

# Developer Documentation

Capgemini – Sentiment Analysis and Response Generation Application

## 1. Introduction

This documentation supports both end users and developers in understanding, using, and maintaining the Sentiment Analysis and Response Generation system developed for Capgemini. It includes high-level system goals, usage instructions, technical architecture, testing procedures, and a glossary.

## 2. User Documentation

### 2.1 Getting Started

To begin using the Sentiment Analysis and Response Generation System, users should open the provided demo application via a browser or local interface, depending on deployment. No login is required for basic functionality. Input can be entered manually or uploaded as a CSV file containing customer feedback data in a column labeled 'message'. The platform is designed for ease-of-use with minimal technical expertise required.

### 2.2 Using the API Interface

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1-Screenshot of Web App Submission UI

The user-facing interface provides a text box for entering a single customer message, along with an option to upload a CSV file for batch processing. Once submitted, the application sends the input to a backend Flask API which returns the following: a sentiment classification (positive, negative, neutral), an AI-generated response, and a response alignment score. Users receive results displayed directly on-screen in an organized and editable format.

### 2.3 Understanding Output and Responses

Each submission yields a set of results:

* AI-Generated Response: A context-aware reply crafted to match the sentiment and emotional tone of the user's message.
* Response Quality Score: Measures how well the generated response aligns with the detected sentiment and brand tone, using metrics like F1-Score.
* Sentiment Label: Classifies the input as Positive, Negative, or Neutral.
* Empathetic AI-Generated Response: Automatically generated reply reflecting emotional awareness.
* Response Score: Evaluates alignment with customer sentiment and brand tone using F1-Score and similar metrics.
* Sarcasm Detection: Flags whether the input contains sarcasm, helping avoid misinterpretation in automated replies.
* Emotion Detection: Identifies the underlying emotion (e.g., anger, joy, sadness) expressed in the input message.
* Aspect-Based Sentiment Analysis: Breaks down the input by specific topics or features and assigns sentiment to each (e.g., "battery life – negative", "screen quality – positive").
* Dashboard Visualization: Displays all processed submissions, sentiment trends, and analysis breakdowns in an interactive interface for monitoring and review.

### 2.4 CMS Backend: Customizing Responses

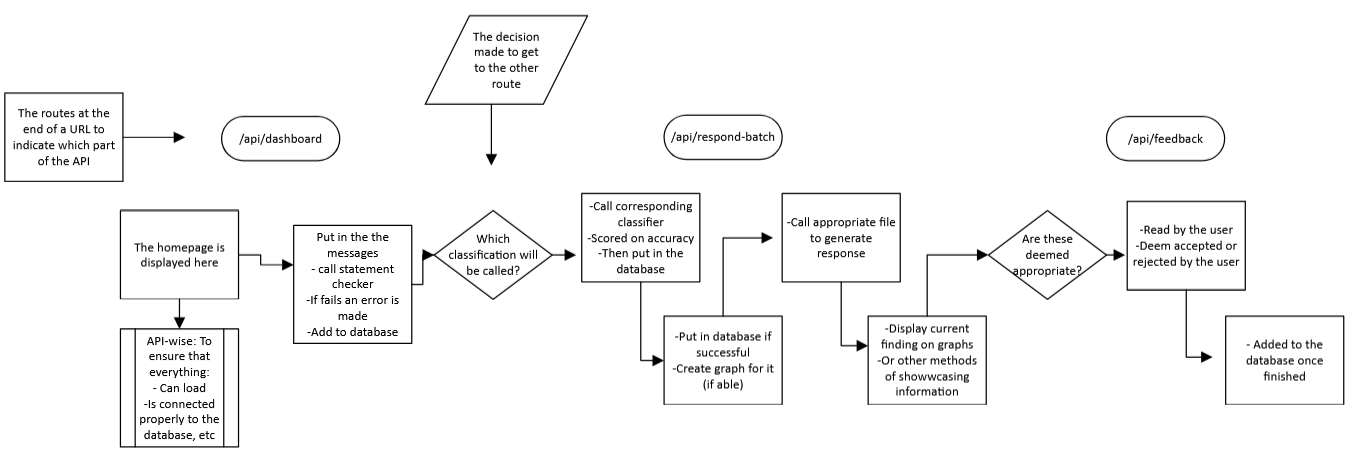
The CMS backend allows Capgemini team members to adjust tone, language, and brand-related messaging. Admins can configure rules such as enforcing formal tone, regional phrasing, or prioritizing empathy in responses. The CMS also supports saving branded response templates for repeated use. All configurations influence how AI-generated responses are constructed.

