System Design and Architecture for VentiAI

Ranjan Khadka

University of Minnesota

Math, Science and Technology Department

Software Construction : SE2300

Silvia Preston

Feb 03, 2026

Figures and Diagrams

The Problem Statement

The VentiAI system is an AI-powered emotional support application designed to provide users with a personal, judgment-free space to vent their frustations. The system must allow users to input "vents" via text or speech, which the application then processes to identify emotional sentiment. Once analyzed, the system generates feedback and offers gentle coping suggestions, like breathing exercises.

To ensure the system is robust and ethical, the following constraints are implemented:

- Error Handling & Validation: The system must validate that inputs are not empty and handle potential failures in AI processing or speech-to-text conversion.
- User Interface: A calming, simple graphical user interface (GUI) developed in React.js or vanilla HTML/CSS will guide the user through the process.
- Data Persistence: To maintain privacy, all vent data and sentiment results must be stored locally on the user's device rather than a central server.
- Safety Disclaimer: The system must explicitly state it is not a replacement for professional medical services.

Design of the System

Using the modular design approach from Chapter 5 , VentiAI is decomposed into four primary components to ensure low coupling and high cohesion (Pressman & Maxim, 2020, Chapter 5).

- Classes and Objects: The system utilizes a UserInterface class to handle display, a SentimentAnalyzer to process emotions, a ResponseGenerator for feedback, and a StorageManager for data handling.
- Attributes and Methods: Each class encapsulates specific data (e.g., the emotionalTone string) and behaviors (e.g., analyzeInput()) to maintain a clean architecture.

Class Structure Summary

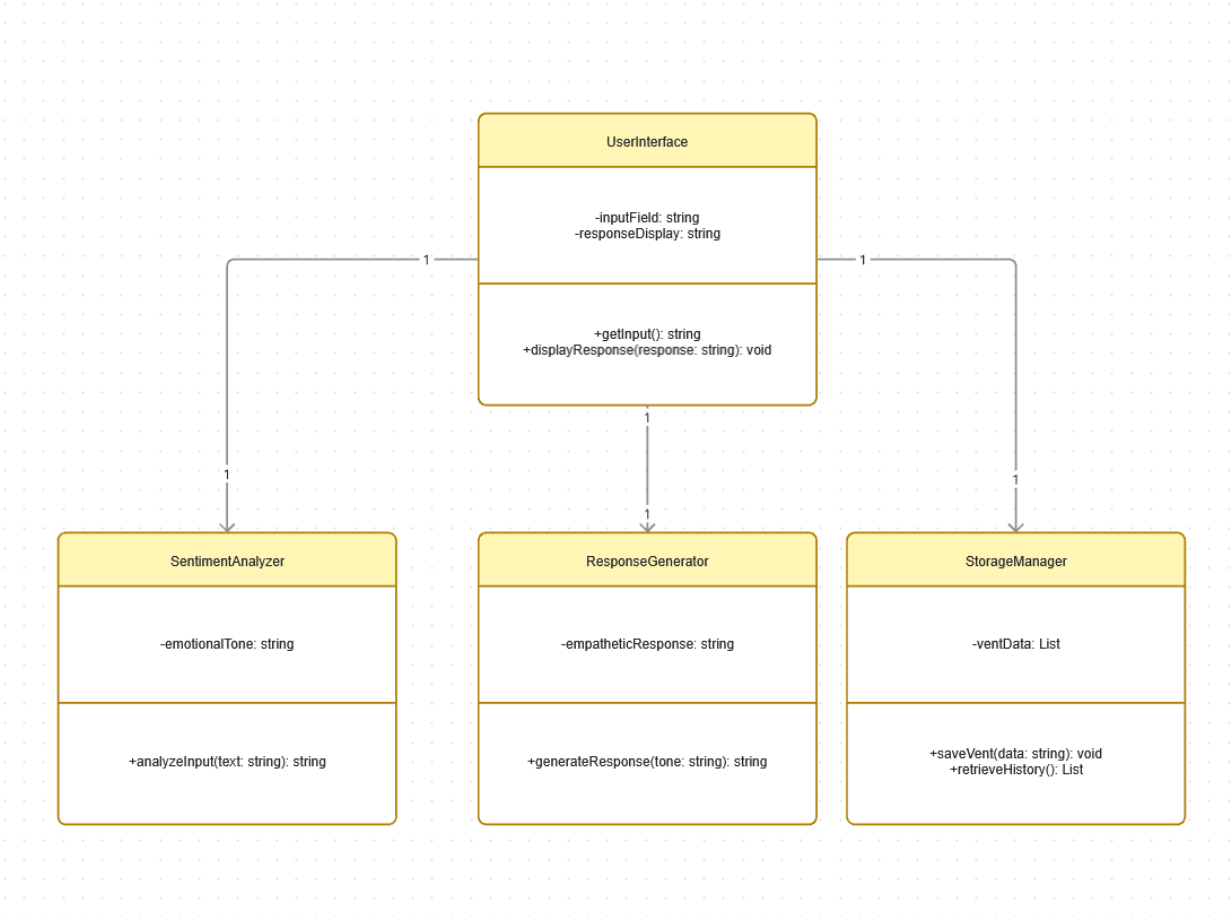

Figure 1: UML case diagram

The UserInterface class acts as the hub, managing the primary input/output via attributes; inputField and responseDisplay. It maintains a one-to-one relationship with three components,

effectively delegating core tasks: it sends user text to the SentimentAnalyzer to determine emotional context, requests a tailored reply from the ResponseGenerator, and ensures the interaction is logged through the StorageManager.

