Human Audio Mixer: System and Personalization Details

Generated: 2025-12-01

1) System Overview

- Web frontend (React) + FastAPI backend. Demucs splits audio into 4 stems; frontend mixes stems with user gains and plays returned WAV.
- Personalization is client-side using TensorFlow.js: learns per-user gain preferences from extracted audio features + rule-based genre vector. Weights live in browser localStorage (key userModel_<userId>).

2) Backend: Separation and Mixing (backend/main.py)

- /start_separation: saves upload to backend/separated/input_<id>.wav, resets demucs_progress.json, runs Demucs htdemucs in a thread, writes stems drums/bass/other/vocals to backend/separated.
- Progress file demucs_progress.json holds {"progress": p} with p in [0,1]; /separation_progress reports status completed when p>=1.
- Separation path: mono duplicated to stereo -> tensor shape [1,2,N] -> apply_model(htdemucs) -> stems saved.
- /mix: loads stems, applies per-stem gains, aligns lengths, sums, normalizes by peak, saves to backend/mixed_outputs/final_mix_<id>.wav and returns /mixed_outputs/<id>.wav for playback.
- Gain math: linear_gain = 10^(gain_db/20); mix is peak-normalized: mix = mix / max(|mix|).

3) Frontend Feature Extraction (frontend/src/utils/audioFeatures.js)

- Uses Web Audio API to decode upload and compute 23 features: duration, sampleRate, RMS, zeroCrossingRate, spectralCentroid, spectralRolloff (85%% energy), spectralFlux, tempo heuristic, 5 MFCC-like log-energy bands, plus 10-elem genre vector.
- Normalization (examples): duration_norm = min(duration/300,1); sampleRate_norm = sr/48000; rms_norm = min(rms*10,1); tempo_norm = (tempo-60)/120 (from heuristic BPM peaks: bpm = peaks/duration*60, clamped 60-180).
- Genre heuristic: soft one-hot (length 10) based on tempo/centroid/flux/RMS; defaults to pop-like prior when unknown.
- featuresToVector concatenates normalized scalars + genre vector => 23-D input to the preference model.

4) Personalization Model (frontend/src/utils/userPreferenceModel.js)

- Architecture: tf.sequential dense net, input 23 -> Dense64(relu) -> Dense32(relu) -> Dense16(relu) -> Dense4(tanh).
- Output mapping: tanh gives y in [-1,1]; per-stem gain_db = y*18 - 6, then clamped to [-24,12] dB.
- Training target mapping: normalized_gain = (gain_db + 6)/18 to fit tanh range.
- Online train (single song): epochs=5, batch=1, Adam lr=0.001. Batch retrain: epochs=10, batch<=8 over all saved songs with features+gains.
- Prediction flow: ensure init (loads weights if present) -> tensor2d([features]) -> model.predict -> map to dB.
- Weight storage: model.getWeights() -> arrays -> JSON in localStorage under key userModel_<userId>; loadWeights rebuilds tensors from that JSON. Model is recreated and

weights restored on init.
- Saved songs: per-user array in localStorage key audioMixSongs; each entry stores title, gains, and feature vector for retraining.

5) Frontend Mixing UI (frontend/src/components/Mixer.js)
- Upload triggers /start_separation, then polls /separation_progress. Progress bar uses returned progress in [0,1].
- Sliders labeled with stem emojis: Vocals (ð⁄), Drums (ð¥), Bass (ð ¶), Other (ð §); values sent to /mix for rendered playback. Real-time changes debounced and crossfaded to new mix URL.

6) Data Persistence Locations
- Browser localStorage: userModel_<userId> (TF.js weights), audioMixSongs (saved mixes with gains+features), museTheme (UI preference).
- Backend filesystem: backend/separated/ (latest stems), backend/mixed_outputs/ (rendered mixes), backend/demucs_progress.json (progress state), backend/uploads/ (raw uploads), backend/user_data/ (reserved).

7) Key Formulas
- dB to linear: g_linear = 10^(g_db/20).
- Model output to dB: g_db = clamp(tanh_out*18 - 6, -24, 12).
- Training target: norm_gain = (g_db + 6)/18 mapped to [-1,1].
- Tempo heuristic: bpm = (peak_count / duration_seconds) * 60, clamped to [60,180].