The User's Perspective (Frontend) and the System's Perspective (Backend).

Visual Overview

This diagram shows the high-level interaction between the components:

```
+-----------+     (Types message)     +-------------------+     (Sends API Request)
+--------------------------+
|           | ----------------------> |                   | --------------------------> |                       |
|   User    |                         |  Frontend Browser |                             |   Backend Server (Python)|
|           | <---------------------- |  (HTML, CSS, JS)  | <-------------------------- |      (FastAPI)          |
+-----------+     (Displays reply)    +-------------------+     (Receives Response)
+--------------------------+
                                                                                        |
                                                                                        | (Analyzes text & gets resources)
                                                                                        V
                                                                              +--------------------+
                                                                              | AI Model &         |
                                                                              | resources.json     |
                                                                              +--------------------+
```

Step-by-Step Workflow

Here is the detailed sequence of events when a user interacts with AuraMind.

Part A: The User's Perspective (Frontend Workflow)

This all happens in the user's web browser.

1. Initial View: The user navigates to the application's URL. The `index.html` page loads, displaying the chat interface with the AuraMind header and the initial welcome message, including the critical safety disclaimer.
2. User Action: The user types a message (e.g., *"I've been feeling so overwhelmed lately"*) into the input field and clicks the "Send" button or presses `Enter`.
3. Instant UI Update: The `script.js` file immediately captures the user's text. It creates a new blue message bubble (`user-message`) and appends it to the chat window. This provides instant feedback that the message has been sent.
4. Sending to Backend: In the background, the JavaScript code initiates an asynchronous `fetch` request to the `/chat` endpoint on the backend server. The user's message is sent as a JSON payload.
5. Receiving and Displaying Response: The frontend waits for the backend to send a response. Once the JSON response is received, the JavaScript code parses it, creates a new grey message bubble (`bot-message`), and displays the AI's reply and any associated resources in the chat window. The window automatically scrolls down to show the latest message.

Part B: The System's Perspective (Backend Workflow)

This is the core logic happening on the server (`main.py`).

1. Request Received: The FastAPI server receives the `POST` request at the `/chat` endpoint. It validates the incoming data to ensure it contains the user's text.
2. Safety First: The Crisis Check: This is the most important step. The server takes the user's text and checks if it contains any of the high-risk keywords defined in the `CRISIS_KEYWORDS` set (e.g., "suicide", "kill myself").
   - If a crisis keyword is found: The workflow is immediately redirected. The server retrieves the predefined crisis message and the list of helpline resources (Vandrevala Foundation, Nagpur Suicide Prevention Helpline, etc.) from `tier3_crisis` in the `resources.json` file. It skips all AI analysis and sends this critical information back to the frontend as the response.
   - If no crisis keywords are found: The workflow proceeds to the next step.
3. AI-Powered Emotion Analysis: The user's text is passed to the pre-loaded Hugging Face `emotion-classifier` pipeline. The AI model analyzes the text and outputs the most dominant emotion (e.g., `sadness`, `fear`, `joy`) along with a confidence score (e.g., `0.88`).
4. Tiered Logic Decision: The server uses the output from the AI to determine the appropriate response tier:
   - Tier 2 (Medium Risk): If the dominant emotion is strongly negative (like 'sadness' or 'fear') and the confidence score is high (e.g., greater than 0.6), the system decides that a Tier 2 response is needed.
   - Tier 1 (Low Risk): For all other cases (e.g., neutral, positive, or mildly negative emotions), the system defaults to a Tier 1 response.
5. Resource Selection: Based on the decided tier (1 or 2), the server accesses the `resources.json` file. It randomly selects one of the pre-written responses from the corresponding list (`tier1` or `tier2`) to ensure the conversation feels more natural and less repetitive.
6. Response Sent: The server packages the selected reply into a JSON object and sends it back to the user's browser, completing the API request cycle. The frontend JavaScript then takes over to display this response, as described in Part A.