytics
• Pending Requests → Answer escalations
• Resolved Requests → View answered history
• Unresolved Requests → View timed-out/unresolved
• Knowledge Base → Manage KB entries



Request Lifecycle

```
1. Customer calls → Voice agent receives call
2. Agent searches knowledge base (Mem0 semantic search)
3a. Answer found → Agent responds immediately
3b. Answer NOT found → Agent escalates to supervisor
4. Supervisor receives notification (WebSocket + Webhook)
5. Supervisor answers via dashboard
6. Answer saved to MongoDB + Mem0
7. Customer notified (Webhook → SMS/Email integration)
8. Knowledge learned → Future calls auto-answered
```

Key Design Principles

1. **Semantic Search First**: Uses Mem0 AI for intelligent, context-aware knowledge retrieval
2. **Real-time Communication**: WebSocket for instant supervisor notifications
3. **Automatic Learning**: Every resolved escalation becomes permanent knowledge
4. **Timeout Management**: Auto-timeout after 30 minutes with warnings at 25 minutes
5. **Scalable Architecture**: MongoDB for persistence, stateless services, horizontal scaling ready

---

3. Service-by-Service Breakdown

3.1 BACKEND (Node.js/Express)

Technology Stack

- **Runtime**: Node.js 18+
- **Framework**: Express 5
- **Database**: MongoDB (Mongoose ODM 8)
- **Semantic Memory**: Mem0 AI 2.1.38
- **Real-time**: Socket.io 4
- **Job Scheduler**: node-cron 4

- **HTTP Client**: Axios (for webhooks)

Database Schema

HelpRequest Collection

Field	Type	Description
`_id`	ObjectId	Unique request identifier
`question`	String	Customer's question (required, trimmed)
`customerPhone`	String	Customer phone number (required)
`customerContext`	String	Additional context (optional)
`status`	Enum	`pending` | `resolved` | `unresolved`
`answer`	String	Supervisor's answer (when resolved)
`supervisorNotes`	String	Internal notes from supervisor
`timeoutAt`	Date	Auto-timeout timestamp (30 min from creation)
`resolvedAt`	Date	Resolution timestamp
`memoryId`	String	Reference to Mem0 memory ID
`createdAt`	Date	Auto-generated creation timestamp
`updatedAt`	Date	Auto-generated update timestamp

Indexes:

- `{ status: 1, createdAt: -1 }` - For filtering pending/resolved requests
- `{ timeoutAt: 1 }` - For timeout checker job

KnowledgeBase Collection

Field	Type	Description
`_id`	ObjectId	Unique KB entry identifier
`question`	String	Question text (required, trimmed)
`answer`	String	Answer text (required)
`category`	Enum	`hours` | `services` | `pricing` | `location` | `booking` | `other`
`tags`	Array[String]	Searchable tags

Field	Type	Description
`source`	Enum	`initial` (seeded)| `learned` (from escalations)
`sourceRequestId`	ObjectId	Reference to originating HelpRequest
`usageCount`	Number	Times this answer was used (default: 0)
`memoryId`	String	Reference to Mem0 memory ID
`isActive`	Boolean	Soft delete flag (default: true)
`createdAt`	Date	Auto-generated creation timestamp
`updatedAt`	Date	Auto-generated update timestamp

Indexes:

- `{ category: 1, isActive: 1 }` - For category filtering
- `{ source: 1, createdAt: -1 }` - For analytics

API Endpoints

Help Requests APIs

Method	Endpoint	Description
`GET`	`/api/help-requests`	Get all requests (supports pagination, filtering)
`GET`	`/api/help-requests/:id`	Get single request by ID
`POST`	`/api/help-requests`	Create new help request
`PATCH`	`/api/help-requests/:id/resolve`	Resolve request with answer
`PATCH`	`/api/help-requests/:id/unresolved`	Mark request as unresolved
`DELETE`	`/api/help-requests/:id`	Delete request

Knowledge Base APIs

Method	Endpoint	Description
`GET`	`/api/knowledge-base`	Get all KB entries
`GET`	`/api/knowledge-base/search?q=query`	Semantic search via Mem0
`POST`	`/api/knowledge-base`	Add new KB entry

Method	Endpoint	Description
PATCH	/api/knowledge-base/:id	Update KB entry
DELETE	/api/knowledge-base/:id	Soft delete KB entry

