Proposed IoT Project: Emotion-Aware Monitoring & Access System

1. Overview of the revised project idea aligned with university criteria

We propose an **Emotion-Aware IoT Monitoring & Access System** that builds on the original chatbot-sentiment idea while meeting all course requirements. The ESP32 reads *three* inexpensive sensors (for example, an LDR light sensor, a DHT11 temperature/humidity sensor, and an MQ-2 gas sensor) as required. Sensor data are shown in real time on a 16×2 LCD display. An onboard servo motor serves as the actuator (e.g. turning a smart-lock or a character figurine), and LEDs indicate detected emotions. All sensor readings are published via MQTT (using HiveMQ) to the cloud. A Flutter mobile app (with ≥5 screens: login/signup, dashboard, control, alerts, chatbot, etc.) subscribes to the MQTT topics and displays live data. The app integrates Supabase for user **Authentication** and database storage. Crucially, the app includes a chatbot interface: when a user types a message, the system performs sentiment analysis (detecting joy, sadness, etc.) and sends a matching control command via MQTT to the ESP32. For example, detecting "joy" in the text triggers the servo to rotate the Joy character (and lights an LED). In summary, this design preserves the original emotion-driven chatbot concept and fully satisfies the project requirements: 3 sensors, servo actuator, LCD display, HiveMQ communication, Supabase backend, and a multi-screen Flutter app.

2. Precedence-centered pipeline architecture (step-by-step system flow)

1. **Sensor Data Acquisition:** The ESP32 reads values from the three sensors (e.g. light, temperature, gas). It continuously updates these readings on the LCD in real time.
2. **Data Publishing:** After each read, the ESP32 publishes the sensor readings to the HiveMQ MQTT broker under a topic like esp32/sensors/data.
3. **Cloud/Database Logging:** Optionally (bonus), a backend or Node-RED flow listens to the MQTT stream and pushes the values via Supabase's REST API into the database (tables like sensor_readings). This archives the data for history and analytics.
4. **User Authentication & Subscription:** In parallel, users log into the Flutter app using Supabase Auth. After login, the app subscribes to the MQTT topics (esp32/sensors/data) via HiveMQ to receive live updates. The dashboard screen plots the incoming sensor data in real time.
5. **Chatbot Sentiment Analysis:** The user types a message into the app's chatbot screen. An onboard sentiment module (or a LangChain agent) analyzes the text for

emotions. If "joy" is detected, for instance, the app publishes a control message (e.g. esp32/actuator/control with payload "joy") over MQTT.

6. **Actuator Control:** The ESP32, subscribed to esp32/actuator/control, receives the control message. Based on the command, it rotates the servo motor and/or toggles an LED. (For example, "joy" causes the servo to turn the Joy figurine, while other emotions could light LEDs or remain idle.)
7. **Feedback Loop:** The action taken (servo position or LED color) can be published back over MQTT as an acknowledgement, and the app displays any alerts or status updates (e.g. an "Access Granted" message) on a control screen.
8. **Optional Flask API (Bonus):** A Flask middleware runs separately, perhaps fetching external data (e.g. weather or pollution via a public API) or providing additional APIs. The LangChain chatbot calls this Flask API to answer user queries like "What's the temperature now?" or to enrich the data stream.
9. **Cloud Deployment (Bonus):** Each component (mock ESP32, Flask API, Supabase API, Flutter web UI) can be containerized. These Docker containers can be deployed on Azure (App Service or AKS), and optionally Azure IoT Hub can be integrated for a production-level MQTT broker.

