Absolutely. Here’s a concrete, “do-this-next” implementation plan to bake your empathy/safety features into the Lighter Tomorrow coach—starting with prompt tweaks (fast), then adding retrieval, signals (sentiment/flags), and finally optional fine-tuning. I’ve included drop-in prompts, JSON schemas, and code scaffolds you can paste into your stack.

**0) Pick your path (you can mix & match)**

* **Hosted model (e.g., OpenAI/GPT)**: Fastest to ship. Use system prompts + tool calling + RAG. Add fine-tuning later if needed.
* **Open-source (e.g., Llama/Mistral)**: You’ll implement the same orchestration, but you *can* fine-tune/LoRA sooner. Great if you need offline or stricter PII control.

**1) System prompt (drop-in)**

Use this as your **system** message for any “coach” chat. It encodes empathy, boundaries, and safety:

You are “Lighter Tomorrow,” a supportive weight-loss coach for users who may struggle with emotional or addictive eating.  
**Style**: Start by validating feelings in 1–2 short lines (“That sounds rough…”) before any tips. Be concise, warm, non-judgmental, trauma-informed. Offer 1–3 gentle, evidence-aligned options—never overwhelm.  
**Scope**: Coaching, not therapy or medical advice. If you detect medical risk or eating-disorder red flags (e.g., purging, restriction, self-harm, severe distress), encourage professional help and share pre-approved resources. Do NOT diagnose or prescribe.  
**Personalization**: Use known user context (goals, swaps, triggers, wins) positively, never shame.  
**Weight-loss framing**: Tie suggestions back to behaviors (stress eating, late-night snacking, plateaus) and small next steps (environment, swaps, urge surfing, postponement, planning).  
**Safety & data**: If user mentions crisis, provide the escalation tool. Never retain or reveal sensitive content unless user has consented; summarize gently when asked.  
**When asked for facts**: Prefer the app’s knowledge base (RAG). If uncertain, say so briefly.

**2) Message template (server-side)**

Wrap each user message with your app context so the model always “feels” the app:

{

"user\_message": "<raw user text>",

"app\_context": {

"mode": "vent| craving| swap| reflect| history\_view",

"user\_profile": {

"first\_name": "…",

"goals": ["…"],

"sensitivities": ["wheat","chocolate", "…"],

"wins": ["lost 10 lb", "3-day streak"],

"current\_streak\_days": 3

},

"recent\_events": [

{"type":"captured\_moment","text":"…","ts":"…"},

{"type":"swap\_done","swap":"tea for soda","ts":"…"}

],

"consents": {"personalization": true, "store\_vents": false}

}

}

**3) Tool (function) calling schema**

Expose a few **safe tools** the model can call. Your backend executes them and returns results to the model. This keeps responses grounded and builds your product features.

**a) RAG lookup** (facts over hallucinations)

name: "retrieve\_resources"

arguments: { "topic": "emotional\_eating | urges | plateaus | sleep | fiber | hydration", "maxItems": 3 }

returns: [{title, summary, url, safety\_tag}]

**b) Suggest next steps** (keep it tiny)

name: "suggest\_micro\_steps"

arguments: { "context": string, "mood": "low|neutral|high", "time": "1m|5m|15m" }

returns: [{step, rationale, duration\_seconds}]

**c) Coping exercises** (grounding/urge surfing)

name: "guide\_exercise"

arguments: { "exercise": "box\_breathing|54321\_senses|urge\_surfing", "length":"short|standard" }

returns: {script: string}

**d) Log a moment** (with consent)

name: "log\_captured\_moment"

arguments: { "text": string, "tags": ["trigger","emotion","location"], "store": boolean }

returns: {id, stored:boolean}

**e) Escalation resource**

name: "offer\_support\_resources"

arguments: { "reason": "self\_harm|ed\_flag|crisis", "region":"US" }

returns: [{label, description, contact}]

**4) Sentiment + red-flag prepass (cheap, high-impact)**

Before you hit the LLM, run a lightweight classifier to detect **mood** and **flags**. Use the results to condition the system prompt or temperature.

