Unified Mood-Based Multi-Domain Recommendation System

It takes a free-text mood description and (optionally) user entertainment preferences, then returns **Meal (with recipe)**, **Music playlist**, and **Entertainment** (movie / anime / series) recommendations — all aligned, explained, and tailored.

What you'll get here

- ✅ Problem summary & requirements (brief)
- ✅ Design & architecture (concise)
- ✅ All OpenAI prompts used (clear & editable)
- ✅ A single app.py Streamlit application you can run locally
- ✅ Instructions to run

Problem summary (one sentence)

Design and implement a system that converts a user's natural-language mood description into coherent, multi-domain recommendations for a meal (with a short recipe), a music playlist, and entertainment (movie / anime / series tuned to user's likes), plus brief reasoning explaining alignment.

Design / Architecture (brief)

1. **Mood Analyzer** — extracts axes: energy, valence, stress, sensory hints, and any explicit preferences. Uses heuristics + LLM fallback.
2. **Rulebook (Mood → Anchors)** — human-designed mapping from mood axes to domain anchors (meal types, music styles, movie tone, anime genres). This ensures consistent, interpretable recommendations.
3. **Generators** — each domain (meal, music, entertainment) builds an LLM prompt combining: user mood, anchors, and optional user likes/preferences; it returns structured JSON.
4. **UI** — Streamlit app for quick usage and toggling options.

You can tune MOOD_RULES and prompts to change behavior.

Prompts (you can edit in code)

Prompts are embedded in the code. They are:

- MOOD_PARSE_PROMPT — LLM fallback to parse mood into axes (energy, valence, stress, sensory_preferences).
- MEAL_PROMPT — generate meal title, reasoning, concise recipe JSON.
- MUSIC_PROMPT — generate playlist title, 6 tracks, reasoning.
- ENTERTAINMENT_PROMPT — generate 3 entertainment recommendations (movie/anime/series) prioritized by user likes + reasoning.

Prompts ask the model to return JSON. If model output fails to parse, the app shows raw output for debugging.

Run requirements

pip install streamlit openai

Streamlit run:

streamlit run app.py

Full Streamlit app — save as app.py

"""

Unified Mood → Meal / Music / Entertainment Streamlit App

- Enter mood text

- Optionally provide entertainment likes (genres, shows, actors)

- Choose outputs (meal, music, entertainment)

- App uses a small rulebook + OpenAI LLM to generate structured recommendations + reasoning

Install:

pip install streamlit openai

Run:

streamlit run app.py

Notes:

- Enter your OpenAI API key in the sidebar.

- The app uses the OpenAI Responses API via the official `openai` Python client.

