**Name:** Itte Aravind
LinkedIn | GitHub

# Customer Support Chatbot
# Project Documentation

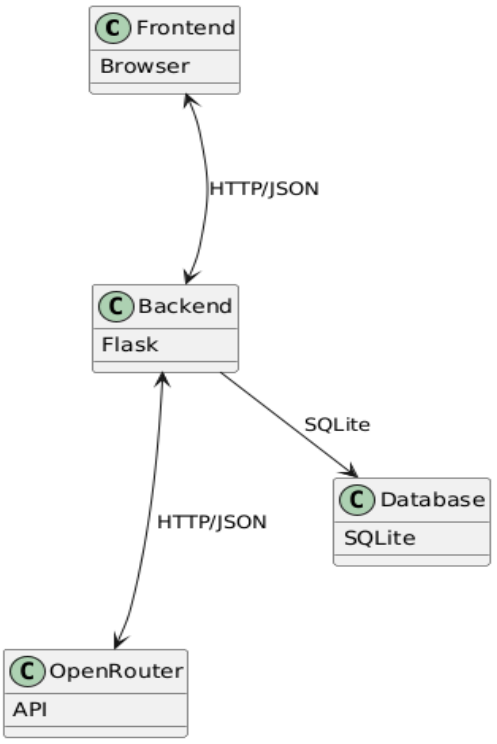
## Project Overview

This is a comprehensive customer support solution designed to provide instant assistance to users of the DataForge AI platform. This documentation covers the architecture, components, functionality, and implementation details of the chatbot system. [Project Link]

## Architecture

The DataForge AI Chatbot follows a client-server architecture with the following components:

High-Level Architecture




- **Frontend:** HTML, CSS, and JavaScript running in the browser
- **Backend:** Python Flask application
- **External API:** OpenRouter API for accessing DeepSeek's LLM
- **Database:** SQLite for storing chat analytics and history

# Backend Components

Core Components

1. **Flask Application (app.py)**

    - Initializes the web server and API endpoints
    - Manages session data for conversation context
    - Handles routing and request processing

2. **Knowledge Base Management**

    - Loads markdown files from the knowledge_base directory
    - Converts knowledge base content into context for the LLM
    - Provides fallback responses when API calls fail

3. **Query Categorization**

    - Analyzes user queries to determine their category
    - Categories include: product, features, pricing, use_cases, technical, support

4. **Database Management**

    - Initializes SQLite database
    - Provides connection handling
    - Stores analytics data and chat history

5. **LLM Integration**

    - Connects to DeepSeek's LLM via OpenRouter API
    - Formats messages with conversation history and knowledge context
    - Handles API errors and provides fallback responses

## Key Functions

- *load_knowledge_base():* Loads markdown files from the knowledge_base directory
- *create_knowledge_context()*: Formats knowledge base content for the LLM
- *categorize_query():* Determines the category of a user query
- *get_ai_response()*: Sends requests to the LLM and processes responses
- *track_analytics():* Records query and response data for analytics

# Frontend Components

## Pages

1. **Chat Interface (index.html)**

    - Main user interface for interacting with the chatbot
    - Displays conversation history
    - Provides input field and suggested questions

2. **Analytics Dashboard (analytics.html)**

    - Displays usage metrics and charts
    - Shows query categories distribution
    - Tracks satisfaction rates and fallback frequency

3. **Chat History Viewer (history.html)**

    - Displays past conversations for the current session
    - Shows metadata like timestamps, categories, and feedback
    - Provides session statistics

## Key JavaScript Files

1. **Main Chat Functionality (script.js)**

    - Handles user input and message sending
    - Manages conversation display
    - Provides feedback mechanisms

2. **Analytics Dashboard (analytics.js)**

    - Fetches analytics data from the backend
    - Renders charts and metrics
    - Handles data refreshing

3. **UI Components**

    - **Message Bubbles:** Display user and assistant messages
    - **Typing Indicator:** Shows when the assistant is generating a response
    - **Feedback Buttons:** Allow users to rate responses
    - **Suggested Questions:** Provide quick access to common queries
    - **Navigation Sidebar:** Enables switching between different sections

## Key Features

1. **Conversational AI**

   The chatbot leverages DeepSeek's large language model to provide natural, context-aware responses to user queries. Key aspects include:

   - **Context Awareness:** Maintains conversation history to provide relevant follow-up responses
   - **Knowledge Integration:** Uses a comprehensive knowledge base about DataForge AI
   - **Natural Language Understanding:** Interprets user questions regardless of phrasing

2. **Analytics Dashboard**

   The analytics dashboard provides insights into chatbot usage and performance:

   - Query Categories: Distribution of questions by topic
   - Satisfaction Rate: Percentage of responses rated positively
   - Fallback Rate: Frequency of fallback responses when the API fails
   - Total Queries: Count of all interactions

3. **Chat History**

   The chat history feature allows users to review past conversations:

   - **Chronological View:** Lists all interactions in reverse chronological order
   - **Metadata Display:** Shows timestamps, categories, and feedback for each interaction
   - **Session Statistics:** Provides aggregate metrics for the current session

4. **Fallback Mechanism**

   When the LLM API is unavailable or fails, the system provides predefined responses:

   - **Category-Based Fallbacks:** Different responses based on query category
   - **Graceful Degradation:** Maintains functionality even when external services fail
   - **Error Logging:** Records failures for troubleshooting

## API Endpoints

Chat Endpoint

```
POST /api/chat
```

Request Body:

```
{
  "message": "User's question here",
  "session_id": "unique_session_identifier"
}
```

Response:

```
{
  "response": "Assistant's response here",
  "chat_id": 123
}
```

Feedback Endpoint

```
POST /api/feedback
```

Request Body:

```
{
  "chat_id": 123,
  "rating": 1  // 1 for positive, 0 for negative
}
```

Response:

```
{
  "success": true
}
```

Analytics Endpoint

```
GET /api/analytics
```

Response:

```
{
  "categories": [
    {"category": "product", "count": 10},
```

```
    {"category": "features", "count": 15},
    {"category": "pricing", "count": 5}
  ],
  "fallback_rate": 12.5,
  "satisfaction": 0.85
}
```

Chat History Endpoint

```
GET /api/chat-history?session_id=unique_session_identifier
```

Response:

```
{
  "session_id": "unique_session_identifier",
  "history": [
    {
      "id": 123,
      "query": "User's question",
      "response": "Assistant's response",
      "category": "product",
      "is_fallback": false,
      "satisfaction": 1,
      "formatted_time": "2025-04-13 12:34:56"
    }
  ]
}
```

## Database Schema

The analytics data is stored in the chat_analytics table with the following structure:

```
CREATE TABLE chat_analytics (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id TEXT,
    query TEXT,
    response TEXT,
    category TEXT,
    satisfaction INTEGER,
    is_fallback BOOLEAN,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
)
```

# Chat History

The chat history feature allows users to review past conversations within their current session:

## Features

- **Session-Based History:** History is tied to a unique session ID
- **Metadata Display:** Shows timestamps, categories, and feedback
- **Fallback Indicators:** Clearly marks responses that used fallback mechanisms
- **Session Statistics:** Provides aggregate metrics for the session

## Implementation

- Session IDs are generated randomly and stored in localStorage
- History is retrieved from the backend via the /api/chat-history endpoint
- The frontend displays the history in reverse chronological order

# Deployment

The project is containerized using Docker for easy deployment:

## Components

- Backend Container: Python Flask application
- Frontend Container: Nginx serving static files
- Docker Compose: Orchestrates the containers

## Environment Variables

```
OPENROUTER_API_KEY: API key for accessing OpenRouter
SECRET_KEY: Secret key for Flask session encryption
```

## Future Enhancements

Potential improvements for future versions:

- **Multi-language Support:** Add capability to respond in different languages
- **Voice Interface:** Integrate speech-to-text and text-to-speech capabilities
- **Rich Responses:** Support images and formatted text in responses
- **Knowledge Base Expansion:** Automated ingestion of documentation updates
- **Advanced Analytics:** More detailed visualization of chatbot performance
- **User Authentication:** Support for authenticated users with personalized experiences
- **Integration with Ticketing Systems:** Escalate complex queries to human agents
- **Proactive Suggestions:** Offer suggestions based on user behavior patterns
- **Conversation Export:** Allow users to export their chat history
- **A/B Testing Framework:** Test different response strategies for optimization