**Signals to detect**

* Sentiment: low / neutral / high
* Urge intensity: 0–3
* Red flags: self-harm, ED risk (purging, laxatives, severe restriction), substance use, medical emergency

**Routing**

* If **red flag**, append a one-line instruction to the system msg:  
  “If red\_flag is present, respond in ≤3 sentences and call offer\_support\_resources.”
* If **low mood**, set temperature lower and cap suggestions to 1–2.

(You can ship v1 using keyword lists + heuristics; upgrade later to a tiny fine-tuned classifier.)

**5) Retrieval-Augmented Generation (RAG)**

Create a small, curated, citation-light knowledge base (KB) of coach-approved summaries (e.g., urge surfing, stimulus control, fiber & satiety, sleep & hunger signals). Store as short notes (300–600 tokens) with **clear safety tags**(“educational, not medical”).

* **Stack suggestion** (easy with your ecosystem): *Supabase Postgres + pgvector*  
  **Table**: kb\_entries(id, title, summary, tags[], safety\_tag, url, embedding vector)
* **At runtime**: embed the user query + context → top-k kb entries → put as RAG\_CONTEXT in the system prompt *only*when the user asked for information or when the coach wants to support a suggestion with a short fact.

**6) Response “shape” contract (keeps UI predictable)**

Ask the model to always return **structured JSON** you can render consistently:

{

"style":"empathetic\_v1",

"opening\_validation": "Short validation line…",

"core\_reply": "1–3 tight sentences…",

"micro\_steps": [{"label":"Box breathing (1 min)","script":"…"}],

"swap\_suggestions": [{"name":"Protein-yogurt cup"}, {"name":"Herbal tea w/ lemon"}],

"offer\_to\_log": true,

"kb\_refs": [{"title":"Urge Surfing—3 steps"}],

"safety": {"flag":"none|ed\_flag|self\_harm","resources":[]}

}

If your SDK prefers text + tool calls, you can still produce this JSON as the final text response body. Your iOS UI can then decide which widgets to show (e.g., “Start 1-min Box Breathing”).

**7) Guardrails & boundaries (what it *shouldn’t* do)**

* **Never**: calories/meal plans, diagnosis, med dosing, “cure” claims.
* **Do**: “education + options + consent + tiny next step.”
* **Red-flag wording** (ready-to-use):  
  “I’m really glad you told me. I can’t provide crisis help, but you’re not alone. If you can, consider reaching out to a professional. Here are some options: … (via offer\_support\_resources).”

**8) Fine-tuning (optional but powerful)**

When you’re ready:

**Dataset format (jsonl)**

{"messages":[

{"role":"system","content":"<same system prompt>"},

{"role":"user","content":"I binged last night and hate myself."},

{"role":"assistant","content":"I’m sorry—that’s a heavy feeling... (empathetic, short)... Here are 2 tiny options… Would you like to log a moment?"}

], "rating":"good", "tags":["empathy","shame\_reframe","tiny\_steps"]}

* **Size**: 1k–10k multi-turn examples; include “bad vs good” pairs to shape tone.
* **PEFT/LoRA**: tune only attention layers; keep base model frozen for cost/safety.
* **RLHF**: use thumbs-up/down + “why” to build a preference dataset later.

**9) Feedback loop**

* In the chat footer: 👍 / 👎 / “This missed the mark because…”.
* Log (prompt hash, context hash, outputs, user rating, reason).
* Nightly cron:
  + Reweight simple prompt heuristics (e.g., fewer suggestions if “overwhelming” feedback rises).
  + Add high-value conversations (after anonymization + consent) to the fine-tune pool.

**10) Privacy & consent (practical defaults)**

* Default **off** for storing raw vents. On first attempt to “save,” show a one-screen consent:  
  “Store my notes to show patterns later (you can delete anytime).”
* Store only **summaries** if user declines raw storage (e.g., “stress after work; chose walk”).
* Make **Delete All** accessible in Settings → Privacy.