### 2.5 Common Issues and Troubleshooting

If the application fails to produce a response, check the following:  
- Ensure the uploaded CSV contains a 'message' column with valid content.  
- Confirm a stable internet connection for accessing the Azure-hosted API.  
- If responses seem off-tone, review CMS branding rules for conflicting configurations.  
For further help, refer to the Contact and Support section or consult developer documentation.

* **AI-generated responses were not appearing after submitting feedback**  
   *Cause:* Results were not yet available before rendering.  
   *Fix:* Ensure Sentiment Analysis Report only displays once results are returned from the backend.
* **AI responses seem generic or irrelevant. Responses adhered to fallback.**  
   *Cause:* System may still be using cached or static fallback logic.  
   *Fix:* Confirm that Phi-3 is correctly configured via Azure AI Studio, and that PHI3\_ENDPOINT and PHI3\_KEY are set in App Service.
* **Response and input text not clearly separated in UI**  
   *Cause:* Original design lacked labels and structure.  
   *Fix:* Updated frontend layout to display feedback and response side-by-side with clear labels in the ResponseDisplay component.

## 3. Developer Documentation



### 3.1 System Architecture Overview

**Classifier Model Testing: Naïve Bayes Model for Emotion Detection**

**Training & Testing Setup:**

* The model was trained using a dataset of **42,000+ user feedback entries** with an **80/20 train-test split**.
* Due to the original dataset having **279+ unique emotion labels**, label reduction was performed. Filtering the dataset down to the **six most frequent emotions** (≥2,000 entries each) improved performance and stability.
* Final label set: anger, joy, anticipation, neutral, disgust, sadness.

**Performance Metrics:**

* **Accuracy:** 55.81%
* **Precision:** 0.56
* **Recall:** 0.56
* **F1-Score:** 0.55
* **Runtime:** ~86 seconds on full dataset

**Evaluation Notes:**

* The Naïve Bayes model showed consistent detection accuracy across all six emotions, making it a strong choice for integration despite its simplicity.
* Outperformed random guessing (~18%) by over 200%.
* Example test case: *“I’m so frustrated with the late delivery”* → correctly classified as **anger**.
* Model and vectorizer were saved as .pkl files for loading into the Flask API.
* The model was trained and evaluated using the train\_classifier.py script, which also serializes the classifier and vectorizer for deployment.

**Challenges:**

* **High label diversity** was a major bottleneck to early accuracy. Reducing the label set significantly improved model clarity and generalization.
* Naïve Bayes was chosen for its efficiency and interpretability over slower but more accurate transformer models.

### 3.2 Technology Stack and Dependencies

Lists all frameworks, libraries, and APIs used (e.g., Flask, Azure App Services, Hugging Face, OpenAI, Cosmos DB). Includes installation and environment setup guidance.

The system utilizes the following technologies:

* **Flask**: Web framework for building the API.
* **Azure App Services**: Cloud platform for hosting the API and managing the database.
* **Hugging Face Transformers**: For advanced sentiment and sarcasm detection.
* **Scikit-learn**: Used for training the sentiment classification model.
* **PyODBC**: For database connectivity (SQL Server).
* **NLTK**: For sentiment analysis with VADER (Valence Aware Dictionary and Sentiment Reasoner).
* **React with Vite:** Frontend framework for building a fast and responsive web interface.
* **Naive Bayes Model:** Custom-trained model for emotion detection.
* **Phi-3 Model:** Lightweight LLM used for generating context-aware, empathetic responses.

