Project Report – Conversational Insights Generator

Submitted By: Mannan Gosrani

Technology Stack: FastAPI, React (Vite), Google Gemini API, PostgreSQL, Railway, Vercel

Project Category: AI-Powered Conversation Intelligence System

1. Introduction

Organisations in the financial services and debt-collection domain handle a high volume of customer–agent phone calls every day. These calls contain important information about customer intent, sentiment, payment likelihood, and potential follow-up actions. However, manually analysing transcripts is time-consuming, inconsistent, and resource-heavy.

To address this challenge, I built a full-stack AI-powered Conversational Insights Generator capable of analysing single or multiple (CSV) transcripts using Google Gemini. The system extracts structured insights, stores them in a database, and presents them in a clean, user-friendly interface.

The project includes a deployment-ready backend, an interactive frontend, a batch-processing system, and a full explanation of the development process.

2. Objectives

The core objectives of the project were:

1. Develop an API that accepts single or bulk transcripts and generates structured insights.
2. Integrate a modern LLM (Gemini 2.0) for intent detection, sentiment classification, summary generation, and action-required prediction.
3. Store processed results in a persistent database.
4. Create a clean and intuitive user interface to display insights.
5. Support CSV upload for large-scale batch analysis.
6. Deploy the system end-to-end on cloud platforms.

These objectives directly align with the assignment requirements and demonstrate a complete full-stack implementation.

3. System Architecture

The architecture is divided into four primary layers:

Frontend

- React with Vite
- Provides user interface for both single analysis and batch uploads
- Displays insights through tables, modals, and summary views
- Deployed on Vercel

Backend

- FastAPI framework
- Receives single and batch requests
- Sends transcripts to Gemini
- Validates and structures responses
- Saves each analysis into PostgreSQL
- Deployed on Railway

Database

- PostgreSQL hosted on Railway
- Stores transcript, intent, sentiment, summary, action-required status, and timestamps

AI Layer

- Google Gemini Flash 2.0
- Generates structured JSON-style responses for each transcript
- Prompt-engineered for consistency

This modular architecture ensures scalability, clarity, and maintainability.

4. Technology Stack Overview

Frontend:
React, Vite, CSS, Fetch API

Backend:
FastAPI, Pydantic, python-multipart, asyncpg

AI Model:
Google Gemini Flash 2.0

Database:
PostgreSQL

Deployment:
Vercel (frontend), Railway (backend and database)

5. Development Process

The development went through several key phases.

Phase 1: Backend Development

The backend development process began by setting up the FastAPI server and connecting it to the Gemini model for transcript analysis. Structured output formats were defined for intent, sentiment, action requirement, and summary.

A database connection layer was created using PostgreSQL, and tables were initialised to store results. Endpoints were added for single transcript processing and later expanded to support batch processing.

Key achievements in this phase:

- Stable Gemini integration
- Output sanitation and validation
- Database persistence
- Well-structured API routes

Phase 2: Frontend Development

The UI was designed to be simple, clean, and easy to navigate. Components were developed for:

- Single transcript input
- Batch CSV upload
- Result cards
- Table display of batch outputs
- Modal views for detailed summaries

The interface communicates with the backend using Fetch API. User feedback elements (loading states, error handling) were also implemented.

Phase 3: Deployment and Infrastructure Setup

This phase involved several challenges due to frontend–backend integration and hosting.

Issues Encountered and Resolved

1. **CORS Errors:**
   The deployed backend initially rejected calls from the Vercel frontend. This was resolved by correctly configuring CORS middleware on the backend.
2. **Incorrect API Endpoint Usage:**
   The frontend initially targeted localhost even after deployment, leading to failed requests. Updating the API base URL resolved the issue.
3. **Railway Database Connection Issues:**
   Railway provides both public and internal database URLs. The internal connection string was required for the server environment, and switching to it resolved failures.
4. **Missing python-multipart Dependency:**
   The batch upload endpoint required a multipart parser. Installing the missing dependency resolved repeated runtime errors.
5. **CSV Parsing Issues:**
   Uploaded CSVs had inconsistent headers (for example, “Transcript” instead of “transcript”).
   Normalisation logic was added so the application accepts multiple header variations.
6. **Vercel Build Failure:**
   The frontend failed to build due to an incorrect import for the CSV parsing library. Correcting the import fixed the issue.

7. **UI Layout Problems:**
   The batch results table initially appeared cramped.
   Layout fixes included:
   - Expanding table width
   - Removing unnecessary sidebars
   - Improving readability and spacing
   - Adding a modal for detailed summaries

Phase 4: Batch Processing Feature

Once single transcript analysis was stable, a CSV-based batch processing feature was added. This feature supports multiple transcripts, shows real-time progress counts, displays results in tabular format, and allows viewing extended summaries per row.

It was the most complex and time-consuming feature and required enhancements on both frontend and backend.

6. Features Implemented

Single Transcript Analysis

- User enters a call transcript
- Backend analyses and returns structured insights
- Results saved automatically to database
- Displayed in a readable card format

Batch CSV Upload

- Users can upload CSV files with multiple transcripts
- Each row is processed individually
- Displays a full results table with several columns
- A detailed modal allows viewing the complete summary
- Errors are handled per-row to avoid batch failures

AI-Powered Insights

Model extracts:

- Intent of customer
- Sentiment
- Whether action is required
- Summary of the conversation

Database Storage

All single analyses are inserted into PostgreSQL, enabling future retrieval and audit.

Frontend UI

- Clean typography and spacing

- Full-width tables
- Interactive “View Details” pop-ups
- Intuitive batch results page

Deployment

- Fully deployed backend
- Fully deployed frontend
- Persistent cloud-hosted database
- Environment variables securely configured

7. Alignment With Submission Requirements

The project fully satisfies all expected criteria:

- GitHub link included
- Project report provided
- Demo video can be recorded easily from the deployed application
- Backend and frontend both implemented
- AI model integrated for advanced analysis
- CSV batch upload functionality included
- Database integration complete
- Deployed end-to-end in production environments

The system meets and exceeds the functional requirements outlined in the assignment.

8. Conclusion

This project demonstrates the end-to-end development of a complete AI-driven application. It showcases skills in backend development, frontend engineering, cloud deployment, database integration, API design, prompt engineering, and error-handling.

The Conversational Insights Generator is a scalable and practical solution for analysing large volumes of customer–agent call transcripts in the financial domain. The modular design, clean user interface, and robust backend make it suitable for real-world use cases.