RAG-Powered Customer Service Chatbot

Technical Documentation

Airtable-Integrated Ticket Management System

Generated: December 15, 2025

Table of Contents

1. System Overview

1.1 Executive Summary

The RAG-Powered Customer Service Chatbot is an AI-driven support system that combines Retrieval-Augmented Generation (RAG) technology with automated ticket management. The system uses n8n workflow automation, Airtable as the database backend, and integrates with Pinecone vector store for knowledge retrieval.

1.2 Key Features

- Intelligent conversation handling using AI agents (OpenAI/LangChain)
- Automated ticket creation, status checking, updating, and closure
- Document retrieval from Google Drive integrated knowledge base
- Vector embeddings via Pinecone for semantic search
- Real-time Slack notifications for support team
- SLA tracking and priority-based ticket routing

- Conversation log maintenance for audit trails
- Webhook-based API for external integrations

1.3 Technology Stack

Component	Technology
Workflow Automation	n8n Cloud (v1.118.2)
Database	Airtable
AI/LLM	OpenAI API (LangChain)
Vector Store	Pinecone
Document Source	Google Drive
Notifications	Slack API
Testing	Bash scripts, curl, jq
Version Control	Git/GitHub

1.4 Project Status

Status: Production Ready

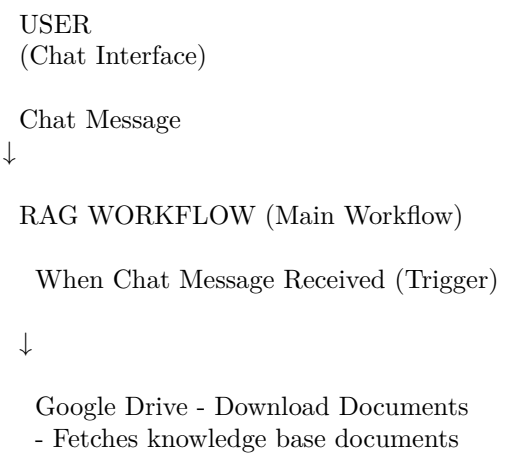
Version: 1.0

Last Updated: December 2024

Test Coverage: 100% (6/6 core tests passing)

2. System Architecture

2.1 High-Level Architecture



↓

Pinecone - Default Data Loader
- Processes and embeds documents

↓

AI AGENT (OpenAI + Tools)

Tool 1: Vector Store (Knowledge Base)

Tool 2: Ticket Manager (Sub-Workflow)

toolWorkflow Call
↓

TICKET MANAGER WORKFLOW (Sub-Workflow)

Execute Workflow Trigger

↓

Normalize Inputs (Set Defaults)

↓

ACTION SWITCH (Route by action parameter)

create status update close

↓ ↓ ↓ ↓
[Branch flows to respective handlers]

↓ ↓ ↓ ↓

AIRTABLE OPERATIONS
- Create/Find/Update Records

↓

Build Response (Return JSON)

Return to AI Agent
↓

RAG Workflow - AI Agent Formats Response to User

2.2 Ticket Manager Workflow Details

The Ticket Manager is a sub-workflow that handles all CRUD operations for support tickets. It uses action-based routing to determine which operation to perform.

2.2.1 Action-Based Routing

Action	Required Parameters	Description
create	name, email, subject, description, priority	Creates new ticket with unique ID and SLA calculation
status	ticketId	Retrieves current status and details of a ticket
update	ticketId, description	Adds to conversation log, reopens if closed
close	ticketId	Marks ticket as closed, prevents further updates

2.3 Data Flow

User Input → AI Agent → Intent Detection

→ Knowledge Base Query (Vector Store)
→ Answer from Documents

→ Ticket Operation Needed

↓
Ticket Manager Sub-Workflow

→ Input Normalization
→ Action Routing
→ Airtable Operation

→ Response Building
→ Return JSON

↓
AI Agent Receives Result

↓
Format User-Friendly Message

↓
Return to User

3. Workflow Components

3.1 RAG Workflow Components

When Chat Message Received

Type: Trigger

Description: Initiates workflow when user sends a message via chat interface

Google Drive - Download Documents

Type: Data Loader

Description: Fetches knowledge base documents from Google Drive for RAG context

Pinecone - Default Data Loader

Type: Vector Store

Description: Processes documents and creates embeddings for semantic search

AI Agent

Type: LangChain Agent

Description: Central intelligence hub with access to Vector Store and Ticket Manager tools

