Supozy AI

Intelligent Customer Support System

Complete Technical Documentation

Version 1.0 | September 2025

Table of Contents

1. Overview

1.1 Introduction

Supozy AI is an advanced, intelligent customer support system designed to revolutionize how organizations handle customer queries. By leveraging cutting-edge semantic search technology and AI-powered information retrieval, Supozy AI automatically processes and responds to customer inquiries using an organization's existing documentation and historical support data.

> **Mission Statement:** To reduce manual support workload by 70% while improving response accuracy and customer satisfaction through intelligent automation.

1.2 Problem Statement

Modern customer support teams face several critical challenges:

- High volume of repetitive queries that drain support resources
- Inconsistent responses across different support agents
- Long response times during peak hours
- Difficulty in maintaining up-to-date knowledge bases
- Scaling support operations as business grows

1.3 Solution

Supozy AI addresses these challenges through an intelligent, automated system that:

- Learns from every resolved support ticket
- Provides instant, accurate responses using semantic understanding
- Maintains consistency across all customer interactions
- Scales effortlessly with business growth
- Continuously improves through machine learning

1.4 Target Audience

- **SaaS Companies:** Businesses with high support volumes and documentation
- **E-commerce Platforms:** Organizations needing 24/7 customer support
- **Tech Companies:** Teams with complex technical documentation
- **Enterprise Organizations:** Large-scale operations requiring consistent support

2. Key Features

🔍 Semantic Search

Advanced vector-based search that understands context and intent, not just keywords. Finds relevant solutions even when queries are phrased differently.

🤖 Dual-Agent System

Specialized agents work in tandem: QueryAgent handles inquiries while EmbeddingAgent builds the knowledge base.

💬 Conversation Management

Complete conversation tracking with status management (active, closed, escalated) for comprehensive support workflows.

📊 Vector-Powered Retrieval

Utilizes 1024-dimensional embeddings with cosine similarity for precise information matching and retrieval.

📚 Self-Learning System

Automatically indexes resolved tickets to continuously expand and improve the knowledge base.

⚡ Real-Time Processing

FastAPI backend ensures lightning-fast response times and efficient query processing.

2.1 Technical Capabilities

- **Natural Language Understanding:** Processes queries in natural language without requiring specific keywords
- **Context-Aware Responses:** Maintains conversation context for multi-turn interactions
- **Multi-Source Knowledge:** Combines documentation and historical ticket data
- **Scalable Architecture:** Handles thousands of concurrent conversations
- **Real-Time Indexing:** New solutions are immediately available for future queries

2.2 Business Benefits

- **70% Reduction** in manual support workload
- **90% Faster** average response times
- **100% Consistent** answers across all queries
- **24/7 Availability** without additional staffing costs
- **Continuous Improvement** with every interaction

3. System Architecture

3.1 High-Level Architecture

Supozy AI follows a modern, microservices-inspired architecture with clear separation of concerns:

Architecture Diagram: See supozy.drawio.png for detailed visual representation

3.2 Component Overview

Frontend Layer

- **Technology:** Node.js v22.14.0, Modern JavaScript Framework
- **Responsibilities:** User interface, conversation display, real-time updates
- **Communication:** REST API calls to backend services

Backend Layer

- **Technology:** Python 3.10, FastAPI
- **Responsibilities:** Business logic, agent orchestration, API endpoints
- **Features:** Async processing, request validation, authentication

Agent Layer

- **QueryAgent:** Handles incoming queries, performs semantic search
- **EmbeddingAgent:** Processes and indexes resolved tickets

Data Layer

- **Database:** MariaDB with vector support
- **Vector Store:** Optimized for similarity search
- **MCP Server:** MariaDB Model Context Protocol integration

3.3 Data Flow

1. **User Query:** Customer submits question through frontend
2. **Query Processing:** Backend receives and validates request
3. **Semantic Search:** QueryAgent generates embedding and searches vector store
4. **Result Retrieval:** Most relevant solutions are retrieved
5. **Response Generation:** AI composes appropriate response
6. **Delivery:** Response sent back to user interface
7. **Learning:** Resolved conversation indexed by EmbeddingAgent

3.4 Technology Stack

Component	Technology	Version	Purpose
Backend Framework	FastAPI	Latest	REST API, async processing
Programming Language	Python	3.10	Core business logic
Database	MariaDB	Latest	Data persistence, vector storage
Frontend Runtime	Node.js	22.14.0	JavaScript execution
Package Manager	pnpm	Latest	Frontend dependencies
Python Package Manager	uv	Latest	Backend dependencies
MCP Integration	MariaDB MCP	Latest	Database connectivity

4. Agent System

4.1 QueryAgent

The QueryAgent is the primary interface for handling customer inquiries. It employs sophisticated natural language processing and semantic search capabilities.

Core Responsibilities

- **Query Understanding:** Analyzes and interprets customer questions
- **Embedding Generation:** Converts queries into 1024-dimensional vectors
- **Semantic Search:** Matches queries against stored knowledge using cosine similarity
- **Context Management:** Maintains conversation history and context
- **Response Formulation:** Generates appropriate, helpful responses

Search Algorithm

1. Receive user query
2. Generate query embedding using same model as EmbeddingAgent
3. Perform vector similarity search in database
4. Retrieve top-k most similar documents (typically k=5)
5. Rank results by relevance score
6. Compose response using retrieved context

> **Performance:** QueryAgent typically responds within 100-300ms, including database query time.