Agent Integration APIs

Method	Endpoint	Description
POST	/api/agent/check-knowledge	Search KB for answer (used by voice agent)
POST	/api/agent/escalate	Create escalation request
GET	/api/agent/kb-sync	Sync entire knowledge base
POST	/api/agent/track-usage	Increment usage count for KB entry

Dashboard APIs

Method	Endpoint	Description
GET	/api/dashboard/stats	Get system statistics
GET	/api/dashboard/analytics	Get analytics data

Services and Methods

HelpRequestService

Method	Description
createRequest(data)	Create new help request, set 30-min timeout, notify supervisor
resolveRequest(requestId, resolution)	Resolve request, create KB entry, notify customer
markUnresolved(requestId, reason)	Mark request as unresolved
getPendingRequests(filters)	Get all pending requests
getAllRequests(options)	Get requests with pagination and sorting
checkTimeouts()	Find and auto-resolve timed-out requests
getStats()	Get request statistics (pending, resolved, unresolved)

KnowledgeBaseService

Method	Description
`searchKnowledge(query, limit)`	Semantic search using Mem0
`addKnowledge(data)`	Add entry to MongoDB + Mem0
`updateKnowledge(id, updates)`	Update entry in MongoDB + Mem0
`deleteKnowledge(id)`	Soft delete from MongoDB, hard delete from Mem0
`trackUsage(id)`	Increment usage counter
`getAllKnowledge(filters)`	Get all active KB entries
`getStats()`	Get KB statistics (total, by category, most used)
`seedInitialKnowledge()`	Seed 5 initial salon FAQs on startup

Mem0Service

Method	Description
`addMemory(question, answer, metadata)`	Add memory to Mem0 cloud
`searchMemory(query, limit)`	Semantic search in Mem0
`getAllMemories()`	Retrieve all memories for agent
`updateMemory(memoryId, content, metadata)`	Update existing memory
`deleteMemory(memoryId)`	Delete memory from Mem0

NotificationService

Method	Description
`sendWebhook(eventType, payload)`	Send POST request to configured webhook URL
`notifySupervisor(request)`	Console log + webhook for new escalation
`notifyCustomer(phone, message)`	Console log + webhook for customer notification
`notifyAgent(kbEntry)`	Console log + webhook for KB update
`sendTimeoutWarning(request, minutesRemaining)`	WebSocket + webhook for timeout warning

WebSocketService

Method	Description

Method	Description
`initialize()`	Set up Socket.io event handlers
`handleConnection(socket)`	Handle new client connections
`handleSubscribe(socket, room)`	Subscribe client to room (default: "supervisor")
`handleUnsubscribe(socket, room)`	Unsubscribe from room

WebSocket Events Emitted:

- `connection_info` - Initial connection metadata
- `new_help_request` - New escalation created
- `request_resolved` - Request answered
- `request_unresolved` - Request marked unresolved
- `kb_updated` - Knowledge base modified
- `request_timeout_warning` - Request about to timeout

DashboardService

Method	Description
`getStats()`	Get dashboard statistics (requests, KB, response times)
`calculateAvgResolutionTime()`	Calculate average and median resolution times
`getAnalytics(startDate, endDate)`	Get analytics for date range

Background Jobs

TimeoutChecker (node-cron)

- **Schedule**: Every 5 minutes (`*/5 * * * *`)
- **Tasks**:
  1. Find requests with `status: pending` and `timeoutAt <= now`
  2. Mark as `unresolved` with reason "Auto-timeout: No response within 30 minutes"
  3. Send customer notification via webhook
  4. Find requests timing out in 5 minutes
  5. Send timeout warnings via WebSocket + webhook

3.2 AGENT (Python/LiveKit)

Technology Stack

- **Language**: Python 3.9+
- **Voice Framework**: LiveKit Agents SDK
- **LLM**: OpenAI GPT-4.1
- **Speech-to-Text**: Deepgram Nova-3 (multilingual)
- **Text-to-Speech**: Cartesia (voice ID: `a167e0f3-df7e-4d52-a9c3-f949145efdab`)
- **Voice Activity Detection**: Silero VAD
- **HTTP Client**: requests library

Agent Overview

Name: Glamour
Role: Friendly and professional voice assistant for Glamour Salon
Personality: Warm, polite, calm, confident
Communication Style: Natural conversational speech (no symbols, emojis, or formatting)

Agent Tools

1. query_knowledge_base(query: str)

Purpose: Search the salon's knowledge base using semantic search

Flow:

1. Agent calls this tool with customer's question
2. Sends POST to `/api/agent/check-knowledge`
3. Backend searches Mem0 for relevant answers
4. Returns JSON with `found`, `answer`, `confidence`, `kbEntryId`
5. Agent speaks the answer naturally if found

Example Response:

```
{
  "found": true,
  "answer": "We're open Monday through Saturday from 9 AM to 7 PM.",
  "confidence": 0.95,
  "kbEntryId": "507f1f77bcf86cd799439011"
}
```

2. notify_human_operator(question: str, customer_phone: str)

Purpose: Escalate unknown questions to human supervisor

Flow:

1. Agent calls this tool when KB search fails
2. Sends POST to `/api/agent/escalate`
3. Backend creates HelpRequest with 30-min timeout
4. Notifies supervisor via WebSocket + webhook
5. Returns escalation confirmation
6. Agent tells customer: "I have shared your question with our team. We'll get back to you shortly."

Example Response:

```
{
  "success": true,
  "requestId": "507f1f77bcf86cd799439011",
  "message": "Request escalated to supervisor",
  "estimatedResponseTime": "30 minutes"
}
```

System Prompt (Abbreviated)

```
You are Glamour, the friendly voice assistant for Glamour Salon.

Rules:
1. Greet customers warmly
2. Use query_knowledge_base for all questions
3. Never make up information
4. Use notify_human_operator if KB has no answer
5. Speak naturally (no symbols, bullets, or formatting)

Salon Info:
- Location: 123 Beauty Lane, San Francisco
- Hours: Mon-Sat 9 AM - 7 PM
- Services: Haircuts ($50), Coloring ($120), Manicures ($35), etc.
```

Entry Point

The agent uses LiveKit's `AgentSession` with:

- **STT**: Deepgram Nova-3 (multilingual support)
- **LLM**: GPT-4.1 (OpenAI)

- **TTS**: Cartesia (natural voice)
- **VAD**: Silero (voice activity detection)
- **Noise Cancellation**: BVC (background voice cancellation)

Initial greeting: "Hello, this is Glamour Salon. How can I help you today?"

3.3 FRONTEND (React + Vite)

Technology Stack

- **Framework**: React 18
- **Build Tool**: Vite 5
- **UI Library**: Ant Design 5
- **Icons**: @ant-design/icons
- **Routing**: React Router DOM 6
- **HTTP Client**: Axios
- **WebSocket**: Socket.io-client 4
- **Date Handling**: Day.js

Pages

1. Dashboard (`/`)

Purpose: Overview of system statistics and analytics

Features:

- Total requests (pending, resolved, unresolved)
- Resolution rate percentage
- Average response time
- Knowledge base growth chart
- Top questions chart
- WebSocket connection status indicator
- Real-time updates via Socket.io

Components Used:

- `Card`, `Statistic`, `Progress`, `Badge` (Ant Design)
- Custom `useWebSocket` hook for real-time data

2. Pending Requests (`/pending`)

Purpose: View and answer pending escalations

Features:

- List of all pending help requests
- Urgency indicators (time since creation)
- Filter by status and date
- Quick answer modal with:
  - Answer text area
  - Category selector (hours, services, pricing, location, booking, other)
  - Supervisor notes field
- Auto-refresh on new WebSocket events
- Resolve/Unresolved actions

Components Used:

- `RequestCard` - Individual request display
- `AnswerForm` - Modal for answering requests
- `List`, `Modal`, `Form`, `Select`, `Input` (Ant Design)

3. Knowledge Base (`/knowledge`)

Purpose: Manage knowledge base entries

Features:

- Searchable table of all KB entries
- Filter by category
- Usage count tracking
- Add new KB entry (manual)
- Edit existing entries
- Soft delete entries
- Syncs with Mem0 automatically
- Shows source (initial vs learned)

Components Used:

- `KnowledgeBaseTable` - Main table component
- `Table`, `Tag`, `Button`, `Modal`, `Form` (Ant Design)

4. Resolved Requests (`/resolved`)

Purpose: View history of resolved requests

Features:

- List of all resolved requests
- Shows question, answer, resolution time
- Filter by date range
- View linked KB entry
- Analytics data

Components Used:

- `List`, `Card`, `Timeline` (Ant Design)

5. Unresolved Requests (`/unresolved`)

Purpose: View history of unresolved and timed-out requests

Features:

- List of all unresolved requests
- Shows timeout reason (auto-timeout after 30 min or manual)
- Time elapsed tracking (creation to unresolved)
- Customer contact information
- Detailed view modal with full context
- Filter by date range
- Pagination support