### 3.3 API Reference and Endpoints

Presents full endpoint documentation including method types (GET, POST), request/response format, error codes, and authentication (if applicable).

/api/dashboard (GET)

* Confirms the API is running, the database is running and connected, and that the frontend and backend are communicatingcapsense (GET, POST)
* Can fetch previous records

/api/respond-batch (POST)

* Is given the file, from the frontend in JSON
* Ensures the input is in strings
* Each text in the input is given to the classifiers then placed into objects
* Obtains the AI response and F1 – score
* The captured results are given to the database and to the frontend
* Otherwise will given an error if a fault occurs

/api/batch-analyze (POST)

* Used for single line without the database in the circumstance that it is not running
* Able to obtain the classifiers, AI responses, and F1 scores

/api/feedback (POST)

* Validates the inputs
* Stores the feedback from the users to the database
* Can run without the database connected
* Detects sarcasm in a given text using a pre-trained model.

### 3.4 CMS Backend Design and Integration

Describes how the CMS backend stores and applies branding rules to the AI responses. Includes database schema, admin interface behavior, and integration into the API.

### 3.5 Deployment Instructions (Azure)

Step-by-step deployment instructions for setting up the system on Azure, including App Services, Functions, DBs, and securing API keys.

**Integrating Phi-3 Model for Response Generation:**

* The Phi-3 model was set up using Azure's AI model endpoints or loaded directly into the Flask API depending on deployment size/performance constraints.
* The model is called from within the Flask API after sentiment/emotion/sarcasm detection is complete, generating a context-aware response.
* Ensure any environment variables or secrets related to Phi-3 model configuration (e.g., model path or API credentials) are added securely in **Azure App Service Configuration Settings**.
* If the model is hosted separately (e.g., in an Azure Function or Container App), make sure the Flask API has the correct internal endpoint or access key to communicate with it.

**Phi-3 Model Integration for Empathetic Response Generation**

The CapSense backend integrates the **Phi-3 Mini (phi-3-mini-4k-instruct)** model via **Azure AI Studio** to generate empathetic, context-aware replies to customer feedback. Unlike earlier static responses, this deployment dynamically tailors outputs based on classifier results (sentiment, emotion, sarcasm).

Phi-3 is not used for classification but for **response generation**, ensuring the tone and content reflect the customer’s emotional state and feedback tone.

Phi-3 is not used for classification but for **response generation**, ensuring the tone and content reflect the customer’s emotional state and feedback tone.

**Implementation**

* The logic is implemented in phi3resgen.py.
* The generate\_response() function constructs a prompt using:
  + **Customer text**
  + **Classifier outputs**: Sentiment, Emotion, and Sarcasm
* Returns both a response\_text and a basic empathy\_score (derived from response length).
* Returns both a response\_text and a basic empathy\_score (derived from response length).

**Environment Configuration**

* Set the following in Azure App Service:
  + PHI3\_KEY – The API key from Azure AI Studio
  + PHI3\_KEY – The API key from Azure AI Studio

**Azure Deployment Steps**

1. In Azure AI Studio:
   * Go to **Models** → Search for **phi-3-mini-4k-instruct**
   * Click **Deploy** → Choose *Serverless* compute
   * Once deployed, copy **endpoint URL** and **API key**
2. In Azure App Service (backend):
   * Navigate to **Configuration** → **Application Settings**
   * Add environment variables: PHI3\_ENDPOINT, PHI3\_KEY
3. Test the system:
   * Expect a full JSON response with sentiment, classifiers, and ai\_response.response\_textapi/respond
   * Expect a full JSON response with sentiment, classifiers, and ai\_response.response\_text

**Cost and Runtime**

* Phi-3 response latency is minimal and runs fully serverless (no VM required)
* Phi-3 response latency is minimal and runs fully serverless (no VM required)

**Example Prompt Logic (in phi3resgen.py):**

\*\*Customer Feedback\*\*: "The delivery was so late, again!"