3. Role division for 5 team members based on responsibilities

- **Member 1 – Hardware & Electronics:** Handles the physical setup and wiring. Tasks include assembling the sensors (LDR, DHT11, MQ-2), connecting the LCD and servo to the ESP32, and building the maquette/model to showcase the setup. This member ensures proper circuit design and tests the sensor readings on Wokwi.
- **Member 2 – ESP32 Firmware Developer:** Writes the ESP32 Arduino/C++ code. Responsibilities include interfacing with sensors (analog/digital inputs), reading data, updating the LCD, controlling the servo/LED based on incoming commands, and implementing MQTT publish/subscribe logic. This person tests the ESP32 logic end-to-end.
- **Member 3 – Mobile App Developer (Flutter):** Designs and implements the Flutter app UI with ≥5 screens (login/signup, dashboard, control, alerts, chatbot). This role integrates HiveMQ for subscribing to sensor topics and publishing control messages. They also set up Supabase Auth for user login and store/fetch data via Supabase REST. They implement the chatbot input interface and display actuator status.

- **Member 4 – Cloud/Backend Developer:** Sets up the Supabase project (creating tables for users, sensor_readings, etc.) and implements any needed REST endpoints or Node-RED flows. This member may also develop the Flask middleware (bonus) to fetch real-world data or provide APIs. They ensure secure access (Supabase Auth) and that data flows from the cloud to the app as specified.
- **Member 5 – AI/Chatbot & DevOps:** Implements the sentiment analysis/chatbot logic. This includes configuring a LangChain agent (or another model) to process user queries and detect sentiment from text. They tie the chatbot's output to MQTT commands (e.g., send "joy" if Joy is detected). Additionally, this member handles Dockerization of components and explores Azure deployment (deploying containers, possibly using Azure IoT Hub) as time permits. They also compile final documentation and demo materials.

4. Technologies and components list (with affordable sensor suggestions)

- **Sensors (3):** Light sensor (LDR, ~$0.5), Temperature/Humidity sensor (DHT11, $2–3), Gas/MQ sensor (MQ-2, $3–5). Alternatives: PIR motion sensor ($2), IR obstacle sensor ($1). These are low-cost and easy to interface with ESP32.
- **Actuator:** Standard 9g servo motor (~$3–5) to turn the Joy figurine or control a lock. Also a few LEDs (single-color or RGB, ~$0.5 each) to indicate different emotions.
- **Display:** 16×2 character LCD with I^2C interface (~$4–5) for showing live sensor readings.
- **Microcontroller:** ESP32 development board (~$8–10) for Wi-Fi and MQTT capability.
- **Components:** Jumper wires, breadboard, resistors, power supply, and wood/foam board to build a small maquette model (as required).
- **Communication:** Wi-Fi network; HiveMQ Cloud broker (free tier) for MQTT pub/sub.
- **Cloud/Backend:** Supabase (PostgreSQL backend + Auth) – free tier covers our needs. Optional Node-RED (open-source) to route MQTT to REST.
- **App/Software:** Flutter (Dart) for cross-platform UI. Flutter packages: *mqtt_client* for MQTT, *supabase_flutter* for Auth/DB. Python 3 with Flask for the API middleware (bonus) and LangChain (bonus) for the chatbot logic.
- **Other Tools:** VS Code or Arduino IDE for coding, Wokwi or Tinkercad for circuit simulation, Git/GitHub for version control. Docker for containerizing components (bonus).

5. Recommendations for cloud integration (Azure vs HiveMQ trade-offs)

- **HiveMQ Cloud:** A fully managed MQTT broker with a developer tier. It supports the full MQTT standard (3.1.1 and 5)hivemq.com, making pub/sub straightforward. The free plan easily handles small-scale messaging. HiveMQ is simple to set up (just configure topics in the Flutter app and ESP32) and has open MQTT client compatibility. This is ideal for quick prototyping and fulfills the course requirement for using HiveMQ.
- **Azure IoT Hub:** A scalable enterprise IoT service that also supports MQTT. It integrates well with other Azure services (App Services, Functions, etc.), which is useful if we deploy the system on Azure. However, **Azure's MQTT support is limited**: it restricts topic namespaces and does *not* fully implement MQTT 5hivemq.comhivemq.com. In practice, using IoT Hub requires device provisioning and specific MQTT topic formats. Azure offers a free tier (e.g. 8,000 messages/day), but costs can rise for higher usage. For our 9-day project, HiveMQ's free MQTT broker is easier and faster. We recommend starting with HiveMQ for development and only integrating Azure if aiming for the deployment bonus (e.g. hosting containers on Azure App Service/IoT Hub)hivemq.com.