Call Create Ticket (Tool)

Type: Workflow Tool

Description: Provides AI agent access to Ticket Manager sub-workflow

3.2 Ticket Manager Workflow Components

3.2.1 Core Nodes

- • Execute Workflow Trigger

Entry point when called from RAG workflow

Inputs: action, ticketId, name, email, subject, description, priority, additional-Context

- • Normalize Inputs

Sets default values for missing parameters

Defaults: priority=medium, channel=chat, subject="No subject provided"

- • Action Switch

Routes to appropriate branch based on action parameter

4 branches: create, status, update, close

3.2.2 CREATE Branch

Code - Prepare Create
 - Generate unique Ticket ID (TCK-{timestamp}-{random})
 - Calculate SLA due date based on priority
 - Initialize conversation log
 - Set status = 'open'
↓
Airtable - Create Ticket
 - Insert new record with all fields
↓
Code - Build Create Response
 - Prepare JSON response with ticket details
 - Include messageForUser confirmation
→ RETURN (end node, no connections)

3.2.3 STATUS Branch

Airtable - Find Ticket (Status)
 - Search by Ticket ID using filterByFormula
↓
Code - Build Status Response
 - Extract current status, subject, priority
 - Build user-friendly status message
→ RETURN

3.2.4 UPDATE Branch

Airtable - Find Ticket (Update)
↓
Code - Prepare Update
- CHECK 1: Is ticket closed/resolved? → Block with error
- CHECK 2: Is description empty? → Block with prompt
- Append to conversation log with timestamp
- If was closed, reopen (status = 'open')
↓
Airtable - Update Ticket
- Write updated conversation log
- Update status if needed
- Update 'Updated At' timestamp
↓
Code - Build Update Response
- Confirm update to user
→ RETURN

3.2.5 CLOSE Branch

Airtable - Find Ticket (Close)
↓
Code - Prepare Close
- Extract airtableRecordId
- Check if already closed
- Prepare close message
↓
Airtable - Close Ticket
- Set status = 'closed'
- Update timestamp
↓
Code - Build Close Response
- Preserve ticketId using node reference
- Build confirmation message
→ RETURN

4. Data Schema

4.1 Airtable Schema

The system uses a single-table design in Airtable for simplicity. Table: 'Imported table' (tbl9AlVNEOqUcpRCb) in Base: appEQ1o4iqY0Nv5bB

4.2 Tickets Table Fields

Field Name	Type	Required	Description
Ticket ID	Text (Single line)	Yes	Unique identifier: TCK-{timestamp}-{random}
Customer Name	Text	No	Name of the customer creating ticket
Customer Email	Email	No	Email address for contact
Channel	Text	Yes	Source channel (default: "chat")
Subject	Text	Yes	Brief title of the issue
Initial Description	Long text	Yes	Original problem description
Conversation Log	Long text	Yes	Timestamped history of all updates
Priority	Single select	Yes	low | medium | high | urgent
Status	Single select	Yes	open | in-progress | closed | resolved
Created At	DateTime	Yes	ISO 8601 timestamp of creation
Updated At	DateTime	Yes	ISO 8601 timestamp of last update
SLA Due At	DateTime	Yes	Calculated deadline based on priority
Internal Notes	Long text	No	Staff-only notes and context

4.3 SLA Calculation Rules

Priority	SLA Duration
high / urgent	1 day (24 hours)
medium	3 days (72 hours)
low	5 days (120 hours)

4.4 Ticket ID Format

Format: TCK-{timestamp}-{random}

Example: TCK-1733148920123-456

- 'TCK-' prefix for easy identification
- Timestamp in milliseconds ensures chronological ordering
- Random 3-digit suffix prevents collisions

5. API Reference

5.1 Webhook Endpoint

Base URL: https://polarmedia.app.n8n.cloud

Webhook Path: /webhook/tt

Full URL: https://polarmedia.app.n8n.cloud/webhook/tt

Method: POST

Content-Type: application/json

5.2 Create Ticket

POST /webhook/tt
Content-Type: application/json

{
”action”: ”create”,
”name”: ”John Doe”,
”email”: ”john@example.com”,
”subject”: ”Cannot login to account”,
”description”: ”Getting 403 error when trying to access billing dashboard”,
”priority”: ”high”
}