\*\*Sentiment\*\*: negative

\*\*Emotion\*\*: anger

\*\*Sarcasm Detected\*\*: True

## 4. Azure SQL Database

# 4.1 Database Overview

Purpose:  
 The SentimentAnalysisDB stores, manages, and processes sentiment-related data uploaded via the CapSense AI web application. This includes storing user-uploaded text, CSV-imported sentiment datasets, and NLP-analyzed results.  
   
Main Functionalities:  
 - Stores raw text and sentiment metadata  
 - Tracks user-uploaded documents  
 - Holds pre-processed and analyzed sentiment results  
 - Integrates with Flask API for real-time queries  
 - Powers Power BI dashboards for visualization

# 4.2 Database Schema

4.2.1 Tables

|  |  |
| --- | --- |
| Table Name | Description |
| Users | Contains application users |
| UploadedFiles | Metadata and status of uploaded files |
| SentimentResults | Stores results from NLP sentiment analysis |
| Logs | Tracks access and system-level events |

4.2.2 Example Table Definition – SentimentResults

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| ID | INT | Primary key |
| FileID | INT | Foreign key to UploadedFiles |
| SentimentLabel | VARCHAR(50) | e.g., Positive, Neutral, Negative |
| SentimentScore | FLOAT | Polarity score (-1.0 to 1.0) |
| ProcessedDate | DATETIME | Date the analysis was completed |

# 4.3 Integration with Application

Backend: Flask API connects to this SQL database using pyodbc or SQLAlchemy with an Azure connection string.  
 Frontend: Users upload CSV files or text via a Tkinter GUI or web portal. Data is validated, then inserted into the database.  
 Azure Connection String Example:  
   
 Driver={ODBC Driver 17 for SQL Server};  
 Server=tcp:1sqlcapsenseserver.database.windows.net,1433;  
 Database=SentimentAnalysisDB;  
 Uid=your\_username;  
 Pwd=your\_password;  
 Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;

# 4.4 Security and Access

- Authentication: Azure Active Directory or SQL Auth  
 - Firewall Rules: Restricted to app service IP ranges  
 - Encryption: Transparent Data Encryption (TDE) enabled  
 - Backups: Geo-redundant backups (daily snapshots)

# 4.5 Performance Monitoring

- Azure SQL Insights: Monitoring enabled  
 - DTU Usage: Scaled to support API traffic  
 - Query Performance: Optimized with indexes and stored procedures

# 4.6 Maintenance and Scaling

- Auto-Pause: Disabled (always-on for web app usage)  
 - Elastic Pool: No (Single database setup)  
 - Manual Scaling: Can be upgraded to higher service tier (e.g., S1 → S3)  
 - Scheduled Maintenance: Azure-managed

# 4.8 Example Stored Procedure

CREATE PROCEDURE GetSentimentSummary  
 AS  
 BEGIN  
 SELECT SentimentLabel, COUNT(\*) AS Count  
 FROM SentimentResults  
 GROUP BY SentimentLabel;  
 END

## 5. Change Log and Known Issues

Lists all major changes since the initial version, with dates and responsible team members. Also includes known limitations or bugs that may affect performance.

Flask API: Added Input Validation and Error Logging

* Introduced robust input validation to ensure that all incoming requests conform to expected formats, preventing invalid data from causing system errors.
* Integrated comprehensive error logging to track any issues or failures, aiding in quicker debugging and system performance analysis.

Created Phi 3 Response Generation

* Developed a response generation system that uses data from other classifiers to send out a basic preset response based on a score. This functionality allows the system to evaluate incoming inputs and determine an appropriate response by referencing predefined responses associated with different classification scores.

Phi 3 Response Generation Revamp

* Transitioned from preset responses to contextually relevant, dynamically generated responses.
* This update significantly enhances the user experience by allowing the system to generate more personalized and accurate replies based on the given context.

Known Issues

Occasional Misinterpretation of Context

* The system may occasionally misinterpret the context of a query, leading to less relevant responses.

