src/ui_gradio.py — Full Code & Run Instructions

This document contains the complete implementation of `src/ui_gradio.py` for the AI Virtual Interviewer project, along with instructions to run it.

## Full Code: src/ui_gradio.py

```
"""
Simple & readable Gradio UI for the AI Interviewer.

What it does:
- Lets the user pick a role and enter their name.
- Shows a small live webcam preview (top-right) so they know the camera is on.
- Records answers via microphone, transcribes to text, then scores:
    - Answer rating (TF-IDF cosine + keyword coverage)
    - Behavior rating (audio proxies always, optional webcam metrics)
- Shows friendly per-question feedback + a smile ■
- Saves a Markdown report at the end and exposes it for download.
"""

import os
import json
import math
import numpy as np
import gradio as gr
import speech_recognition as sr

from utils import load_questions, append_session_log, now, REPORTS
from scoring import score_answer
from report import save_report
from behavior import audio_proxies_from_pcm, behavior_rating, capture_webcam_metrics


# -------------------------- helpers --------------------------

def _resample_linear(wave: np.ndarray, src_hz: int, dst_hz: int = 16000) -> np.ndarray:
    """Very small, dependency-free linear resample to 16 kHz mono float32."""
    if src_hz == dst_hz or len(wave) == 0:
        return wave.astype(np.float32)
    ratio = dst_hz / float(src_hz)
    new_len = int(math.ceil(len(wave) * ratio))
    x_old = np.linspace(0, 1, len(wave), endpoint=False)
    x_new = np.linspace(0, 1, new_len, endpoint=False)
    return np.interp(x_new, x_old, wave).astype(np.float32)


def transcribe_from_gradio(numpy_audio, sample_rate_hz):
    """Convert Gradio's (numpy_audio, sample_rate) → (transcript_text, pcm_bytes_16k_16bit)."""
    if numpy_audio is None:
        return "", b""

    # Ensure mono float32 [-1, 1]
    wav = np.asarray(numpy_audio, dtype=np.float32)
    if wav.ndim == 2:
        wav = wav.mean(axis=1)
    wav = _resample_linear(wav, sample_rate_hz, 16000)
    wav = np.clip(wav, -1.0, 1.0)

    # Convert to 16-bit PCM bytes
    pcm_int16 = (wav * 32767.0).astype(np.int16)
    pcm_bytes = pcm_int16.tobytes()

    # SpeechRecognition transcription
    recognizer = sr.Recognizer()
    audio = sr.AudioData(pcm_bytes, 16000, 2)
    try:
        text = recognizer.recognize_google(audio)
    except Exception:
        text = ""
    return text, pcm_bytes


# --------------------------- callbacks ---------------------------

def start_session(role, name, n_questions):
    if not role or not name:
        return gr.update(value="■■ Please select a role and enter your name."), None, None, None

    questions = load_questions(role)[:max(1, int(n_questions))]
    session = {
        "role": role,
        "name": name.strip(),
        "ts": now(),
        "items": questions,
        "idx": 0,
        "qa": []
    }
    first_q = questions[0]["question"] if questions else "(no question)"
    return gr.update(value=f"**Q1:** {first_q}"), session, "", ""


def submit_answer(audio_tuple, role, session, use_webcam_metrics):
    if session is None:
        return "■■ Click Start first.", session, None, None, None

    i = session["idx"]
    if i >= len(session["items"]):
        return "■ Interview finished. Click 'Finish & Save Report'.", session, None, None, None

    item = session["items"][i]

    if audio_tuple is None or audio_tuple[0] is None:
        return "■■ Record your answer with the mic button.", session, None, None, None

    numpy_audio, sample_rate = audio_tuple
    text, pcm_bytes = transcribe_from_gradio(numpy_audio, sample_rate)

    ans = score_answer(text, item)
    audio_feats = audio_proxies_from_pcm(pcm_bytes)

    webcam = {"face_visible_ratio": 0, "centered_ratio": 0, "avg_brightness": 0, "motion_jitter": 0, "ey
    if use_webcam_metrics:
        webcam = capture_webcam_metrics(seconds=6)

    beh = behavior_rating(webcam, audio_feats)

    session["qa"].append({
        "question": item["question"],
        "answer": text,
        "answer_score": ans,
