User Browser

├─ Microphone (Web Audio API)

├─ Capture Audio Frames

└─ Send audio snippet → Backend Server

Backend Server

├─ Receives audio

├─ Processes audio (preprocessing: MFCC/spectrogram)

├─ AI Model predicts bark → returns result

└─ Server responds → Browser triggers voice playback

Browser (Frontend) Responsibilities

Capture audio:

Use Web Audio API to record short snippets of audio (e.g., 1–2 seconds).

Send audio to backend:

* Convert audio blob to Base64 or ArrayBuffer.
* Send via HTTP POST or WebSocket for real-time streaming.

Receive model prediction:

* Backend responds with JSON:

Play voice response:

* If barkDetected is true, trigger pre-recorded voice or Web Speech API.
* Backend responds with JSON:

**3. Backend Responsibilities**

* **Receive audio snippets** from the frontend.
* **Preprocess audio:**
  + Convert raw audio to **spectrograms or MFCC features**.
* **Run AI model inference:**
  + Model can be **TensorFlow, PyTorch, or ONNX**.
  + Returns probability of dog bark.
* **Send result back to frontend** immediately.
* **Optional logging:**
  + Store detection timestamps and user settings in a database.
* **Example Backend Tech Stack:**
  + **Server:** Node.js/Express, Python Flask, or FastAPI
  + **AI Model:** TensorFlow/PyTorch model loaded in Python
  + **Database (optional):** MongoDB, PostgreSQL, Firebase

**4. Voice Playback**

* Can be handled entirely on the frontend:
  + Pre-recorded audio files (MP3/WAV)
  + Or Web Speech API for TTS
* This keeps the server lightweight and avoids streaming audio back for TTS.

💡 **Recommendation for your web app**:

* **Phase 1:** Don’t implement login/register. Keep it simple and offline first.
* **Phase 2 (optional):** Add login if you later want cloud features, multi-device access, or user-specific history.

User Browser

------------

1. UI

- Home Screen:

• Microphone

• Status

• Voice Mode

- Settings:

• Sensitivity

• Voice Select

- Logs Panel

2. Audio Capture

- Web Audio API

- Records audio in 1–2 sec frames

3. Feature Extraction

- Meyda.js / Web Audio API

- Compute MFCC / Spectrogram

4. ML Inference (Client-Side)

- TensorFlow.js

- Detects bark

5. Voice Response

- Pre-recorded audio (MP3/WAV) OR

- Web Speech API (TTS)

6. Local Storage

- LocalStorage / IndexedDB

- Saves settings and logs

Service Worker

--------------

- Caches all app files (HTML, JS, ML model, audio)

- Ensures offline availability

**How it Works (Step-by-Step)**

1. **App loads** → Service Worker caches all assets for offline use.
2. **User starts listening** → Web Audio API captures microphone input.
3. **Audio frames are processed** → Feature extraction (MFCC/spectrogram).
4. **TensorFlow.js runs ML model** → Detects whether a dog is barking.
5. **If bark detected** → Plays human voice via pre-recorded audio or TTS.
6. **Settings and detection logs** → Stored locally in LocalStorage or IndexedDB.
7. **Offline operation** → No internet needed; all processing and storage is on-device.

**Advantages of This Setup**

* Works fully offline.
* No user login required.
* Low latency because processing happens locally.
* Easy to expand later with optional backend or cloud sync.

A screen shot of a phone

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.