## 6. Glossary

Defines technical terms, acronyms, and key phrases for both technical and non-technical readers.

* **Sentiment Analysis**: The process of determining the emotional tone behind a series of words.
* **F1-Score**: A metric used to evaluate a model’s accuracy, balancing precision and recall.
* **Sarcasm Detection**: Identifying when a statement is sarcastic based on its context.

## 7. Contact and Support

Provides contact information for internal team leads and support channels, as well as instructions for submitting feedback or requesting enhancements.

# User Documentation

Capgemini – Sentiment Analysis and Response Generation Application

Developed for Capgemini

Final Capstone Project by Kennesaw State University Students

Supervised by Professor Donald Privitera

# Table of Contents

1. System Requirements
2. Accessing the Application
3. Uploading Data
4. Dashboard Overview
5. Settings Panel
6. Understanding the Visualizations
7. About Section
8. Troubleshooting & FAQ

# 1. System Requirements

- Any modern web browser (Chrome, Edge, Firefox, Safari)  
 - Stable internet connection  
 - CSV files with correctly formatted textual data

# 2. Accessing the Application

To launch CapSense AI, navigate to:  
 <https://capsenseapp-dcgpgvh6bvgufken.eastus-01.azurewebsites.net/>

# Screens screenshot of a computer AI-generated content may be incorrect.

# 3. Uploading Data

1. Click the 'Upload Data csv' button on the top left.  
 2. Choose a valid CSV file from your device.  
 3. Once uploaded, the application automatically processes and updates the visualizations.

Expected CSV Format: (Header for first column is to be “message”)

message   
 "Loved this event!"   
 "Not impressed..."

**4. Dashboard Overview**

The dashboard provides two primary visualizations:  
 - Sentiment Chart – Shows proportions of Positive, Negative, and Neutral sentiment.  
 - Platform Chart – Displays sentiment distribution by social platform.

# A screenshot of a computer AI-generated content may be incorrect.

# 8. Technical Documentation

This section provides an overview of the technical architecture, tools, and frameworks used to develop CapSense AI.

## Architecture Overview

CapSense AI is a web-based sentiment analysis application deployed on Microsoft Azure. The frontend interface allows users to upload CSV files and view sentiment analysis results through interactive charts. The backend performs sentiment analysis using machine learning models and returns structured insights to the frontend for visualization.

## Technologies Used

- \*\*Frontend:\*\* HTML, CSS, JavaScript (React.js)  
- \*\*Backend:\*\* Python (Flask), NLTK for natural language processing  
- \*\*Deployment Platform:\*\* Microsoft Azure App Service  
- \*\*Data Storage:\*\* Azure Blob Storage (for uploaded CSV files)  
- \*\*Model Serialization:\*\* joblib  
- \*\*Visualization:\*\* Plotly, Chart.js

## Sentiment Analysis

The application uses Natural Language Toolkit (NLTK) and a trained Naive Bayes classifier for sentiment analysis. Text preprocessing includes tokenization, stopword removal, and stemming. The model outputs one of three sentiment categories: Positive, Negative, or Neutral.

## Backend Workflow

1. User uploads a CSV file via the frontend.  
2. The file is sent to the backend Flask server.  
3. The server reads the data and extracts text from each row.  
4. The text is preprocessed and passed through the sentiment classifier.  
5. The results are structured and returned to the frontend for visualization.