"""

```
import streamlit as st

from openai import OpenAI

import json

import textwrap


st.set_page_config(page_title="Mood → Meal/Music/Entertainment", layout="wide")

st.title("🎭 Mood → Meal · Music · Entertainment")

st.write("Describe your mood and get a meal (with recipe), a music playlist, and entertainment picks (movie/anime/series) — all aligned with your mood and preferences.")


# ------------------------

# Sidebar: API & options

# ------------------------

api_key = st.sidebar.text_input("OpenAI API Key", type="password")

model_choice = st.sidebar.selectbox("Model", ["gpt-4.1", "gpt-4.1-mini", "gpt-o1"], index=0)

st.sidebar.markdown("---")

st.sidebar.write("Mapping strictness controls how much the system adheres to the built-in mood rules vs LLM nuance.")

strictness = st.sidebar.slider("Mapping Strictness", 0, 100, 70)

st.sidebar.write("Higher = more rule-driven anchors; Lower = more LLM freedom.")

st.sidebar.markdown("---")
```

```
st.sidebar.write("Tip: Add entertainment likes (genres, shows, actors) to bias recommendations.")


if not api_key:
    st.warning("Please enter your OpenAI API key in the sidebar to use the app.")
    st.stop()


client = OpenAI(api_key=api_key)


# ------------------------
# Human-designed mood -> anchor rules (editable)
# ------------------------
MOOD_RULES = {
    "energy": {
        "low": {
            "meal": ["warm, light, easy-to-digest (soups, porridges)", "low-prep comfort food"],
            "music": ["slow tempo, ambient, acoustic"],
            "entertainment": ["gentle pacing, feel-good dramas, slice-of-life"]
        },
        "medium": {
            "meal": ["balanced bowls (protein + veg + grain)", "simple pastas, stir-fries"],
            "music": ["mid-tempo pop/indie, chill electronic"],
            "entertainment": ["dramas with moderate pacing, comedies"]
        },
        "high": {
            "meal": ["energizing foods: spicy, crunchy, protein-rich"],
            "music": ["upbeat, fast-tempo genres: dance, rock"],
            "entertainment": ["action, thrillers, fast-paced series"]
```

```
    }
  },
  "valence": {
    "positive": {
      "meal": ["bright flavors, citrus, fresh salads"],
      "music": ["major-key upbeat songs"],
      "entertainment": ["feel-good, visually joyful films / anime"]
    },
    "neutral": {
      "meal": ["familiar, low-risk comfort food"],
      "music": ["ambient, lo-fi, focus playlists"],
      "entertainment": ["documentaries, slice-of-life, low-intensity shows"]
    },
    "negative": {
      "meal": ["grounding comfort food (stews, mashed textures)"],
      "music": ["soft acoustic, soothing instrumental"],
      "entertainment": ["cathartic or uplifting films; healing 'iyashikei' anime"]
    }
  },
  "stress": {
    "low": {
      "meal": ["something creative or celebratory"],
      "music": ["playful or exploratory playlists"],
      "entertainment": ["quirky or thoughtful films"]
    },
    "high": {
      "meal": ["very low-prep, familiar dishes to reduce cognitive load"],
      "music": ["calming steady playlists"],
```

```
        "entertainment": ["comfort shows, short episodic series"]
      }
    }
}


# ------------------------
# Helpers: call LLM
# ------------------------
def call_llm(prompt: str, model: str = None, max_tokens: int = 700):
    model = model or model_choice
    try:
        resp = client.responses.create(model=model, input=prompt,
max_output_tokens=max_tokens)
        return resp.output_text
    except Exception as e:
        return f"ERROR: {e}"


# ------------------------
# Mood analysis
# ------------------------
def heuristic_mood_parse(text: str):
    t = text.lower()
    energy = "medium"
    valence = "neutral"
    stress = "low"
    sensory = ""

    if any(w in t for w in ["tired", "drained", "lethargic", "sleepy", "exhausted"]):
```

```
    energy = "low"
  if any(w in t for w in ["energetic", "pumped", "hyper", "excited", "restless"]):
    energy = "high"

  if any(w in t for w in ["happy", "joyful", "optimistic", "good", "cheerful"]):
    valence = "positive"
  if any(w in t for w in ["sad", "down", "depressed", "lonely", "angry", "upset"]):
    valence = "negative"

  if any(w in t for w in ["anxious", "stressed", "overwhelmed", "nervous", "tense"]):
    stress = "high"

  # sensory hints
  if any(w in t for w in ["warm", "cozy", "hot", "cold", "refreshing", "spicy", "sweet"]):
    sensory = ", ".join([w for w in ["warm", "cozy", "hot", "cold", "refreshing", "spicy",
"sweet"] if w in t])

  return {"energy": energy, "valence": valence, "stress": stress, "sensory": sensory}

# LLM fallback to parse mood in ambiguous cases or to refine heuristics
MOOD_PARSE_PROMPT = """
You are a concise assistant that converts a short mood description into three axes.
Return JSON only:

{
 "energy": "low|medium|high",
 "valence": "negative|neutral|positive",
 "stress": "low|high",
```

```
 "sensory": "comma-separated short keywords or empty"
}

Mood description:
\"\"\"{mood_text}\"\"\"
"""

def parse_mood(mood_text: str):
   parsed = heuristic_mood_parse(mood_text)
   # If the input contains conjunctions or 'but' or the strictness is low, ask LLM to be sure
   ambiguous = ("but" in mood_text.lower()) or ("," in mood_text and "but" in
mood_text.lower())
   if ambiguous or strictness < 90:
      prompt = MOOD_PARSE_PROMPT.format(mood_text=mood_text)
      out = call_llm(prompt)
      try:
         j = json.loads(out)
         # sanitize values
         for k in ("energy", "valence", "stress", "sensory"):
            if k in j and j[k]:
               parsed[k] = j[k]
      except Exception:
         # keep heuristic if parsing fails, but attach raw output to notes
         parsed["parse_debug"] = out[:800]
   return parsed

# -------------------------
# Anchors: gather suggested anchor texts from rules
```

```
# ------------------------
def anchors_for(parsed, domain):
    anchors = []
    for axis in ("energy", "valence", "stress"):
        val = parsed.get(axis)
        try:
            domain_anchors = MOOD_RULES[axis][val][domain]
            anchors.extend(domain_anchors)
        except Exception:
            continue
    # include sensory if present
    if parsed.get("sensory"):
        anchors.append(parsed["sensory"])
    # dedupe and limit
    seen = []
    for a in anchors:
        if a not in seen:
            seen.append(a)
    return seen[:6]


# ------------------------
# Prompts for generators
# ------------------------
MEAL_PROMPT = """
You are a friendly culinary assistant. Input:
- mood_text: {mood}
- mood_anchors: {anchors}
```