**11) Testing & monitoring**

* **Unit prompts**: a suite of 50–100 canonical scenarios (late-night craving, shame after binge, ED warning, celebratory win).
* **Scorecards** (automated & human): Empathy first ✓, ≤3 suggestions ✓, no medical advice ✓, correct escalation ✓.
* **Live monitors**: error rate, tool-call failure, time-to-first-token, % red-flag sessions, user CSAT.

**12) Minimal code scaffolds**

**TypeScript (server) — OpenAI-style tool flow**

import OpenAI from "openai";

const openai = new OpenAI({ apiKey: process.env.OPENAI\_API\_KEY });

const tools = [

{ type: "function", function: { name:"retrieve\_resources", parameters:{ type:"object", properties:{ topic:{type:"string"}, maxItems:{type:"number"} }, required:["topic"] } } },

{ type: "function", function: { name:"guide\_exercise", parameters:{ type:"object", properties:{ exercise:{type:"string"}, length:{type:"string"} }, required:["exercise"] } } },

// ...other tools

];

export async function coachReply(payload) {

const { user\_message, app\_context } = payload;

const system = buildSystemPrompt(app\_context); // includes flags/consent

const messages = [

{ role: "system", content: system },

{ role: "user", content: JSON.stringify({ user\_message, app\_context }) }

];

const res = await openai.chat.completions.create({

model: "gpt-4o-mini", // or your chosen model

messages,

tools,

tool\_choice: "auto",

temperature: chooseTemp(app\_context) // lower if low mood

});

const out = await resolveToolCalls(res); // run retrieve\_resources/guide\_exercise/etc

return normalizeToCoachJSON(out); // shape it like the JSON contract above

}

**Supabase pgvector table**

create table kb\_entries (

id uuid primary key default gen\_random\_uuid(),

title text,

summary text,

tags text[],

safety\_tag text,

url text,

embedding vector(1536)

);

**Swift (iOS) — simple API call**

struct CoachResponse: Codable {

let opening\_validation: String

let core\_reply: String

let micro\_steps: [MicroStep]?

let swap\_suggestions: [Swap]?

let safety: SafetyBlock?

}

func fetchCoachReply(userText: String, context: AppContext) async throws -> CoachResponse {

let payload = ["user\_message": userText, "app\_context": context]

var req = URLRequest(url: URL(string: "<YOUR>/coachReply")!)

req.httpMethod = "POST"

req.addValue("application/json", forHTTPHeaderField: "Content-Type")

req.httpBody = try JSONEncoder().encode(payload)

let (data, \_) = try await URLSession.shared.data(for: req)

return try JSONDecoder().decode(CoachResponse.self, from: data)

}

**13) Rollout plan (4 sprints)**

**Sprint 1 (Prompt + Tools)**

* Ship system prompt, JSON contract, guide\_exercise, log\_captured\_moment.
* Heuristic red-flag detection.
* UI shows validation → 1–2 options → “Log this?” CTA.

**Sprint 2 (RAG + Sentiment)**

* Stand up pgvector KB and retrieve\_resources.
* Add lightweight sentiment/urge scoring to modulate tone/length.

**Sprint 3 (Feedback + Patterns)**

* Thumbs up/down + reasons.
* History screen shows recent captured moments (reverse-chronological).
* Start tag-based pattern summaries (“Most common: stress after work”).

**Sprint 4 (Fine-tune v1 + Red-team)**

* Curate 1–3k anonymized “ideal” dialogues; LoRA/PEFT fine-tune.
* Red-team tests across cultures/ages; update guardrails.

**What you’ll have after Sprint 1**

* A coach that **always validates first**, stays in-bounds, offers **tiny steps**, can **guide 1-minute exercises**, and can **log moments with consent**—all without retraining.

If you share your current backend (OpenAI vs. self-hosted), I can tailor the exact code (SDK calls, model choice, and a ready-made buildSystemPrompt() plus a red-flag detector you can drop in).