6. Optional bonus extensions (Flask API, LangChain)

- **Flask API Middleware:** Build a small Flask service that *simulates or fetches real-world data*. For example, it could periodically pull weather or pollution index data and expose them via REST endpoints. This service can be called by the chatbot and/or ESP32 (via Wi-Fi) to augment sensor data. It also exemplifies the bonus requirement of a "real-life" API layer.
- **LangChain Chatbot Agent:** Implement a LangChain-based chatbot that can answer user questions about the system status. For instance, queries like "What's the temperature now?" or "Is the room well-lit?" would be answered by calling the Flask API or reading current values. The LangChain agent can also incorporate the sentiment analysis logic: user text is processed to detect "joy" or other emotions, and then an MQTT control message is emitted. This preserves the original sentiment-control concept and aligns with the LangChain bonus feature.
- **Dockerization:** Containerize each component (ESP32 simulator server, Flask API, web app, Supabase emulator) as shown in the requirements. This allows easy local deployment and testing. If time permits, set up a docker-compose or Kubernetes workflow to run all parts together.

- **Azure Deployment:** For extra credit, deploy the Docker containers on Azure App Service or Azure Kubernetes Service. Optionally connect the ESP32 to Azure IoT Hub (instead of HiveMQ) to demonstrate enterprise-grade integration. Deploy the Flutter app as a web app on Azure Static Web Apps or serve it from App Service.

7. Deployment and documentation milestones (to finish within 9 days)

- **Days 1–2 (Planning & Setup):** Finalize architecture and component selection. Assign roles/tasks. Order or gather all sensors and hardware. Set up development environments (ESP32 toolchain, Flutter project skeleton, Supabase project, and HiveMQ broker). Sketch initial app screens and chatbot flow.
- **Days 3–4 (Hardware & Firmware):** Assemble the circuit (ESP32 + 3 sensors + LCD + servo). Write and test ESP32 code to read sensors and display on the LCD. Verify sensor readings on Wokwi or hardware. Set up HiveMQ topics and implement basic MQTT publish from ESP32. Simultaneously, begin Flutter UI development (login and dashboard screens) and Supabase schema (tables for sensors and users).
- **Days 5–6 (Integration – Data & Control):** Implement ESP32 MQTT publish and subscribe logic so that sensor data reaches the app, and control commands from the app control the servo. In Flutter, finalize login/signup (Supabase Auth) and subscribe to esp32/sensors/data. Display incoming data on the dashboard. Create a control screen to send simple MQTT messages (e.g. open/close door). Connect an LED to the ESP32 and test a sample MQTT command (e.g. toggle LED).
- **Days 7–8 (Chatbot & Cloud):** Integrate the sentiment-chatbot feature: implement a text input in the app, analyze the input (using a simple NLP library or LangChain) for emotion, and upon "joy" emit the MQTT command that rotates the servo. Test end-to-end so that user text triggers the physical servo movement. Develop bonus features: set up a Flask API that returns simulated data, and a LangChain agent that queries it for Q&A. Dockerize components locally and (optionally) deploy to Azure to verify.
- **Day 9 (Finalization & Deliverables):** Complete testing of all features. Prepare the 5–6 minute demo video (Arabic narration as required) showing sensors, app screens, and the emotional response (servo/LED) in action. Finalize the code with comments. Create the Wokwi circuit diagram. Assemble the presentation slides (≥15 slides) covering system overview, tech stack, architecture, demo results, and future work. Push all code and media to GitHub with a README. Ensure the team worksheet (member contributions) is ready.

Each day's progress should be documented. By day 9, all core functionality (sensor monitoring, MQTT flow, Supabase integration, chatbot control) must be working, and all deliverables (code, video, slides) completed.

Sources: University IoT project requirements document; HiveMQ comparison of Azure IoT Hub vs HiveMQ (MQTT support)hivemq.comhivemq.comhivemq.com.