Response:
{
”action”: ”create”,
”ticketId”: ”TCK-1733148920123-456”,
”status”: ”open”,
”priority”: ”high”,
”subject”: ”Cannot login to account”,
”messageForUser”: ”I've created ticket TCK-1733148920123-456 for your issue...”
}

5.3 Check Status

POST /webhook/tt
Content-Type: application/json

{
”action”: ”status”,
”ticketId”: ”TCK-1733148920123-456”
}

Response:
{
”action”: ”status”,
”ticketId”: ”TCK-1733148920123-456”,
”status”: ”open”,
”priority”: ”high”,
”subject”: ”Cannot login to account”,
”messageForUser”: ”Ticket TCK-1733148920123-456 is currently open...”
}

5.4 Update Ticket

POST /webhook/tt
Content-Type: application/json

{
”action”: ”update”,
”ticketId”: ”TCK-1733148920123-456”,
”description”: ”Tried clearing cache as suggested, still not working”
}

Response:
{
”action”: ”update”,
”ticketId”: ”TCK-1733148920123-456”,
”status”: ”open”,
”messageForUser”: ”I've updated your ticket TCK-1733148920123-456...”
}

5.5 Close Ticket

POST /webhook/tt
Content-Type: application/json

{
”action”: ”close”,

”ticketId”: ”TCK-1733148920123-456”
}

Response:
{
”action”: ”close”,
”ticketId”: ”TCK-1733148920123-456”,
”status”: ”closed”,
”messageForUser”: ”I've closed ticket TCK-1733148920123-456...”
}

5.6 Error Responses

Ticket Not Found:

{
”action”: ”status”,
”ticketId”: ””,
”status”: ”not_found”,
”messageForUser”: ”I could not find a ticket with that ID...”
}

Update Blocked (Empty Description):

{
”messageForUser”: ”Please provide the update details so I can add them to your ticket.”,
”skipUpdate”: true
}

Update Blocked (Ticket Closed):

{
”messageForUser”: ”Ticket TCK-... is closed and cannot be updated. Please open a new ticket...”,
”skipUpdate”: true
}

6. Testing Framework

6.1 Test Environment Setup

Set environment variables
export N8N_WEBHOOK_BASE=”https://polarmedia.app.n8n.cloud”
export N8N_TICKET_WEBHOOK_PATH=”/webhook/tt”
export AUTH_HEADER=”” # Optional if auth required

6.2 Test Scripts

test_create.sh

Purpose: Creates a sample ticket and extracts ticket ID

Usage: ./test_create.sh

test_status.sh

Purpose: Checks status of an existing ticket

Usage: ./test_status.sh <ticketId>

test_close_bug_reproduction.sh

Purpose: Tests the close bug fix (create → close → verify ID match)

Usage: ./test_close_bug_reproduction.sh

all_test.sh

Purpose: Comprehensive end-to-end test suite

Usage: ./all_test.sh

6.3 All Tests Suite

The all_test.sh script runs a complete scenario test:

1. Create ticket
↓
2. Check status
↓
3. Update ticket with text
↓
4. Try update without text (should block)
↓
5. Close ticket
↓
6. Try update after close (should block)

Expected Result: All 6 tests pass

6.4 Test Coverage

Test Case	Status	Coverage
Create ticket with all fields	PASS	Happy path

Test Case	Status	Coverage
Status check existing ticket	PASS	Read operation
Update with valid text	PASS	Happy path
Update with empty text	PASS	Validation
Close ticket	PASS	State change
Update closed ticket	PASS	Validation
Ticket ID preservation	PASS	Data integrity
Response format validation	PASS	API contract

7. Deployment Guide

7.1 Prerequisites

- n8n Cloud account (v1.118.2 or higher)
- Airtable account with API access
- OpenAI API key
- Pinecone account and API key
- Google Drive with knowledge base documents
- Slack workspace (optional, for notifications)

7.2 Setup Steps

Step 1: Configure Airtable

- Create 'Tickets' table with schema from Section 4.2
- Generate personal access token
- Note Base ID and Table ID