        "behavior_raw": {"webcam": webcam, "audio": audio_feats},
        "behavior": beh
    })
    session["idx"] += 1

    # Feedback
    lines = [
        f"**Your transcript:** {text or '(empty)'}",
        f"**Answer rating:** {ans['total']:.2f} (rel {ans['relevance']:.2f}, kw {ans['keyword_coverage']
        f"**Behavior rating:** {beh['behavior_total']:.2f}",
    ]
    if ans["tips"] or beh["tips"]:
        lines.append("**Tips:**")
        lines += [f"- {t}" for t in ans["tips"] + beh["tips"]]
    lines.append("■")
    feedback_md = "\n\n".join(lines)
    if session["idx"] < len(session["items"]):
        next_q = session["items"][session["idx"]]["question"]
        next_text = f"**Q{session['idx']+1}:** {next_q}"
    else:
        next_text = "■ All questions answered. Click **Finish & Save Report**."

    return next_text, session, feedback_md, None, None


def finish_and_save(session):
    if session is None or not session["qa"]:
        return "■■ Nothing to save yet.", None

    avg_ans = sum(q["answer_score"]["total"] for q in session["qa"]) / len(session["qa"])
    avg_beh = sum(q["behavior"]["behavior_total"] for q in session["qa"]) / len(session["qa"])

    append_session_log({
        "ts": session["ts"],
        "role": session["role"],
        "name": session["name"],
        "avg_answer": round(avg_ans, 2),
        "avg_behavior": round(avg_beh, 2),
    })

    out_path = save_report(session, f"{session['role']}_{session['name'].replace(' ','_')}.md")
    msg = f"■ Report saved: `{out_path.name}`  \nAverage Answer: **{avg_ans:.2f}**  |  Average Behavior:
    return msg, str(out_path)


# --------------------------- Gradio app ---------------------------

def make_app():
    qpath = os.path.join(os.path.dirname(__file__), "..", "data", "questions.json")
    roles = list(json.load(open(qpath, encoding="utf-8")).keys())

    with gr.Blocks(title="AI Virtual Interviewer") as demo:
        gr.Markdown("# ■ AI Virtual Interviewer — Friendly Mode")

        with gr.Row():
            with gr.Column(scale=4):
                role = gr.Dropdown(choices=roles, value="java_developer", label="Role")
                name = gr.Textbox(label="Your name", placeholder="e.g., Ananya")
                n_questions = gr.Slider(1, 6, value=3, step=1, label="Number of questions")
                use_webcam = gr.Checkbox(False, label="Include webcam behavior metrics (optional)")

            with gr.Column(scale=1):
                gr.Markdown("**Webcam Preview** (just to show you’re being captured)")
                cam_preview = gr.Video(source="webcam", streaming=True, height=160, interactive=False)

        start_btn = gr.Button("Start")
        question_md = gr.Markdown("Pick a role, enter your name, then click Start.")
        mic = gr.Audio(source="microphone", type="numpy", label="Record your answer")
        submit_btn = gr.Button("Submit answer")
        feedback_md = gr.Markdown("")
        finish_btn = gr.Button("Finish & Save Report")
        result_md = gr.Markdown("")
        download_files = gr.Files(label="Download your report (after Finish)", interactive=False)

        session_state = gr.State()

        start_btn.click(start_session, [role, name, n_questions], [question_md, session_state, feedback_
        submit_btn.click(submit_answer, [mic, role, session_state, use_webcam], [question_md, session_st
        finish_btn.click(finish_and_save, session_state, [result_md, download_files])

        gr.Markdown("— built with ❤■ using Python, Gradio, scikit-learn, and OpenCV.")

    return demo


if __name__ == "__main__":
    app = make_app()
    app.launch()
```

## How to Run

```
# 1. Install Gradio (if not already installed)
pip install gradio

# 2. Launch the Gradio UI app
python -m src.ui_gradio

# 3. Open the link shown in the terminal (usually http://127.0.0.1:7860)
#    - Left side: select role, enter name, choose number of questions
#    - Right side: see live webcam preview
#    - Record answer with mic, submit, and receive feedback
#    - At the end, click 'Finish & Save Report' to download the report
```