Architecture Overview

This document provides a comprehensive overview of the Support Marketing Agent architecture, including system design, data models, technology choices, and scalability considerations.

Table of Contents

System Architecture

High-Level Overview

The Support Marketing Agent follows a **modern monolithic architecture** with clear separation of concerns.

```
┌──────────────────────────────────────────────────────────────┐
│                    Client (React SPA)                         │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐     │
│  │Dashboard │  │Campaigns │  │ Tickets  │  │Analytics │     │
│  └──────────┘  └──────────┘  └──────────┘  └──────────┘     │
└──────────────────────────────────────────────────────────────┘
                         │ tRPC over HTTP
                         ▼
┌──────────────────────────────────────────────────────────────┐
│                 Server (Node.js + Express)                    │
│  ┌────────────────────────────────────────────────────────┐  │
│  │              tRPC Router Layer                          │  │
│  │  ┌────────┐  ┌────────┐  ┌────────┐  ┌────────┐       │  │
│  │  │  Auth  │  │Campaign│  │ Ticket │  │   AI   │       │  │
│  │  └────────┘  └────────┘  └────────┘  └────────┘       │  │
│  └────────────────────────────────────────────────────────┘  │
│  ┌────────────────────────────────────────────────────────┐  │
│  │              AI Services Layer (NEW)                    │  │
│  │  • Groq API Client    • Ticket Classifier               │  │
│  │  • Response Generator • Context Builder                 │  │
│  └────────────────────────────────────────────────────────┘  │
│  ┌────────────────────────────────────────────────────────┐  │
│  │              Business Logic Layer                       │  │
│  │  • Email sending   • Workflow execution                 │  │
│  │  • Webhook handling                                     │  │
│  └────────────────────────────────────────────────────────┘  │
│  ┌────────────────────────────────────────────────────────┐  │
│  │              Data Access Layer (Drizzle ORM)            │  │
│  └────────────────────────────────────────────────────────┘  │
└──────────────────────────────────────────────────────────────┘
                         │
         ┌───────────────┼───────────────┬───────────────┐
         ▼               ▼               ▼               ▼
    ┌────────┐      ┌────────┐      ┌────────┐      ┌────────┐
    │  TiDB  │      │   S3   │      │ Redis  │      │ Groq   │
    │Database│      │Storage │      │ Queue  │      │  API   │
    └────────┘      └────────┘      └────────┘      └────────┘
```

Technology Stack

Layer	Technology	Purpose
Frontend	React 19 + TypeScript	Component-based UI
Styling	Tailwind CSS 4 + shadcn/ui	Design system
Backend	Express 4 + tRPC 11	Type-safe API
Database	MySQL/TiDB + Drizzle ORM	Relational data
AI	Groq API + Llama models	AI processing
Queue	BullMQ + Redis	Job processing
Auth	Manus OAuth 2.0	Authentication

AI Services Layer

The AI Services Layer provides intelligent automation for the helpdesk system, enabling automatic ticket classification, sentiment analysis, and response generation.

Architecture Overview

```
┌─────────────────────────────────────────────────────────────┐
│                    AI Services Layer                        │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ┌──────────────────┐    ┌──────────────────┐               │
│  │  Groq Service    │    │ Context Builder  │               │
│  │ (API Client)     │    │ (Customer Data)  │               │
│  └────────┬─────────┘    └────────┬─────────┘               │
│           │                       │                         │
│           ▼                       ▼                         │
│  ┌──────────────────────────────────────────┐               │
│  │         Ticket Classifier                │               │
│  │  • Category detection                    │               │
│  │  • Priority assessment                   │               │
│  │  • Sentiment analysis                    │               │
│  └──────────────────────────────────────────┘               │
│                      │                                      │
│                      ▼                                      │
│  ┌──────────────────────────────────────────┐               │
│  │        Response Generator                │               │
│  │  • Context-aware responses               │               │
│  │  • Tone customization                    │               │
│  │  • Order issue detection                 │               │
│  │  • Quick action suggestions              │               │
│  └──────────────────────────────────────────┘               │
│                      │                                      │
│                      ▼                                      │
│  ┌──────────────────────────────────────────┐               │
│  │    Knowledge Base & RAG (Phase 3)        │               │
│  │  • Vector Store (TF-IDF similarity)      │               │
│  │  • Knowledge Base CRUD                   │               │
│  │  • RAG Service (retrieval-augmented)     │               │
│  └──────────────────────────────────────────┘               │
│                      │                                      │
│                      ▼                                      │
│  ┌──────────────────────────────────────────┐               │
│  │   Feedback & Analytics (Phase 5)         │               │
│  │  • Feedback Service (ratings)            │               │
│  │  • Analytics Service (metrics)           │               │
│  │  • Performance tracking                  │               │
│  └──────────────────────────────────────────┘               │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Service Components

1. Groq Service (`server/services/ai/groqService.ts`)

The Groq Service is the API client that interfaces with Groq's hosted Llama models.