The subordinate classes provide the functional "muscles" for the system through encapsulated methods. The SentimentAnalyzer processes text to return a string representing the emotionalTone, which is then passed as an argument to the ResponseGenerator's generateResponse(tone: string) method. At the same time, the StorageManager handles data persistence, using saveVent and retrieveHistory to manage a list of user entries. This modular approach follows a clean separation of concerns, where the UI focuses on interaction while the backend classes handle analysis, logic, and data storage.

Pseudocode

```
CLASS SentimentAnalyzer {
  CONSTRUCTOR() {
    this.emotionalTone = ""
  }

  METHOD analyzeInput(inputText) {
    IF inputText is EMPTY OR INVALID THEN
      RETURN "Error: Input required"
```

```
        this.emotionalTone = PERFORM_SENTIMENT_ANALYSIS(inputText)
        RETURN this.emotionalTone
    }
}

CLASS ResponseGenerator {
    METHOD generateResponse(tone) {
        LET response = ""

        IF tone == "sadness" THEN
            response = "I hear how much pain you're in. It's okay to feel this way."
        ELSE IF tone == "stress" THEN
            response = "That sounds overwhelming. Remember to breathe."
        ELSE
            response = "Thank you for sharing your thoughts with me."

        RETURN response
    }
```

```
}


CLASS StorageManager {
   METHOD saveToLocal(ventText, tone, aiResponse) {

      LET record = { timestamp: NOW(), text: ventText, emotion: tone, feedback: aiResponse }
      SAVE record TO LOCAL_BROWSER_STORAGE
   }
}
```

Flowchart

The following flowchart represents the operational logic of a single user interaction, from input to the final persistence of data.

- Start: User launches the app and sees the disclaimer.
- Input: User enters text or speaks (P1/P2).
- Process: The system analyzes the text for emotional tone (P3).
- Decision/Action: The AI selects the appropriate empathetic response based on the detected tone (P4).
- Output: The feedback is displayed on the screen.
- Data Persistence: The interaction is saved to local storage for the history view.

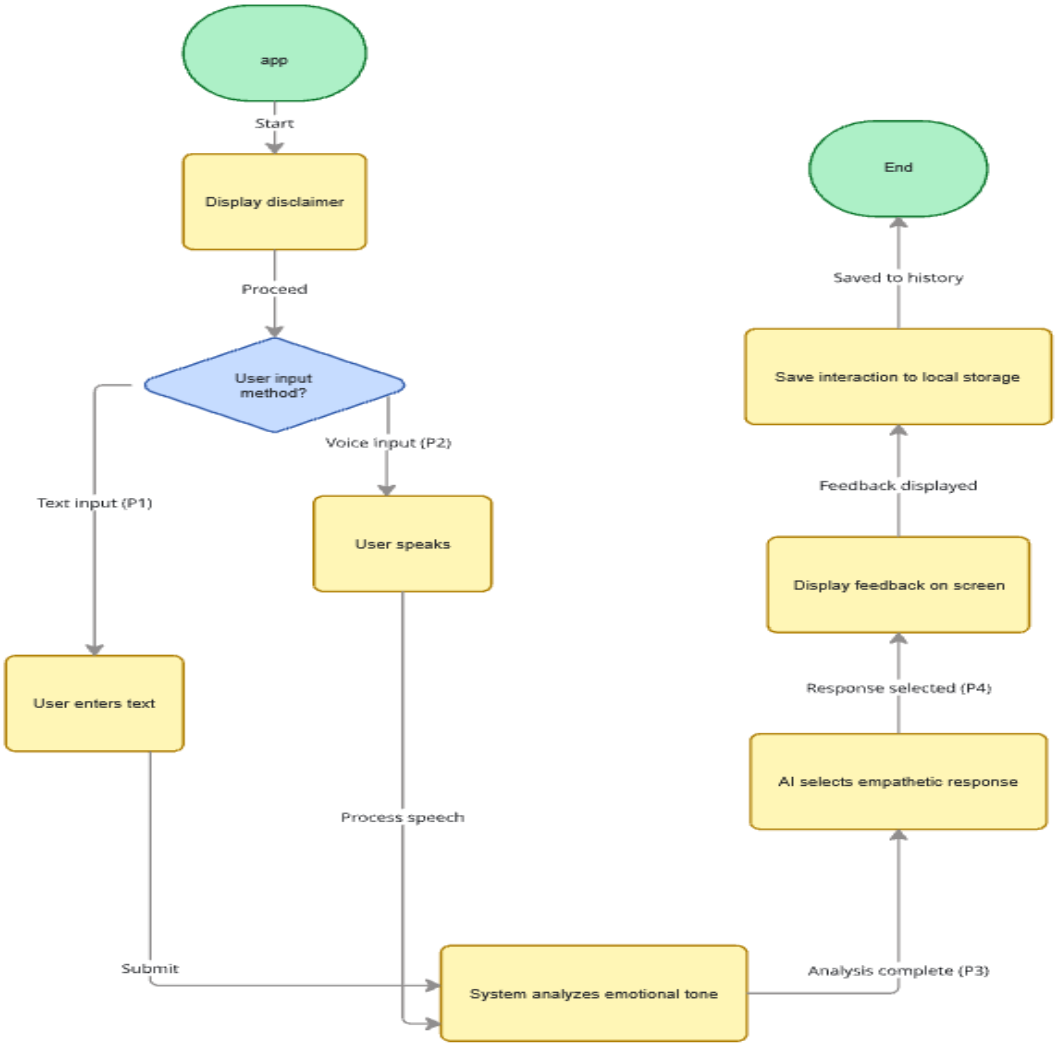

Figure 2: Flow chart

References

Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.