4.2 EmbeddingAgent

The EmbeddingAgent operates in the background, continuously building and maintaining the knowledge base by processing resolved support tickets.

Core Responsibilities

- **Document Processing:** Extracts relevant information from resolved tickets

- **Vector Generation:** Creates 1024-dimensional embeddings
- **Metadata Management:** Attaches context and tags to embeddings
- **Storage:** Persists embeddings in vector store
- **Index Optimization:** Maintains efficient search indices

Embedding Process

1. Monitor for resolved conversations
2. Extract conversation text and metadata
3. Clean and preprocess text
4. Generate embedding vector
5. Store in conversation_embeddings table
6. Update search indices

Metadata Structure

Each embedding includes JSON metadata for enhanced retrieval:

```
{
  "conversation_id": "uuid-string"
}
```

4.3 Agent Collaboration

The two agents work in a symbiotic relationship:

- QueryAgent consumes knowledge that EmbeddingAgent produces
- Better queries lead to better resolutions, which improve embeddings
- System continuously improves through this feedback loop

5. Installation & Setup

5.1 Prerequisites

> ⚠ **Important:** Ensure all prerequisites are installed before proceeding with setup.

System Requirements

- **Operating System:** Linux, macOS, or Windows with WSL2
- **RAM:** Minimum 8GB (16GB recommended)
- **Disk Space:** 5GB free space
- **Internet Connection:** Required for dependency installation

Software Requirements

Software	Version	Installation
Python	3.10	python.org
Node.js	22.14.0	nodejs.org
pnpm	Latest	npm install -g pnpm
uv	Latest	https://github.com/astral-sh/uv
MariaDB	Latest	https://github.com/MariaDB/mariadb-docker
MariaDB MCP	Latest	[install instructions](https://github.com/MariaDB/mcp)

5.2 Clone Repository

```
git clone https://github.com/PratikDavidson/supozy-ai.git
cd supozy-ai
```

5.3 Backend Setup

Step 1: Install Dependencies

```
cd backend
uv sync
```

Step 2: Configure Environment

Create a `.env` file in the backend root directory as per `.env_template`.

```
# Database Configuration
        DB_HOST=localhost
        DB_PORT=3306
        DB_NAME=supozy_db
        DB_USER=your_username
        DB_PASSWORD=your_password

        # Grok API Key
        GROQ_API_KEY=api_key

        # MCP Server URLs
        MARIADB_MCP_URL=mcp_url
```

Step 3: Start Backend Server

```
uvicorn main:app --reload
```

✓ **Success:** Backend server should now be running at http://localhost:8000

5.4 Frontend Setup

Step 1: Install Dependencies

```
cd ../frontend
```

```
pnpm install
```

Step 2: Configure Environment

Create a `.env` file in the frontend root directory as per `.env_template`.

```
# API Configuration
NEXT_PUBLIC_API_BASE=http://localhost:8000
```

Step 3: Start Development Server

```
pnpm run dev
```

✓ **Success:** Frontend should now be accessible at http://localhost:3000

5.5 Verification

Test your installation by:

1. Opening the frontend URL in your browser
2. Creating a test user account
3. Starting a new conversation
4. Sending a test query
5. Verifying you receive a response

6. Database Schema

6.1 Schema Overview

The Supozy AI database is designed for optimal performance with vector operations while maintaining relational integrity. It consists of four primary tables with well-defined relationships.

6.2 Table Definitions

users

Stores user account information and authentication details.

Column	Type	Constraints	Description
id	BINARY(16)	PK	Unique user identifier (UUID)
email	VARCHAR(255)	UNIQUE, NOT NULL	User's email address
name	VARCHAR(100)	-	User's display name
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Account creation timestamp

conversations

Manages conversation state and metadata for support interactions.

Column	Type	Constraints	Description
id	BINARY(16)	PK	Unique conversation identifier
user_id	BINARY(16)	NOT NULL, FK → users(id)	Owner of the conversation
status	ENUM('active','closed','escalated')	DEFAULT 'active'	Current conversation state
started_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	When conversation began
last_message_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Timestamp of most recent message

messages

Stores the content and metadata of all messages exchanged in a conversation.

Column	Type	Constraints	Description
id	BIGINT	PK, AUTO_INCREMENT	Unique message identifier
conversation_id	BINARY(16)	NOT NULL, FK → conversations(id) ON DELETE CASCADE	Parent conversation
sender	ENUM('user','assistant')	NOT NULL	Who sent the message
text	TEXT	NOT NULL	Message content
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	When message was sent

conversation_embeddings

Stores vector embeddings of support documents and resolved tickets for efficient semantic search and retrieval.

Column	Type	Constraints	Description
id	BIGINT	PK, AUTO_INCREMENT	Unique embedding identifier
document	TEXT	NOT NULL	Original text that was embedded
embedding	VECTOR(1024)	NOT NULL	1024-dimensional vector
metadata	JSON	-	Additional context and tags
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	When embedding was created

6.3 Indexes

Specialized indexes are critical for performance, especially for vector search.

- **conversation_embeddings**: Vector index on `embedding` column using cosine distance

6.4 Relationships

Entity-Relationship overview for primary foreign key dependencies.

```
users(1) ————————> (n)conversations
        conversations(1) ————————> (n)messages
```

7. Deployment Guide

7.1 Containerization (Docker)

The recommended deployment method is using Docker and Docker Compose for easy environment management.

```
# Build and run all services (MariaDB, Backend, Frontend)
        docker-compose up --build -d
```

> Configuration for Docker and production environments should be managed through production-specific `.env` files and Docker secrets.