```
// Configuration
GROQ_API_KEY: string  // Environment variable
BASE_URL: 'https://api.groq.com/openai/v1'

// Available Models
- llama-3.3-70b-versatile  // Complex reasoning tasks
- llama-4-scout-16e        // Fast, lightweight tasks
```

Features:
- Automatic retry with exponential backoff
- Token usage tracking
- Latency metrics
- Error handling with fallbacks

2. Ticket Classifier (`server/services/ai/ticketClassifier.ts`)

Analyzes incoming tickets to automatically determine:

Output	Values	Description
Category	order_status, shipping, returns, billing, product, technical, general	Issue type classification
Priority	low, medium, high, urgent	Urgency assessment
Sentiment	positive, neutral, negative, frustrated	Customer emotional state
Confidence	0.0 - 1.0	Classification confidence score

Example Output:

```json
{
  "category": "shipping",
  "priority": "high",
  "sentiment": "frustrated",
  "confidence": 0.92,
  "reasoning": "Customer mentions delayed package and expresses frustration",
  "suggestedActions": ["check_tracking", "offer_expedited_shipping"]
}
```

3. Context Builder (`server/services/ai/contextBuilder.ts`)

Aggregates customer data to provide context for AI responses:

```
interface CustomerContext {
  customer: {
    id: number;
    name: string;
    email: string;
    isVip: boolean;        // 5+ orders OR $500+ LTV
    totalOrders: number;
    totalSpent: number;
    memberSince: Date;
  };
  recentOrders: Order[];    // Last 10 orders
  ticketHistory: Ticket[];  // Last 5 tickets
  emailEngagement: {        // 30-day window
    opened: number;
    clicked: number;
  };
}
```

VIP Detection Logic:
- 5+ completed orders, OR
- $500+ lifetime value

4. Response Generator (`server/services/ai/responseGenerator.ts`)

Generates contextual responses with multiple options:

Tone Options:

Tone	Use Case
`professional`	Standard business communication
`friendly`	Casual, personable interactions
`empathetic`	Frustrated or upset customers

Features:
- Order issue detection (delayed, damaged, missing, wrong item)
- Template fallback for instant responses
- Suggested quick actions
- Confidence scoring
- Response metadata (tokens used, latency)

5. Vector Store (`server/services/ai/vectorStore.ts`)

In-memory TF-IDF vector store for knowledge base similarity search:

```
// Key Methods
TFIDFVectorStore.indexDocument(id, content)  // Add document to index
TFIDFVectorStore.search(query, topK)         // Find similar documents
searchKnowledge(query)                        // Search knowledge articles
refreshKnowledgeIndex()                       // Rebuild full index
```

Features:
- Tokenization and TF-IDF weighting
- Cosine similarity matching
- Auto-indexes active knowledge articles

6. Knowledge Base Service (`server/services/ai/knowledgeBase.ts`)

CRUD operations for helpdesk knowledge articles:

```
// Key Methods
createArticle(title, content, category)
updateArticle(id, updates)
semanticSearch(query, limit)           // Via vector store
findRelevantKnowledge(ticketContent)   // For RAG integration
```

7. RAG Service (`server/services/ai/ragService.ts`)

Retrieval-Augmented Generation for knowledge-grounded responses:

```
// Key Methods
generateRAGResponse(ticket, tone)           // Single response with KB context
generateMultipleRAGResponses(ticket)        // Multiple tone variations
buildRAGContext(ticketContent)              // Knowledge injection
```

Features:
- Knowledge source tracking in metadata
- Confidence boosting based on article relevance
- Fallback to standard generation when no articles found

8. Feedback Service (`server/services/ai/feedbackService.ts`)

Collects agent feedback on AI responses:

```
// Key Methods
submitFeedback(interactionId, rating, wasUsed, wasEdited)
getFeedbackStats(organizationId, dateRange)
```

Tracked Metrics:
- Positive/negative ratings
- Whether response was used
- Edit distance from original

9. Analytics Service (`server/services/ai/analyticsService.ts`)

AI performance metrics and reporting:

```
// Key Methods
getOverviewMetrics(organizationId)     // Summary stats
getMetricsByCategory(organizationId)   // Breakdown by ticket type
getMetricsByTone(organizationId)       // Breakdown by tone
getTrendData(organizationId, days)     // Time series
```

Dashboard Metrics:
- Response acceptance rate
- Average confidence scores
- Usage by category/tone
- Feedback trends

Model Selection Strategy

```
// Complex tasks - use larger model
const COMPLEX_MODEL = 'llama-3.3-70b-versatile';
// - Ticket classification
// - Response generation
// - Sentiment analysis
// - RAG responses

// Fast tasks - use smaller model
const FAST_MODEL = 'llama-4-scout-16e';
// - Quick action suggestions
// - Simple categorization
// - Real-time suggestions
```

Prompt Engineering

Prompts are organized in `server/services/ai/prompts/`:

File	Purpose
`classification.ts`	E-commerce ticket categorization
`sentiment.ts`	Customer sentiment detection
`response.ts`	Response generation templates
`orderResponses.ts`	Order-specific issue templates

API Endpoints

Classification Router (`server/routers/aiClassification.ts`):

```
ai.classification.classifyTicket     // Classify a single ticket
ai.classification.batchClassify      // Classify multiple tickets
ai.classification.getSuggestions     // Get action suggestions
```

Response Router (`server/routers/ai.ts`):

```
ai.responses.generate               // Generate single response
ai.responses.generateMultiple       // Generate 3 tone options
ai.responses.getQuickActions        // Get quick action list
ai.responses.getTemplateResponse    // Get instant template
ai.responses.getCustomerContext     // Get customer profile
```

Database Schema (AI Fields)

Tickets Table Additions:

```
aiCategory      VARCHAR(50)   -- AI-classified category
aiPriority      VARCHAR(20)   -- AI-assessed priority
aiSentiment     VARCHAR(20)   -- Detected sentiment
aiConfidence    DECIMAL(3,2)  -- Classification confidence
aiProcessedAt   TIMESTAMP     -- When AI processed ticket
```

AI Interactions Table:

```
CREATE TABLE aiInteractions (
  id              INT PRIMARY KEY,
  ticketId        INT,
  interactionType ENUM('classification', 'response', 'suggestion'),
  modelUsed       VARCHAR(50),
  inputTokens     INT,
  outputTokens    INT,
  latencyMs       INT,
  confidence      DECIMAL(3,2),
  createdAt       TIMESTAMP
);
```

Data Model

Core Entities

- **Organizations** - Multi-tenant workspaces
- **Users** - Team members with roles
- **Contacts** - Customer database
- **Tickets** - Support requests with AI classification
- **Orders** - E-commerce order data
- **Campaigns** - Email marketing campaigns
- **Workflows** - Automation sequences

API Design

The platform uses **tRPC** for type-safe client-server communication.

Router Structure

```
appRouter = {
  auth: { ... },
  dashboard: { ... },
  contacts: { ... },
  campaigns: { ... },
  workflows: { ... },
  tickets: { ... },
  orders: { ... },
  integrations: { ... },
  analytics: { ... },
  ai: {                    // NEW
    classification: { ... },
    responses: { ... },
  },
}
```

Scalability & Performance

AI Service Optimization

- **Model caching**: Reuse connections to Groq API
- **Batch processing**: Classify multiple tickets in parallel
- **Template fallbacks**: Instant responses without API calls
- **Confidence thresholds**: Skip low-confidence classifications

Database Optimization

- Indexes on AI classification fields
- Composite indexes for common query patterns
- Denormalized counts to avoid aggregations

Security Considerations

AI Data Protection

- Customer PII is minimized in prompts
- AI interactions are logged for audit
- Confidence thresholds prevent low-quality responses
- Human review required for sensitive cases

Conclusion

The Support Marketing Agent architecture combines modern web technologies with AI-powered automation. The AI Services Layer enables intelligent ticket handling while maintaining the flexibility for human oversight and intervention.

For implementation details, see:
- IMPLEMENTATION_STATUS.md (IMPLEMENTATION_STATUS.md)
- docs/INTEGRATIONS.md (INTEGRATIONS.md)
- docs/WORKFLOWS.md (WORKFLOWS.md)