- constraint: keep prep time under 30 minutes unless mood explicitly calls for slow cooking.

Produce JSON ONLY:

{{

"meal": "<short dish title>",

"reasoning": "<1-2 short sentences why this fits (mention energy/valence/stress)>",

"recipe": {{

"time": "<approx minutes>",

"ingredients": ["...","..."],

"steps": ["step 1..","step 2.."]

}}

}}

"""

MUSIC_PROMPT = """

You are a concise music recommender. Input:

- mood_text: {mood}

- mood_anchors: {anchors}

Produce JSON ONLY:

{{

"playlist": "<short playlist title>",

"tracks": ["Artist - Song", "Artist - Song", "..."] ,

"reasoning": "<one sentence>"

}}

"""

```
ENTERTAINMENT_PROMPT = """
You are a helpful entertainment recommender.
Input:
- mood_text: {mood}
- mood_anchors: {anchors}
- user_likes: {likes}   # a short user preference string; can be empty

Produce JSON ONLY: a list of three recommendations (movie / anime / series) ranked 1..3.
Each recommendation must include:
- title (title + year if available)
- type ("movie" / "anime" / "series")
- one-sentence synopsis (no spoilers)
- two-sentence reasoning why it suits the mood and how it aligns with user_likes (if provided)
- suggested viewing context (e.g., "alone, with friends, snack suggestions, time of day")

Return JSON array: [ {{...}}, {{...}}, {{...}} ]
"""

# -------------------------
# Generator functions
# -------------------------
def generate_meal(mood_text, parsed):
    anchors = anchors_for(parsed, "meal")
    anchor_text = "; ".join(anchors) if anchors else "general comforting meal"
    prompt = MEAL_PROMPT.format(mood=mood_text, anchors=anchor_text)
    out = call_llm(prompt)
```

```
    try:
        return json.loads(out)
    except Exception:
        return {"meal": "Error generating meal", "reasoning": out[:600], "recipe": {"time": "",
"ingredients": [], "steps": []}}


def generate_music(mood_text, parsed):
    anchors = anchors_for(parsed, "music")
    anchor_text = "; ".join(anchors) if anchors else "mood-aligned music"
    prompt = MUSIC_PROMPT.format(mood=mood_text, anchors=anchor_text)
    out = call_llm(prompt)
    try:
        return json.loads(out)
    except Exception:
        return {"playlist": "Error", "tracks": [], "reasoning": out[:600]}


def generate_entertainment(mood_text, parsed, user_likes):
    anchors = anchors_for(parsed, "entertainment")
    anchor_text = "; ".join(anchors) if anchors else "mood-aligned entertainment"
    likes_text = user_likes if user_likes else ""
    prompt = ENTERTAINMENT_PROMPT.format(mood=mood_text, anchors=anchor_text,
likes=likes_text)
    out = call_llm(prompt, max_tokens=900)
    try:
        j = json.loads(out)
        # Expecting list
        if isinstance(j, dict):
            # Some models return an object; wrap
```

```
        return [j]
    return j
  except Exception:
    return [{"title": "Error generating entertainment", "type": "error", "synopsis": out[:800], "reasoning": "", "context": ""}]


# ------------------------
# UI: Inputs
# ------------------------
st.subheader("Input")
mood_input = st.text_area("Describe your current mood (e.g., 'slightly drained but hopeful')", height=120)
user_likes = st.text_input("Entertainment likes (optional) — genres, shows, actors, or examples", placeholder="e.g., 'romcoms, Miyazaki, sci-fi, The Crown'")
col1, col2, col3 = st.columns(3)
with col1:
  want_meal = st.checkbox("Meal + Recipe", value=True)
with col2:
  want_music = st.checkbox("Music Playlist", value=True)