## Frontend Functionality

- File upload interface to accept CSV files  
- Interactive charts to visualize sentiment by category and platform  
- Settings panel to modify visualization parameters  
- About section describing the project background

## Deployment Details

- Hosted on Azure using App Services  
- Continuous deployment from GitHub  
- Supports scalable updates and versioning  
- Environment variables used to secure sensitive configuration

### Backend (Python + Flask)

• Primary File: app.py – Entry point for the backend API  
• ML Models:  
 - classifier\_sentiment.py – Handles sentiment analysis logic  
 - classifier\_emotion.py – Detects emotional tone  
 - classifier\_sarcasm.py – Flags sarcastic content  
• Utility Modules:  
 - f1\_score.py – Custom metric computation (F1 score)  
 - db\_fallback.py – Provides database failover logic  
 - phi3resgen.py – Likely generates feature representations or preprocessing

### Endpoints

* /api/dashboard [GET]
  + Enacts when the dashboard of the website loads
  + Ensure that the database is successfully connected to the app.py
* /api/respond [POST]
  + Obtains a single entry of input
  + Checks the input
  + The text is then classified by the sentiment, sarcasm, and emotion
  + Then would be rated on the accuracy of the results of those models
  + Then the AI generates a response
  + The results are submitted to the database
* /api/respond\_batch [POST]
  + Obtains multiple entries of input (batch processing)
  + Checks the input
  + The text is then classified by the sentiment, sarcasm, and emotion
  + Then would be rated on the accuracy of the results of those models
  + Then the AI generates a response
  + The results are submitted to the database

### Frontend (React + Vite)

• Entry File: index.html  
• Frameworks: React, Bootstrap, Axios, Chart.js  
• Modules:  
 - React components located in src/  
 - API calls managed via Axios  
 - State likely handled via Context or useState/useReducer  
• Tooling:  
 - Bundled with Vite (vite.config.js)  
 - TypeScript support through tsconfig.json

The frontend of CapSense AI is developed using **React with Vite** for a fast, modular, and scalable user interface designed specifically for Customer Support staff analyzing feedback.

**Core Functionalities:**

* **Text Input & File Upload:** Users can enter single feedback via a <textarea> or upload CSV files for batch analysis. This is handled in the WebApp.tsx component using React state for input management.
* **API Integration:** The frontend uses **Axios** to communicate with the Flask backend via endpoints like /api/respond and /api/respond\_batch. These calls return a unified AnalysisResponse object containing sentiment, emotion, sarcasm, aspects, and AI-generated replies.
* **Result Display (SA Report):** The application renders a modular breakdown of the analysis using child components:
  + ClassificationDetector for Positive/Negative/Neutral tags
  + EmotionDetector for displaying the detected emotion
  + SarcasmDetector for flagging sarcasm
  + AspectsDetector for aspect-based sentiment summaries
  + ResponseDisplay for the Phi-3 generated empathetic reply  
    These components receive props from the main result state (results[0]) in WebApp.tsx.

**Component Logic Flow:**

* handleAnalyze() triggers API calls based on input type (text or CSV).
* The returned response populates the results array, which is then passed down to child components.
* Each child component renders its own section of the sentiment report independently based on props.

**Design & UX Considerations:**

* **Bootstrap** is used for professional and responsive layout styling.
* The UI aligns with wireframes, with the **left side** for user input and the **right side** for displaying analysis results.
* All labels and result sections are clearly marked to support non-technical users.

**Tooling and Structure:**

* Bundled with **Vite** for optimized development and build performance.
* Supports **TypeScript**, modular component design, and follows single-responsibility principles.
* State is managed via useState in WebApp.tsx; no global state library was needed due to the app’s scoped functionality.

**Deployment:**

* Deployed to **Azure Static Web Apps** and integrated with the Flask backend hosted on **Azure App Services**.
* Environment variables are used to secure endpoint URLs and configurations.
* CI/CD is set up through GitHub Actions for streamlined updates.

### Deployment & CI/CD

• CI Workflow: GitHub Actions defined in .github/workflows/  
• Deployment Platform: Microsoft Azure  
• Package Managers:  
 - Frontend: npm (package-lock.json)  
 - Backend: pip (requirements.txt)

# 9. Troubleshooting & FAQ

Q: My file doesn't upload.  
 A: Ensure it's in `.csv` format and includes valid `text` and `platform` columns.  
   
Q: I see blank charts.  
 A: Make sure your CSV contains enough records for analysis.  
   
Q: Does it support non-English data?  
 A: Not currently. Ensure the data is in English for best results.