Components Used:

- `Table`, `Tag`, `Modal`, `Typography`, `Button` (Ant Design)
- Custom timeout reason formatter

Components

RequestCard.jsx

Displays individual help request with:

- Question text
- Customer phone
- Time elapsed
- Status badge
- Action buttons (Resolve, Mark Unresolved)

AnswerForm.jsx

Modal form for answering requests:

- Answer textarea (required)
- Category select (required)
- Supervisor notes (optional)
- Submit/Cancel buttons

KnowledgeBaseTable.jsx

Table displaying KB entries:

- Columns: Question, Answer, Category, Usage Count, Source, Actions
- Inline editing
- Delete confirmation
- Search/filter functionality

Layout.jsx

Main app layout:

- Top navigation bar with branding
- Sidebar menu with navigation:
  - Dashboard
  - Pending Requests (with badge count)
  - Resolved Requests
  - Unresolved Requests
  - Knowledge Base
- Content area
- WebSocket connection indicator

Hooks

useWebSocket.js

Custom React hook for Socket.io integration:

```
const { connected, stats } = useWebSocket();

// Listens for events:
// - new_help_request
// - request_resolved
// - kb_updated
// - request_timeout_warning
```

```
// Auto-reconnects on disconnect
// Provides connection status
```

Services

api.js

Axios client with base configuration:

- Base URL: `http://localhost:3000/api`
- Timeout: 10 seconds
- Error interceptors
- Request/response logging

---

4. Future Improvements

1. Twilio Integration for Live Calls

Current State: Webhooks simulate SMS/email notifications

Proposed Enhancement:

- Integrate **Twilio Programmable Voice** for real phone calls
- Integrate **Twilio SMS** for customer text notifications
- Use **Twilio Functions** as webhook endpoints

Implementation:

```
// NotificationService.js enhancement
async notifyCustomer(phone, message) {
  const twilioClient = require('twilio')(
    process.env.TWILIO_ACCOUNT_SID,
    process.env.TWILIO_AUTH_TOKEN
  );

  await twilioClient.messages.create({
    body: message,
    from: process.env.TWILIO_PHONE_NUMBER,
    to: phone
  });
}
```

Benefits:

- Real SMS notifications to customers
- Actual phone call routing
- Call recording and transcription
- Multi-channel support (voice + SMS)

2. Live Call Transfer

Feature: If supervisor is available during a call, transfer live instead of async escalation

Flow:

1. Agent detects supervisor is online (WebSocket presence)
2. Agent asks: "Would you like me to transfer you to a specialist?"
3. If yes, use LiveKit room transfer to connect supervisor
4. Supervisor joins same LiveKit room
5. Agent gracefully exits
6. Live conversation continues

Technical Requirements:

- Supervisor presence tracking (Socket.io rooms)
- LiveKit room management API
- Frontend audio/video interface for supervisors
- Call queue management

3. Multi-Language Support

Enhancement: Support multiple languages beyond English

Implementation:

- Use Deepgram's multilingual models (already configured)
- Add language detection in agent
- Store KB entries in multiple languages
- Use GPT-4 for translation
- Add language selector in frontend

4. Redis Caching Layer

Purpose: Reduce database load and improve response times

Use Cases:

- Cache frequently accessed KB entries
- Cache dashboard statistics
- Session management for WebSocket connections
- Rate limiting

5. Authentication & Authorization

Current State: No authentication (internal tool)

Production Requirements:

- JWT-based authentication
- Role-based access control (Admin, Supervisor, Viewer)
- API key authentication for agent
- OAuth integration (Google, Microsoft)
- Audit logging

Conclusion

This Human-in-the-Loop system transforms a traditional AI agent into a **learning system** that gets smarter with every interaction. By combining:

- **LiveKit** for natural voice conversations
- **Mem0** for semantic knowledge retrieval
- **MongoDB** for scalable persistence
- **Socket.io** for real-time collaboration
- **React** for intuitive supervisor interface

We've built a foundation that solves the hallucination problem while maintaining excellent customer experience. The system gracefully handles uncertainty by escalating to humans, then learns from their expertise to handle similar questions autonomously in the future.

Key Metrics:

- 30-minute SLA for escalations
- Semantic search with 80%+ confidence threshold
- Real-time notifications (<1 second latency)
- Automatic knowledge base growth
- Zero hallucinations (agent never guesses)

This architecture is production-ready and scales from 10 requests/day to 1,000+ with minimal changes.