with col3:
  want_ent = st.checkbox("Entertainment Picks (movie/anime/series)", value=True)

if st.button("Generate Recommendations"):
  if not mood_input.strip():
    st.error("Please enter a mood description.")
    st.stop()

  with st.spinner("Parsing mood..."):
    parsed = parse_mood(mood_input)
```

```
st.markdown("### 🧭 Parsed Mood Axes")
st.write(parsed)

# Generate outputs
if want_meal:
    with st.spinner("Generating meal..."):
        meal_obj = generate_meal(mood_input, parsed)
    st.markdown("## 🍽 Meal Recommendation")
    if meal_obj.get("meal"):
        st.markdown(f"**{meal_obj['meal']}**")
        st.markdown(f"**Reasoning:** {meal_obj.get('reasoning','')}")
        rec = meal_obj.get("recipe", {})
        st.markdown(f"**Estimated time:** {rec.get('time','')}")
        st.markdown("**Ingredients:**")
        for ing in rec.get("ingredients", []):
            st.write(f"- {ing}")
        st.markdown("**Steps:**")
        for i, s in enumerate(rec.get("steps", []), start=1):
            st.write(f"{i}. {s}")
    else:
        st.write(meal_obj)

if want_music:
    with st.spinner("Generating playlist..."):
        music_obj = generate_music(mood_input, parsed)
    st.markdown("## 🎧 Music / Playlist")
```

```
    if music_obj.get("playlist"):
        st.markdown(f"**{music_obj['playlist']}**")
        st.markdown("**Tracks:**")
        for t in music_obj.get("tracks", []):
            st.write(f"- {t}")
        st.markdown(f"**Reasoning:** {music_obj.get('reasoning','')}")
    else:
        st.write(music_obj)


if want_ent:
    with st.spinner("Generating entertainment picks..."):
        ent_list = generate_entertainment(mood_input, parsed, user_likes)
    st.markdown("## 🎬 Entertainment Picks")
    for idx, rec in enumerate(ent_list, start=1):
        st.markdown(f"### {idx}. {rec.get('title','(no title)')} — {rec.get('type','')}")
        st.write(f"Synopsis: {rec.get('synopsis','')}")
        st.markdown(f"**Why it fits:** {rec.get('reasoning','')}")
        st.write(f"Viewing context: {rec.get('context','')}")
        st.markdown("---")


st.write("Tip: adjust 'Mapping Strictness' in the sidebar to make outputs more rule-guided (higher) or more LLM-driven (lower).")


# Footer
st.markdown("---")
st.write("Built with ❤️ — a unified mood→recommendation demo using OpenAI models. Edit the rulebook (MOOD_RULES) and prompts inline to customize behavior.")
```

How it works (quick)

1. Enter mood text and any entertainment likes.
2. The app heuristically parses mood into axes; if ambiguous (or strictness allows), it asks the LLM to refine the parsing.
3. For each domain, anchors are gathered from MOOD_RULES and passed into the LLM prompt which returns structured JSON.
4. App displays recommendations with short reasoning and recipe where applicable.

Customization tips

- Edit MOOD_RULES to change how moods anchor to domain features.
- Tweak prompts (MEAL_PROMPT, MUSIC_PROMPT, ENTERTAINMENT_PROMPT) for different verbosity, cuisine styles, or to constrain tracks to popular / obscure artists.
- Add time-of-day or dietary restrictions fields in the UI and include them in prompts.

Safety / Practical notes

- Outputs are creative recommendations, not professional advice (medical or dietary). If you need dietary restrictions or allergies respected, add input fields and pass them to the meal prompt.
- API calls incur cost per model usage.