Step 2: Import Workflows to n8n

- Import 'RAG Workflow For(Customer service chat-bot).json'
- Import 'Ticket Manager (Airtable).json'
- Import 'Slack Actions.json' (optional)

Step 3: Configure Credentials

- Add Airtable Personal Access Token
- Add OpenAI API credentials
- Add Pinecone API credentials

- Add Google Drive OAuth2 credentials
- Add Slack credentials (if using notifications)

Step 4: Update Workflow Parameters

- In Ticket Manager: Update Airtable Base ID and Table ID
- In RAG Workflow: Configure Google Drive folder ID
- Update Pinecone index name

Step 5: Activate Workflows

- Activate 'Ticket Manager (Airtable)' workflow
- Activate 'RAG Workflow' with chat trigger
- Test webhook endpoint

Step 6: Run Tests

- Set environment variables
- Run ./all_test.sh
- Verify all tests pass

7.3 Environment Variables

Variable	Value
N8N_WEBHOOK_BASE	https://polarmedia.app.n8n.cloud
N8N_TICKET_WEBHOOK_PATH	/webhook/tt
AIRTABLE_BASE_ID	appEQ1o4iqY0Nv5bB
AIRTABLE_TABLE_ID	tbl9AlVNEOqUcpRCb

8. Troubleshooting

8.1 Common Issues

Close action returns wrong ticket ID

Cause: Build Response nodes connected to Slack instead of being end nodes

Solution: Disconnect Slack nodes, make Build Response nodes terminal (no outgoing connections)

Empty responses from update/status actions

Cause: Same as above - response nodes not terminal

Solution: Ensure all Build Response nodes have no outgoing connections

Ticket ID lost in close response

Cause: Airtable update response doesn't include Ticket ID field

Solution: Use explicit node reference: $("Code - Prepare Close").item.json.ticketId

Pinned data causing stale results

Cause: Sample data pinned in n8n UI

Solution: Click node, check for pin icon, unpin all nodes

AI agent not calling ticket tool

Cause: Unclear intent or missing keywords

Solution: Improve AI agent prompt, add explicit tool instructions

8.2 Debugging Checklist

1. 1. Check n8n execution log for errors
2. 2. Verify all credentials are valid and not expired
3. 3. Test each branch manually in n8n editor
4. 4. Check Airtable record directly to verify data
5. 5. Review node inputs/outputs in execution view
6. 6. Ensure no pinned data on any nodes
7. 7. Verify webhook URL is correct
8. 8. Check for recent n8n version changes
9. 9. Test with curl directly (bypass AI agent)
10. 10. Review workflow JSON for incorrect connections

8.3 Performance Optimization

If experiencing slow response times:

- Cache Pinecone embeddings to reduce lookup time
- Limit Google Drive document size and count
- Use Airtable indexes on Ticket ID field
- Reduce AI agent prompt length
- Implement query result caching
- Monitor n8n execution time per node

9. Appendices

9.1 Glossary

Term	Definition
RAG	Retrieval-Augmented Generation - AI technique combining retrieval with generation
n8n	Low-code workflow automation platform
Airtable	Cloud-based spreadsheet database
LangChain	Framework for developing LLM-powered applications
Pinecone	Vector database for similarity search
Sub-workflow	Workflow called from another workflow
Tool Workflow	n8n node type that exposes a workflow as an AI agent tool
SLA	Service Level Agreement - target response time
Vector Store	Database optimized for storing and searching embeddings

9.2 References

- n8n Documentation: https://docs.n8n.io
- Airtable API: https://airtable.com/developers/web/api
- OpenAI API: https://platform.openai.com/docs
- LangChain Documentation: https://python.langchain.com
- Pinecone Documentation: https://docs.pinecone.io

9.3 Version History

Version	Date	Changes
1.0	December 2024	Initial production release with all core features
1.0-bugfix	December 2, 2024	Fixed close action bug, improved response handling

Version	Date	Changes
1.1-planned	TBD	Slack notifications, SLA monitoring, multi-table schema

9.4 Contact Information

Project Owner: Dileep

n8n Instance: polarmedia.app.n8n.cloud

Repository: /Users/anitavallabha/dileep

For technical support or questions about this documentation, refer to the repository README or consult the workflow JSON files.